

NELSON MIMURA GONZALEZ

**MPSF: CLOUD SCHEDULING FRAMEWORK
FOR DISTRIBUTED WORKFLOW EXECUTION**

**MPSF: UM ARCABOUÇO PARA ESCALONAMENTO
EM COMPUTAÇÃO EM NUVEM PARA
EXECUÇÃO DISTRIBUÍDA DE FLUXOS DE TRABALHO**

São Paulo

2017

NELSON MIMURA GONZALEZ

**MPSF: CLOUD SCHEDULING FRAMEWORK
FOR DISTRIBUTED WORKFLOW EXECUTION**

**MPSF: UM ARCABOUÇO PARA ESCALONAMENTO
EM COMPUTAÇÃO EM NUVEM PARA
EXECUÇÃO DISTRIBUÍDA DE FLUXOS DE TRABALHO**

Tese apresentada à Escola Politécnica da
Universidade de São Paulo para obtenção do
Título de Doutor em Ciências.

São Paulo

2017

NELSON MIMURA GONZALEZ

**MPSF: CLOUD SCHEDULING FRAMEWORK
FOR DISTRIBUTED WORKFLOW EXECUTION**

**MPSF: UM ARCABOUÇO PARA ESCALONAMENTO
EM COMPUTAÇÃO EM NUVEM PARA
EXECUÇÃO DISTRIBUÍDA DE FLUXOS DE TRABALHO**

Tese apresentada à Escola Politécnica da
Universidade de São Paulo para obtenção do
Título de Doutor em Ciências.

Área de Concentração:
Engenharia de Computação

Orientadora:
Tereza Cristina Melo de Brito Carvalho

São Paulo
2017

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, ____ de _____ de _____

Assinatura do autor: _____

Assinatura da orientadora: _____

Catálogo-na-publicação

Gonzalez, Nelson Mimura

MPSF: Um Arcabouço para Escalonamento em Computação em Nuvem para Execução Distribuída de Fluxos de Trabalho / N. Gonzalez – versão corr. – São Paulo, 2017.

304 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Computação em nuvem 2.Gerenciamento de Recursos I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

"Anyone who has never made a mistake has never tried anything new."

Albert Einstein

AGRADECIMENTOS

À minha orientadora, Tereza Cristina Melo de Brito Carvalho, pela orientação e ajuda desde a época da graduação até hoje. Em quase dez anos de convivência aprendi com ela a importância da dedicação e da criação de laços de trabalho e de amizade que são importantes não só para cumprir prazos, mas para dar sentido ao que fazemos na vida. Neste pouco mais de um ano longe das pessoas com quem trabalhei e convivi por tanto tempo percebi o quanto isto faz falta e o quão importante é estar próximo das pessoas que gostamos. Embora façamos novas amizades, nunca esquecemos ou deixamos de sentir falta dessas pessoas. Muito obrigado.

Ao meu coorientador, Charles Christian Miers, por ser uma fonte de apoio constante ao longo dessa jornada. Minha primeira entrevista de emprego formal foi com ele no LARC e desde então essa relação evoluiu para uma amizade que ousou comparar a de um irmão (mais velho 😊). Sempre admirei seus valores, sua integridade, e sua honra em todos os tratos. Se eu conseguir chegar a uma fração da sua fibra moral me darei por satisfeito.

Aos meus pais, Vilma e Nelson, por tudo o que me proporcionaram e continuam me proporcionando na vida. Meus pais me ensinaram o valor do empenho, do trabalho, e do estudo como formas essenciais para crescer e se tornar uma pessoa que possa, como muitos desejam, fazer a diferença. Espero que eu esteja nesse caminho e espero que eles tenham por mim o mesmo orgulho que eu tenho por eles de serem, acima de tudo, minha fonte de inspiração para viver e caminhar em frente. Muito obrigado.

Finalmente, gostaria de agradecer a todos os amigos do LARC, da Ericsson, e agora da IBM. Que estas relações perdurem e se fortaleçam cada vez mais, e que eu possa contar com eles assim como eles poderão sempre contar comigo.

RESUMO

A computação em nuvem representa um paradigma de computação distribuída que ganhou destaque devido a aspectos relacionados à obtenção de recursos sob demanda de modo elástico e dinâmico. Estas características são consideravelmente desejáveis para a execução de tarefas relacionadas a fluxos de trabalho científicos, que exigem grande quantidade de recursos computacionais e grande fluxo de dados. Uma das principais questões neste sentido é como gerenciar os recursos de uma ou mais infraestruturas de nuvem para execução de fluxos de trabalho de modo a otimizar a utilização destes recursos e minimizar o tempo total de execução das tarefas. Quanto mais complexa a infraestrutura e as tarefas a serem executadas, maior o risco de estimar incorretamente a quantidade de recursos destinada para cada tarefa, levando a prejuízos não só em termos de tempo de execução como também financeiros. Cenários inerentemente mais complexos como nuvens híbridas e múltiplas nuvens raramente são considerados em soluções existentes de gerenciamento de recursos para nuvens. Além destes fatores, a maioria das soluções não oferece mecanismos claros para tratar de fluxos de trabalho com alta intensidade de dados, característica cada vez mais proeminente em fluxos de trabalho moderno. Neste sentido, esta proposta apresenta MPSF, uma solução de gerenciamento de recursos baseada em múltiplas fases de gerenciamento baseadas em mecanismos dinâmicos de alocação de tarefas. MPSF define modelos para descrever e caracterizar fluxos de trabalho e recursos de modo a suportar cenários simples e complexos, como nuvens híbridas e nuvens integradas. MPSF também define modelos de desempenho e confiabilidade para melhor distribuir a carga e para combater os efeitos de possíveis falhas que possam ocorrer no sistema. Por fim, MPSF define um arcabouço e um arquitetura que integra todos estes componentes de modo a definir uma solução que possa ser implementada e utilizada em cenários reais. Testes experimentais indicam que MPSF não só é capaz de prever com maior precisão a duração da execução de tarefas, como também consegue otimizar a execução das mesmas, especialmente para tarefas que demandam alto poder computacional e alta quantidade de dados.

Palavras-chave: Computação em nuvem. Gerenciamento de recursos.

ABSTRACT

Cloud computing represents a distributed computing paradigm that gained notoriety due to its properties related to on-demand elastic and dynamic resource provisioning. These characteristics are highly desirable for the execution of workflows, in particular scientific workflows that required a great amount of computing resources and that handle large-scale data. One of the main questions in this sense is how to manage resources of one or more cloud infrastructures to execute workflows while optimizing resource utilization and minimizing the total duration of the execution of tasks (makespan). The more complex the infrastructure and the tasks to be executed are, the higher the risk of incorrectly estimating the amount of resources to be assigned to each task, leading to both performance and monetary costs. Scenarios which are inherently more complex, such as hybrid and multiclouds, rarely are considered by existing resource management solutions. Moreover, a thorough research of relevant related work revealed that most of the solutions do not address data-intensive workflows, a characteristic that is increasingly evident for modern scientific workflows. In this sense, this proposal presents MPSF, the Multiphase Proactive Scheduling Framework, a cloud resource management solution based on multiple scheduling phases that continuously assess the system to optimize resource utilization and task distribution. MPSF defines models to describe and characterize workflows and resources. MPSF also defines performance and reliability models to improve load distribution among nodes and to mitigate the effects of performance fluctuations and potential failures that might occur in the system. Finally, MPSF defines a framework and an architecture to integrate all these components and deliver a solution that can be implemented and tested in real applications. Experimental results show that MPSF is able to predict with much better accuracy the duration of workflows and workflow phases, as well as providing performance gains compared to greedy approaches.

Keywords: Cloud computing. Resource management.

CONTENTS

List of figures	10
List of tables	15
List of acronyms	17
1 Introduction	18
1.1 Problem and Motivation	20
1.2 Objectives and Contributions	21
1.3 Method.....	25
1.4 Document Organization.....	26
2 Cloud Resource Management	27
2.1 Resource Management: Definition and Concepts	28
2.1.1 Definition by (SINGH; CHANA, 2015b)	28
2.1.2 Definition by (JENNINGS; STADLER, 2015)	31
2.1.3 Definition by (MANVI; SHYAM, 2014).....	33
2.1.4 Other Definitions	35
2.1.5 Intercorrelation and Consolidation.....	36
2.2 Related Work.....	41
2.2.1 Methodology.....	41
2.2.2 Identified Related Work and Surveys	42
2.2.2.1 A Survey of Various Workflow Scheduling Algorithms in Cloud Environment	42
2.2.2.2 Towards inter-cloud schedulers: A survey of meta-scheduling approaches	44
2.2.2.3 Workflow scheduling in cloud: a survey	47
2.2.2.4 A survey on resource allocation in high performance distributed computing systems.....	55
2.2.2.5 Cloud resource provisioning: survey, status and future research directions & A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges	56
2.2.3 Taxonomy Analysis and Consolidation	60
2.2.3.1 A Survey of Various Workflow Scheduling Algorithms in Cloud Environment	60
2.2.3.2 Towards inter-cloud schedulers: A survey of meta-scheduling approaches	61
2.2.3.3 Workflow scheduling in cloud: a survey	62
2.2.3.4 Cloud resource provisioning: survey, status and future research directions	63
2.2.3.5 Consolidation.....	69
2.2.4 Summary of Results.....	72
2.3 Gaps and Challenges	82
2.3.1 Data-Intensive Workflows	82

2.3.2	Hybrid and Multicloud Scenarios	83
2.3.3	Rescheduling and Performance Fluctuations	84
2.4	Problem Definition	85
2.5	Chapter Considerations	85
3	MPSF	87
3.1	Solution Requirements	88
3.1.1	Problems Revisited	88
3.1.2	Solution Requirements.....	89
3.1.3	Mapping Solution Requirements to Problems	90
3.2	Solution Overview	92
3.2.1	Features.....	93
3.3	Performance Model	97
3.3.1	Problem Introduction	98
3.3.2	Simple Model for Data Distribution	98
3.3.3	Exemplar Utilization of Simple Model.....	104
3.3.4	Experimental Verification of Simple Model.....	106
3.3.5	Enhanced Model for Data Distribution.....	110
3.3.6	Experimental Verification of Enhanced Model	118
3.3.7	Final Model for Data Distribution	122
3.3.7.1	Shared Data and Chunk Data.....	122
3.3.7.2	Distribution of Shared Data	124
3.3.7.3	Distribution of Chunk Data	133
3.3.7.4	Equalization for $tp \geq tr + tw$	136
3.3.7.5	Pipelining and Phase Combination for $tp < tr + tw$	138
3.3.8	Probabilistic Model for Performance Fluctuations	140
3.3.8.1	Correction Factors	141
3.3.8.2	Bayesian System of Multiple Hypotheses and Multiple Evidences.....	142
3.3.8.3	Calculating the Correction Factors	144
3.3.8.4	Using the Correction Factors	145
3.3.8.5	Building and Updating the Bayesian System	147
3.3.9	Comparison to Literature	148
3.4	Requirement Coverage	149
3.5	Chapter Considerations	150
4	Experiments and Analysis	152
4.1	Experimental Platform and Scenarios	152
4.2	Implementation.....	154

4.3	Selected Benchmarks.....	155
4.3.1	AXPY	155
4.3.2	GEMM.....	156
4.3.3	WAVG.....	157
4.4	Compared Methods	158
4.5	Results	159
4.5.1	Scenario A: Private Cloud (Gauss).....	159
4.5.2	Scenario B: Hybrid Cloud (Gauss and Vulcan).....	162
4.5.3	Scenario C: Multicloud (Gauss and Mamona).....	164
4.6	Analysis	165
4.7	Chapter Considerations	168
5	Final Considerations	169
5.1	Related Work and Challenges	170
5.2	Proposed Framework and Contributions	172
5.3	Experimental Results.....	178
5.4	Limitations.....	179
5.5	Future Directions and Work	181
	References.....	182
	Glossary	200
	Appendix A – Further Analysis of Related Work	201
A.1	Selection Method.....	201
A.2	Analysis	201
A.2.1	A Particle Swarm Optimization-based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments.....	201
A.2.2	Scheduling Scientific Workflows Elastically for Cloud Computing	203
A.2.3	A Multiple QoS Constrained Scheduling Strategy of Multiple Workflows for Cloud Computing 203	
A.2.4	A Federated Model for Scheduling in Wide-Area Systems.....	204
A.2.5	An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem With Various QoS Requirements.....	205
A.2.6	Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds.....	205
A.2.7	A Multi-Objective Approach for Workflow Scheduling in Heterogeneous Environments	209
A.2.8	Cost- and Deadline-Constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds.....	209
A.2.9	A low-cost rescheduling policy for efficient mapping of workflows on grid systems.....	211

A.2.10	Dependency-based Risk Evaluation for Robust Workflow Scheduling	213
A.2.11	Fault-Tolerant Workflow Scheduling Using Spot Instances on Clouds	213
A.2.12	HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds	215
A.2.13	Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka216	
A.3	Summary of Selected Related Work	217
Appendix B	– Workflow Characterization	219
B.1	Characterization Primitives	219
B.1.1	Input Primitive	219
B.1.2	Split Primitive	223
B.1.3	Compute Primitive	227
B.1.4	Merge Primitive	230
B.1.5	Output Primitive	232
B.2	Branching and Multiple Paths	233
B.3	Nesting and Workflow Composition	238
B.4	Transfer and Computation Costs	241
B.4.1	General Notation	241
B.4.2	Cost Functions	242
B.4.3	Framework Automation	245
B.4.4	Data Structure Notation	246
B.5	Summary of Primitives and Transitions	247
B.5.1	Primitives	247
B.5.2	Transitions	250
B.6	Mapping Primitives	252
B.7	Comparison to Literature	259
Appendix C	– Resource Characterization	261
C.1	Resource Hierarchy	261
C.2	Execution Spaces	263
C.3	Segmentation and Logical Hierarchies	265
C.4	Resource Properties	267
C.5	Comparison to Literature	269
Appendix D	– Reliability Model	270
D.1	Reliability Concepts and Metrics	272
D.2	Probabilistic Reliability Model for Workflow Execution	276
D.3	Task Cost from Reliability Perspective	279

D.4	Comparison to Literature	279
Appendix E	– Scheduling Framework	280
E.1	Scheduling Phases	280
E.2	Scheduling Triggers and Thresholds	286
E.3	Operational Aspects.....	289
Appendix F	– Architecture	291
F.1	Architectural Requirements	291
F.2	Architectural Components	292
F.2.1	Dashboard	294
F.2.2	Environment Monitoring	295
F.2.3	Resource Discovery	295
F.2.4	Resource Characterization	296
F.2.5	Execution Space Management	297
F.2.6	Workflow Characterization.....	298
F.2.7	Task Handling.....	300
F.2.8	Task Execution Management.....	300
F.2.9	Performance Modeling and Reliability Modeling.....	300
F.2.10	Requirements Framework.....	301
F.2.11	Security Framework.....	301
F.2.12	Reallocation management.....	302
F.2.13	Trigger and Threshold Management.....	302
F.3	Integration to Computing Frameworks	302
F.3.1	MPI	302
F.3.2	OpenMP	303
F.3.3	CUDA	304

LIST OF FIGURES

Figure 1 – Resource management functions as defined by (SINGH; CHANA, 2015b).....	29
Figure 2 – Cloud resource provisioning flowchart as defined by (SINGH; CHANA, 2015b)	29
Figure 3 – Resource provisioning and scheduling as defined by (SINGH; CHANA, 2016).....	30
Figure 4 – Conceptual resource management framework defined by (JENNINGS; STADLER, 2015).....	32
Figure 5 – HPC systems and their attributes according to (HUSSAIN et al., 2013)	55
Figure 6 – Resource provisioning evolution by (SINGH; CHANA, 2015b).....	56
Figure 7 – Resource provisioning mechanisms classified by (SINGH; CHANA, 2015b)	58
Figure 8 – QoS parameters used in provisioning mechanisms according to (SINGH; CHANA, 2015b).....	58
Figure 9 – Evolution of scheduling solutions	59
Figure 10 – Properties used by (BALA; CHANA, 2011) to classify the related work.....	61
Figure 11 – Properties used by (SOTIRIADIS et al., 2011) to classify references	62
Figure 12 – Categories defined by (WU; WU; TAN, 2015).....	62
Figure 13 – Taxonomy proposed by (SINGH; CHANA, 2015b) (figure extracted from reference) ...	63
Figure 14 – Taxonomy for Cost Based resource provisioning mechanisms (RPMs) (extracted from (SINGH; CHANA, 2015b)).....	64
Figure 15 – Taxonomy for time-based solutions (extracted from (SINGH; CHANA, 2015b))	64
Figure 16 – Taxonomy for compromised cost-time solutions (extracted from (SINGH; CHANA, 2015b)).....	65
Figure 17 – Taxonomy for bargaining-based solutions (extracted from (SINGH; CHANA, 2015b))	65
Figure 18 – Taxonomy for QoS-based solutions (extracted from (SINGH; CHANA, 2015b))	66
Figure 19 – Taxonomy for SLA-based solutions (extracted from (SINGH; CHANA, 2015b)).....	66
Figure 20 – Taxonomy for Energy-based solutions (extracted from (SINGH; CHANA, 2015b)).....	67
Figure 21 – Taxonomy for optimization-based solutions (extracted from (SINGH; CHANA, 2015b)).....	67

Figure 22 – Taxonomy for nature/bio-inspired solutions (extracted from (SINGH; CHANA, 2015b)).....	68
Figure 23 – Taxonomy for dynamic solutions (extracted from (SINGH; CHANA, 2015b))	68
Figure 24 – Taxonomy for rule-based solutions (extracted from (SINGH; CHANA, 2015b))	69
Figure 25 – Taxonomy for adaptive-based solutions (extracted from (SINGH; CHANA, 2015b))....	69
Figure 26 – Consolidated Taxonomy with categories addressing the main properties observed in the surveys and analyzed publications while addressing the interests of this work	70
Figure 27 – Overview of MPSF conceptual components	92
Figure 28 – Illustration of data distribution	99
Figure 29 – Experimental setup to validate model	106
Figure 30 – Expected time distribution for tasks	108
Figure 31 – Observed time distribution	109
Figure 32 – Data distribution in the enhanced model	111
Figure 33 – Expected time distribution for enhanced model	120
Figure 34 – Time distribution observed for experimental evaluation (<i>tw</i> is waiting time).....	121
Figure 35 – Visual example of matrix multiplication data structures.....	122
Figure 36 – Example of chunk and shared data.....	123
Figure 37 – Example of matrix multiplication implementation using only chunk data.....	123
Figure 38 – Distribution of shared data: node clustering.....	124
Figure 39 – Input sending to one node from each cluster.....	125
Figure 40 – Distribution of shared data to a cluster; before first spread.....	126
Figure 41 – Distribution of shared data to a cluster; after first spread.....	126
Figure 42 – Distribution of shared data to a cluster; after second spread	127
Figure 43 – Distribution of shared data to a cluster; after third spread	127
Figure 44 – Graph showing clustering based on dynamic threshold	130
Figure 45 – Graph showing clustering after applying static threshold	131
Figure 46 – Data distribution techniques. For each node, first bar is transfer and second bar is processing.	135
Figure 47 – Chunk distribution when $tp \geq tr + tw$	137

Figure 48 – Chunk distribution for $tp < tr + tw$	139
Figure 49 – Process of applying correction factors to data distribution	141
Figure 50 – Scenarios used in the experiments	153
Figure 51 – Visual representation of AXPY	156
Figure 52 – Visual representation of GEMM	157
Figure 53 – Visual representation of WAVG	158
Figure 54 - Summary of experimental results showing relative run time of MPSF compared to the Naive/Core approach. Figure shows scenarios (A, B, C), benchmark (AXPY, GEMM, WAVG), and problem size.	166
Figure 55 - Alternative workflow description for GEMM	167
Figure 56 - Alternative workflow description for WAVG	167
Figure 57 – Experimental results obtained by Rodriguez et al. (figure extracted from (RODRIGUEZ; BUYYA, 2014))......	208
Figure 58 – Primitives considered in the workflow characterization model	219
Figure 59 – Use cases for Input primitive	220
Figure 60 – Example of Input primitives with distinct points for preparation and iteration.....	221
Figure 61 – Example of Input primitive acting as restart point	222
Figure 62 – Simple use case of Split primitive	223
Figure 63 – Generic use case of Split primitive	224
Figure 64 – Example of Split after Split.....	224
Figure 65 – Split primitive after a Compute element	225
Figure 66 – Split primitive after a Merge	225
Figure 67 – Similar Merge-Split setup with an intermediary Input phase.....	226
Figure 68 – Output-Split setup	226
Figure 69 – Example of Compute primitive after an Input.....	227
Figure 70 – Example of Compute primitive enclosed by an Input and an Output.....	228
Figure 71 – Typical workflow construct composed by a sequence of Input, Split, Compute, and Merge.....	228
Figure 72 – Construct as internally interpreted by MPSF	229

Figure 73 – Workflow construct with explicit Input between Split and Compute	229
Figure 74 – Derived construct with explicit Input primitives.....	230
Figure 75 – Example of basic usage of Merge primitive to generate the final results of a workflow	230
Figure 76 – Intermediary Merge-Split to reorganize load distribution	231
Figure 77 – Nested Merge primitives	231
Figure 78 – Exemplar use cases for Output primitive	232
Figure 79 – Example of branching	233
Figure 80 – Supported branching and multi-path rules	234
Figure 81 – Example of Input with multiple outputs.....	235
Figure 82 – Example of branching using the Split primitive.....	236
Figure 83 – Enhancement of image processing workflow by adding another Split	237
Figure 84 – Example of construct implementing a loop.....	238
Figure 85 – Abstraction of a workflow (pipeline) into a single Compute element. The abstract Compute element was represented in a different color only for aesthetic differentiation.	239
Figure 86 – Example of reutilization of a workflow	240
Figure 87 – Another example of workflow transformed into a single Compute construct.....	240
Figure 88 – Example of workflow using the construct from Figure 87.....	241
Figure 89 – Example of general notation to indicate cost of transfers and computation	242
Figure 90 – Example of workflow cost characterization with fixed values.....	243
Figure 91 – Example of workflow with costs based on functions	243
Figure 92 – Example of workflow costs based on the data structures.....	246
Figure 93 – Exemplar matrix multiplication workflow	252
Figure 94 – Example of mapping of primitives into implementation.....	253
Figure 95 – Basic internal hierarchy of a cloud and its resources	261
Figure 96 – Complete architecture including the Execution Environment and its components	262
Figure 97 – Example of execution space.....	264
Figure 98 – Example of multiple sequential splits	266
Figure 99 – Classification of resources as defined by (SINGH; CHANA, 2016)	269

Figure 100 – Bathtub curve (KLUTKE; KIESSLER; WORTMAN, 2003)	273
Figure 101 – Probability of at least one failure for different failure rates	275
Figure 102 – Probabilities of zero to five failures and probability of more than 5 failures	276
Figure 103 – Typical event order for resource management and task deployment	280
Figure 104 – Load redistribution in case of failure	282
Figure 105 – Handling performance fluctuations	283
Figure 106 – Phases before and during execution	284
Figure 107 – Complete resource allocation life cycle	285
Figure 108 – Detail of reallocation showing inner steps	286
Figure 109 – Examples of time-based triggers	287
Figure 110 – MPSF Architecture.....	293

LIST OF TABLES

Table 1 – Summary of resource management definitions, actors considered in each case, and QoS/SLA aspects considered in the definitions	37
Table 2 – Explicit phases or steps proposed in each definition	38
Table 3 – How (JENNINGS; STADLER, 2015) and (MANVI; SHYAM, 2014) definitions fit the definition from (SINGH; CHANA, 2015b)	39
Table 4 – Distinction between static and dynamic scheduling as defined by (WU; WU; TAN, 2015). References for each example were not added for the sake of simplicity.	48
Table 5 – Summary of workflow scheduling strategies	49
Table 6 – Summary of identified related work classified using the consolidated taxonomy.....	72
Table 7 – Summary of Problems (P) addressed in this work.....	88
Table 8 – Summary of Solution Requirements (SR).....	90
Table 9 – Mapping between problems (P) and solution requirements (SR).....	91
Table 10 – Features (F) provided by MPSF	94
Table 11 – Parameters for theoretical problem.....	104
Table 12 – Machine parameters for experimental validation	107
Table 13 – Results obtained for experimental verification (one run)	109
Table 14 – Error margin for each predicted value	110
Table 15 – Parameters for theoretical problem.....	114
Table 16 – Machine parameters for experimental validation	118
Table 17 – Results obtained for experimental verification (one run)	121
Table 18 – Distribution after applying correction factors.....	141
Table 19 – Load level and respective hypothesis	142
Table 20 – Bayesian System of multiple hypotheses and evidences	142
Table 21 – Specification of tests and results	143
Table 22 – Probabilities of hypotheses based on the test and the result	144
Table 23 – Calculations for probability of hypotheses	144
Table 24 – Final values for hypotheses	145

Table 25 – Comparison of hypotheses values before and after test.....	146
Table 26 – Utilization of probabilities to calculate weighted average for correction factor.....	146
Table 27 – Architectural requirements and coverage	149
Table 28 – Specification of systems used in the experiments	153
Table 29 – Results (run time in seconds) for AXPY on Scenario A with vector size N of 500 million elements (8 GB of total footprint), 5 billion elements (80 GB), and 50 billion elements (800 GB).....	160
Table 30 - Results (run time in seconds) for GEMM on Scenario A with matrix side sizes N of 10k (total memory footprint of $3 \cdot (10k)^2 \cdot 8B = 2.4$ GB), 20k (9.6 GB), and 40k (38.4 GB).....	161
Table 31 - Results (run time in seconds) for WAVG on Scenario A with vector size N of 500 million elements (8 GB of total footprint), 5 billion elements (80 GB), and 50 billion elements (800 GB).....	162
Table 32 - Results (run time in seconds) for GEMM. Table compares results for 40k matrices running on Scenarios A and B.	163
Table 33 - Results (run time in seconds) for GEMM, 40k matrices running the workflow only on Mamona nodes vs. also using the Gauss nodes.....	164
Table 34 – Summary of Contributions	176
Table 35 – Summary of analyses of related work	217
Table 36 – Examples of resource properties.....	267

LIST OF ACRONYMS

CPOP	Critical-Path-On-a-Processor
DAG	Directed Acyclic Graph
DCP	Dynamic Critical Path
DLS	Dynamic Level Scheduling
ELISA	Estimated Load Information Scheduling Algorithm
FLOPS	Floating Point Operations per Second
GPU	Graphical Processing Unit
HEFT	Heterogeneous-Earliest-Finish-Time
IaaS	Infrastructure as a Service
MPI	Message Passing Interface
MPSF	Multiphase Proactive Scheduling Framework
MTBF	Mean Time Between Failures
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
PaaS	Platform as a Service
PEFT	Predict Earliest Finish Time
PSO	Particle Swarm Optimization
QoS	Quality of Service
RASA	Resource-Aware Scheduling Algorithm
RIC	Resource Information Centre
RPA	Resource Provisioning Agent
SaaS	Software as a Service
SHEFT	Scalable Heterogeneous-Earliest-Finish-Time
SLA	Service Level Agreement
SR	Solution Requirement
TPM	Trusted Platform Module
WaaS	Workflow-as-a-Service
WRM	Workload Resource Manager

1 INTRODUCTION

Cloud computing evolved from the concept of utility computing, which is defined as the provision of computational and storage resources as a metered service, similar to traditional public utility companies (RITTINGHOUSE; RANSOME, 2016). This concept reflects the fact that modern information technology environments require the means to dynamically increase capacity or add capabilities while minimizing the requirement of investing money and time in the purchase of new infrastructure.

Another key characteristic of cloud computing is multitenancy, which enables resource and cost sharing among a large pool of users (REN; WANG; WANG, 2012). This leads to the centralization of the infrastructure and consequent reduction of costs due to economies of scale (WANG et al., 2012). Moreover, the consolidation of resources leads to an increased peak-load capacity as each customer has access to a much larger pool of resources (although shared) compared to a local cluster of machines. Resources are more efficiently used, especially considering that in a local setup they often are underutilized (HOFMANN; WOODS, 2010). In addition, multitenancy enables dynamic allocation of these resources which are monitored by the service provider.

Characteristics such as multitenancy and elasticity perfectly fit the requirements of modern data intensive research and scientific endeavors (DEMCHENKO, 2016). These requirements are associated to the continuously increasing power of computing and storage resources that in many cases are required on-demand for specific phases of an experiment, therefore demanding elastic scaling. This motivates the utilization of clouds by scientific researchers as an alternative to using in-house resources (CHARD et al., 2015).

In parallel, as science becomes more complex and relies on the analysis of very large data sets, data management and processing must be performed in a scalable and automated way. Workflows have emerged as a way to formalize and structure data analysis, execute the required computations using distributed resources, collect information about the derived data products, and repeat the analysis if necessary (TAYLOR et al., 2014). Workflows enable the definition and sharing of analysis and results within scientific collaborations. In this sense, scientific workflows have become an increasingly popular paradigm for scientists to handle complex scientific processes (ZHAO et al., 2015), enabling and accelerating scientific progress and discoveries.

Scientific workflows, like other computer applications, can benefit from virtually unlimited resources with minimal investment. With such advantages, workflow scheduling research has thus shifted to workflow execution in the cloud (SMANCHAT; VIRIYAPANT, 2015), providing a paradigm-shifting utility-oriented computing environment with unprecedented size of data center resource pools and on-demand resource provisioning (ZHAO et al., 2015), enabling scientific workflow solutions to address petascale problems.

1.1 Problem and Motivation

One of the key enablers of this conjunction of cloud computing and scientific workflows is resource management (ARABNEJAD; BUBENDORFER, 2015), which includes resource provisioning, allocation, and scheduling (MANVI; SHYAM, 2014). Even small provisioning inefficiencies, such as failure to meet workflow dependencies on time or selecting the wrong resources for a task, can result in significant monetary costs (YI; ANDRZEJAK; KONDO, 2012; CHARD et al., 2015). Provisioning the right amount of storage and compute resources leads to decisive cost reduction (DEELMAN et al., 2008) with no substantial impact on application performance.

Consequently, cloud resource management for workflow execution is a topic of broad and current interest (WU; WU; TAN, 2015). Moreover, there are few researches on scheduling workflows on real cloud environments, and much fewer cloud workflow management systems, which require even further academic study and industrial practice (WU; WU; TAN, 2015). Workflow scheduling for commercial multicloud environments, for instance, still is an open issue to be addressed (FARD; PRODAN; FAHRINGER, 2013). In addition, data transfer between tasks is not directly considered in most existing studies, thus being assumed as part of task execution. However, this is not the case for data-intensive applications (WU; WU; TAN, 2015), especially ones from the big data era, wherein data movement can dominate both the execution time and cost.

1.2 Objectives and Contributions

This work presents the Multiphase Proactive Scheduling Framework (MPSF), a cloud scheduling framework based on multiple scheduling phases that continuously assess system resources in order to dynamically optimize resource distribution. MPSF is based on a rich high-level characterization of resources and workflows that describes the interactions between workflow phases and the computational cost for each case. Moreover, the framework considers both compute and transfer times to distribute the input to the nodes and also to retrieve the results. The workflow characterization is used to determine the time to process and transfer the input. The scheduling framework computes the optimal data distribution by determining how much data each node must receive in order to balance the execution and transfers across all nodes. In addition, MPSF is able to adequately distribute data in scenarios with multiple clouds, considering the cost of sending and receiving data over slower links. Finally, the framework provides rescheduling capabilities in order to redistribute the workload, addressing the performance fluctuations on the system. MPSF focuses on the Infrastructure-as-a-Service layer of cloud computing, although it may be applied to other layers as part of the infrastructure management for these services.

MPSF focuses on addressing the following challenges:

- **Data-intensive workflows:** Represented by workflows from the big data era (WU; WU; TAN, 2015) and modern workflows that are expected to drive research and development for future machines (APEX, 2016). In these workflows the data movement and communication activities can dominate both execution time and cost, while in most studies tasks such as data uploading and downloading are indirectly assumed as part of the execution (WU; WU; TAN, 2015). MPSF addresses these workflows by 1)

proposing a scheduling algorithm that considers the data transfers before, during, and after the compute phases of the workflow; and 2) proposing a resource and workflow characterization model that describes the compute and data movement patterns and interactions among workflow phases.

- **Hybrid and multcloud scenarios:** These scenarios enable the execution of large scientific workflows with several thousands of tasks, wherein the required computing and storage requirements exceed the limit of an individual cloud platform (JRAD; TAO; STREIT, 2013). Resource and task scheduling for these scenarios is even more complex than for single clouds, as it is fundamental to consider the impact of communication links among clouds (BITTENCOURT; MADEIRA; DA FONSECA, 2012). MPSF addresses these scenarios by leveraging the workflow and resource models and applying the scheduling algorithm considering the time to transfer data through the links connecting distinct clouds.
- **Performance fluctuation and multitenancy:** The most important problem when implementing algorithms for real environments is the uncertainties of task execution and data transmission time, both related to the performance fluctuations observed in the system (LEE; SUBRATA; ZOMAYA, 2009) and which are aggravated by the cloud multitenancy (WU; WU; TAN, 2015). MPSF addresses these fluctuations by implementing rescheduling and continuous assessment of resources and workloads among the systems.

These challenges represent the main aspects addressed by MPSF, providing mechanisms to support the execution of data-intensive workflows characterized by long and expensive data movement operations. MPSF defines algorithms for distributing tasks that

promote the optimization of resource utilization, addressing the requirements of hybrid and multicloud scenarios used for large-scale execution of problems. Moreover, MPSF provides mechanisms to mitigate the effects of performance fluctuation during the execution of workflows.

Two main contributions are identified in this work:

- The identification and definition of challenges for future solutions in cloud computing resource management and orchestration in general; and
- The definition and specification of a performance model that adequately distributes load in order to optimize task execution and minimize makespan, also addressing more complex cloud infrastructures such as hybrid and multicloud scenarios.

Other important aspects provided by this work are:

- A simplified set of primitives to characterize workflows that supports the definition of complex features, such as branching, nesting, and workflow composition;
- The definition of transparent mechanisms to integrate existing applications, including ones that use existing distributed computing frameworks such as MPI and OpenMP;
- Automated workflow analysis and definition of costs for workflow phases and transitions based on data to be transferred and processed;
- Definition of a resource hierarchy and abstractions to address hybrid and multicloud scenarios;

- Definition of a mechanism named segmentation to distribute tasks according to specific hierarchical levels;
- Gradual data distribution to minimize contention effects, leading to much higher task performance prediction accuracy than approaches solely based on weighted distributions;
- Definition of shared and chunk data to optimize data distribution, leading to performance improvement compared to simple data distribution based on multiple access to data sources;
- Pipelining mechanism to increase resource utilization combined to the gradual data distribution and processing mechanism;
- The utilization of a probabilistic model based on a Bayesian system of multiple hypotheses and multiple evidences to calculate correction factors applied to the resource distribution;
- Reliability model based on measurable system properties, as well as the definition of policy levels to handle different probabilities of failure; and
- Triggers and thresholds to control when the scheduling operation should be initiated and when it should be actually deployed onto the infrastructure.

A comprehensive analysis of these elements is presented in Section 5.2. Other contributions of this proposal include the definition of a taxonomy to classify resource management solutions and the identification of current challenges and future research directions regarding cloud computing resource management.

1.3 Method

The method adopted in this work is the applied research based on the hypothesis-deduction, including problem definition, hypotheses specification, and evaluation (WAZLAWICK, 2014). The work was organized in the following steps:

1. *Literature research*: Survey of existing resource management solutions for cloud computing and workflow management solutions. This included a critical evaluation and comparison of these solutions and the identification of open issues and challenges to be considered in the design of a novel solution.
2. *Solution design*: Proposal of the scheduling framework encompassing the necessary features to address the challenges described in the objectives.
3. *Implementation and validation*: Development of a reference implementation to validate the viability of the framework, comparing it to alternative solutions.

1.4 Document Organization

Chapter 2 presents the contextualization and analysis of relevant related work for this proposal. The chapter starts with the definition of resource management (Section 2.1). Then, the chapter presents the related work (Section 2.2), including the methodology used to identify the relevant publications and the definition of a taxonomy that was used to classify these publications. Finally, based on the results of this analysis, the chapter presents a summary of the gaps and challenges regarding resource management (Section 2.3), as well as the problem definition (Section 2.4) used to guide the design of MPSF.

Chapter 3 presents the details of MPSF. The chapter starts by providing a formal definition of the solution requirements (Section 3.1) based on the challenges and the problem definition. Then, the chapter presents an overview of the solution (Section 3.2). The performance model (Section 3.3), the main contribution of this work, is presented next. The other components that complement the framework are presented in appendices, such as workflow characterization primitives (Appendix B), resource characterization (Appendix C), reliability model (Appendix D), framework (Appendix E), and the architecture (0). Chapter 3 finishes by providing an analysis of requirements coverage (Section 3.4).

Chapter 4 presents the experiments and analysis. The chapter presents the experimental platform used (Section 4.1), details of the implementation of MPSF used in the experiments (Section 4.2), the benchmarks selected for the experiments (Section 4.3), the methods used to compare MPSF against (Section 4.4), the results obtained from the tests (Section 4.5), and an analysis of these results (Section 4.6).

Finally, Chapter 5 presents the final considerations of this work and future directions.

2 CLOUD RESOURCE MANAGEMENT

Cloud computing is a model for enabling on-demand self-service network access to a shared pool of elastic configurable computing resources (MELL; GRANCE, 2011). The model is driven by economies of scale to reduce costs for users (FOSTER et al., 2008) and to allow offering resources in a pay-as-you-go manner, thus embodying the concept of utility computing (ARMBRUST et al., 2009, 2010).

In its inception, cloud computing revolved around virtualization as main resource compartmentalization or consolidation strategy (LI et al., 2010; PHAPHOOM; WANG; ABRAHAMSSON, 2013) to support application isolation and platform customization to suit user needs (BUYAYA et al., 2008, 2009), as well as to enable pooling and dynamically assigning computing resources from clusters of servers (ZHANG; CHENG; BOUTABA, 2010). The significant performance improvement and overhead reduction of virtualization technology (NURMI et al., 2009) propelled its adoption as key delivery technology in cloud computing (CHIEU et al., 2009). Nevertheless, developments on Linux Containers and associated technologies (FELTER et al., 2015; MERKEL, 2014) led to the implementation of cloud platforms using lightweight containers (HE et al., 2012) such as Docker (LIU; ZHAO, 2014; SLOMINSKI; MUTHUSAMY; KHALAF, 2015) with smaller overhead compared to virtual machines as containers only replicate the libraries and binaries of the virtualized application (KACAMARGA; PARDAMEAN; WIJAYA, 2015).

Resource management in a cloud environment is a challenging problem due to the scale of modern data centers, the heterogeneity of resource types, the interdependency between these

resources, the variability and unpredictability of the load, and the range of objectives of different actors in a cloud ecosystem (JENNINGS; STADLER, 2015). The objective of this chapter is to provide a thorough investigation of resource management. Section 2.1 presents the definitions and concepts of cloud resource management. Section 2.2 presents an analysis of existing surveys and taxonomies, as well as a comprehensive summary of over 100 related publications. Section 2.3 presents the gaps and challenges identified during this investigation. Section 2.4 provides the problem definition used throughout the rest of this work. Section 2.5 concludes this chapter.

2.1 Resource Management: Definition and Concepts

Resource management comprises different stages or resources and workloads. Due to its importance as fundamental building block for cloud computing, several definitions and concepts are found in the literature. This section explores these definitions and then provides a consolidate view of cloud resource management.

2.1.1 Definition by (SINGH; CHANA, 2015b)

According to (SINGH; CHANA, 2015b) resource management in cloud comprises three functions: resource provisioning, resource scheduling, and resource monitoring. These functions are observed in Figure 1.

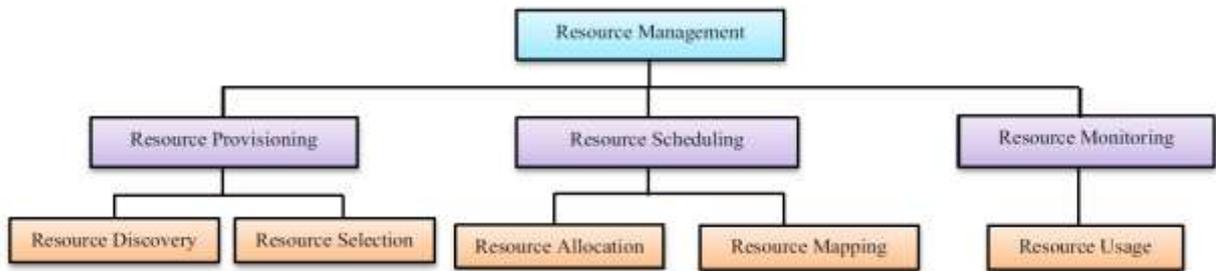


Figure 1 – Resource management functions as defined by (SINGH; CHANA, 2015b)

Resource provisioning is defined by the authors as the stage to identify the adequate resources for a particular workload based on quality of service (QoS) requirements defined by cloud consumers. This stage includes the discovery of resources and also their selection for executing a workload. The provisioning of appropriate resources to cloud workloads depends on the QoS requirements of cloud applications (CHANA; SINGH, 2014). Figure 2 depicts a flowchart of the resource provisioning process as defined by the authors.

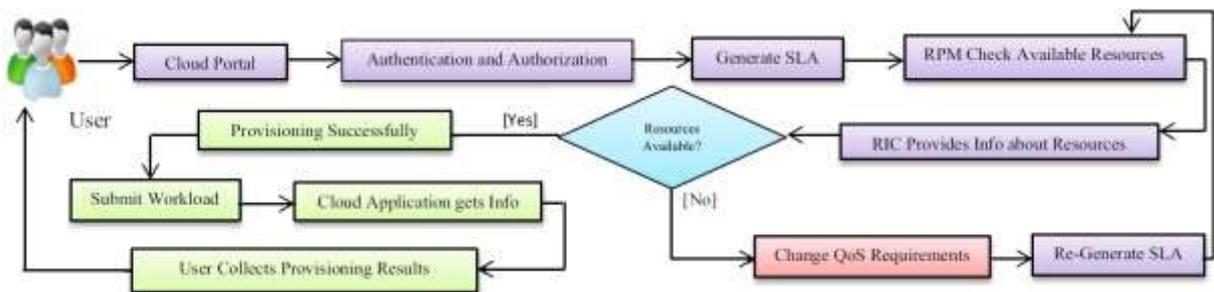


Figure 2 – Cloud resource provisioning flowchart as defined by (SINGH; CHANA, 2015b)

Analyzing Figure 2, the cloud consumer interacts with the cloud via a cloud portal and submits the QoS requirements of the workload after authentication. The Resource Information Centre (RIC) contains the information about all the resources in the resource pool and obtains the result based on requirement of workload as specified by user. The user requirements and the information provided by the RIC are used by the Resource Provisioning Agent (RPA) to check the available resources. After provisioning of resources the workloads are submitted to

the resource scheduler. Finally, the Workload Resource Manager (WRM) sends the provisioning results (resource information) to the RPA, which forwards these results to the cloud user.

Resource scheduling is defined as the mapping, allocation, and execution of workloads based on the resources selected in the resource provisioning phase (SINGH; CHANA, 2016). The scheduling process is illustrated in Figure 3, which also includes the initial provisioning stages and parts of the monitoring phase.

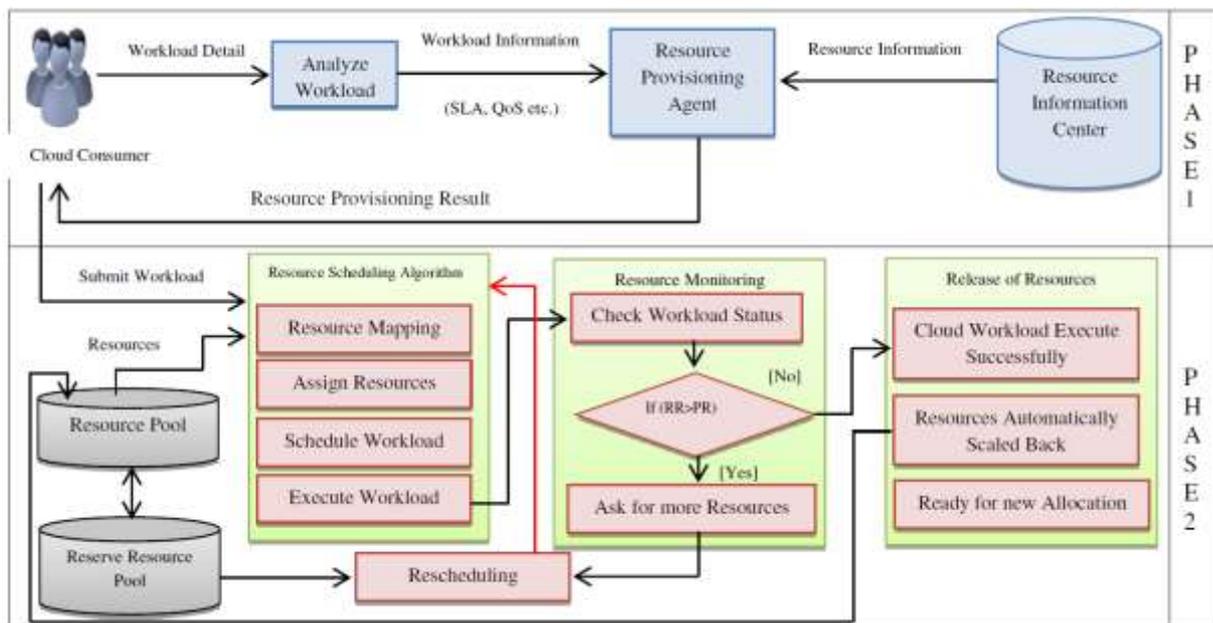


Figure 3 – Resource provisioning and scheduling as defined by (SINGH; CHANA, 2016)

In Figure 3 the Phase I corresponds to the provisioning steps. This comprises obtaining information about the workload, such as QoS attributes and SLAs. The workload is then submitted to the resource scheduler. Phase II comprises the scheduling and monitoring steps. Mapping workloads refers to selecting the appropriate resources based on the QoS requirements as specified by user in terms of SLA to minimize cost and execution time, for instance. The process of finding the list of available resources is referred to as resource detection, while the

resource selection is the process of choosing the best resource from list generated by resource detection based on SLA.

Resource monitoring is a complementary phase to achieve better performance optimization. In terms of service level agreements (SLA) both parties (provider and consumer) must specify the possible deviations to achieve appropriate quality attributes. For successful execution of a workload the observed deviation must be less than the defined thresholds. In this sense, resource monitoring is used to take care of important QoS requirements like security, availability, and performance. Figure 3 shows the monitoring steps related to checking the workload status and verifying if the amount of required resources (RR) is larger than the amount of provided resources (PR). Depending on the result more resources are demanded by the scheduler. On the other hand, based on this result the resources can also be released, freeing them for other allocations. In this sense the monitoring phase also controls the rescheduling activities.

2.1.2 Definition by (JENNINGS; STADLER, 2015)

For (JENNINGS; STADLER, 2015), resource management is the process of allocating computing, storage, networking and energy resources to a set of applications in order to meet performance objectives and requirements of the infrastructure providers and the cloud users. On one hand, the objectives of the providers are related to efficient and effective resource utilization within the constraints of SLAs. The authors claim that efficient resource use is typically achieved through virtualization technologies, facilitating the multiplexing of resources across customers. On the other hand, the objectives of cloud users tend to focus on application

performance, their availability, as well as the cost-effective scaling of available resources based on application demands.

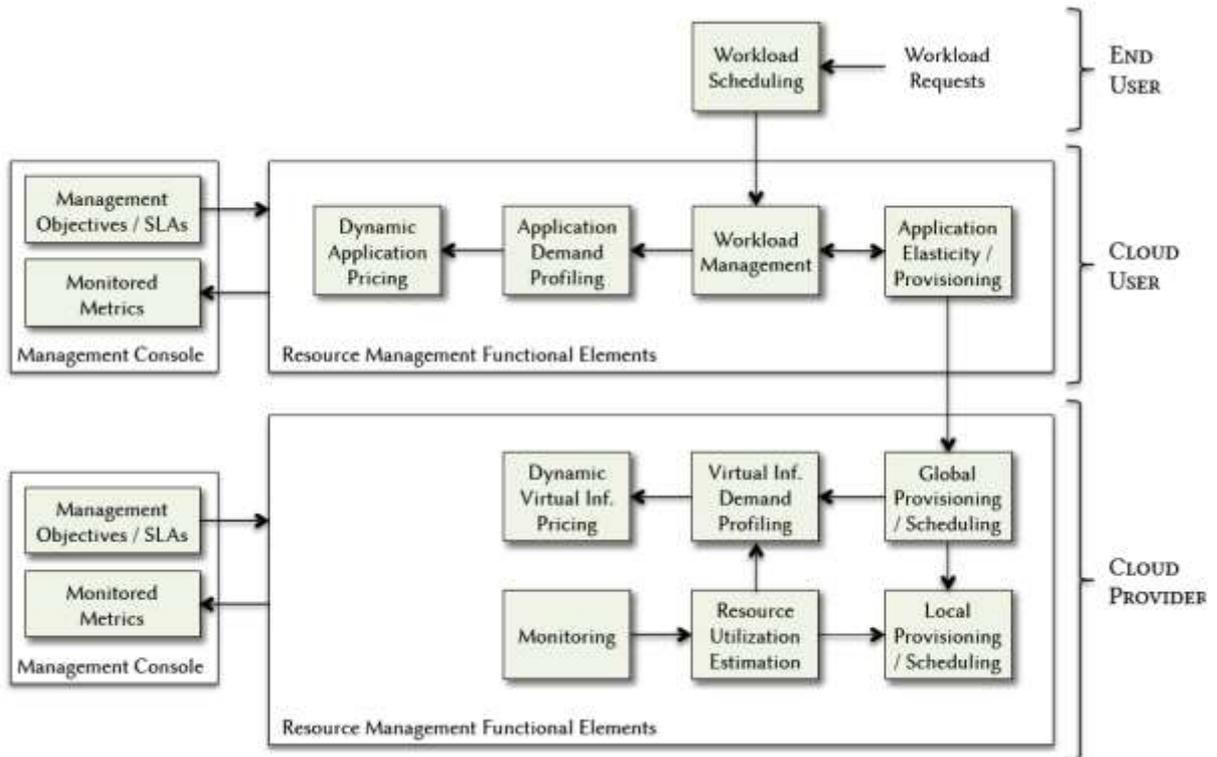


Figure 4 – Conceptual resource management framework defined by (JENNINGS; STADLER, 2015)

Figure 4 presents the functional elements of resource management according to the authors. The cloud provider is responsible for monitoring the utilization of compute, networking, storage, and power resources, as well as for controlling this utilization via global and local scheduling processes. In parallel, the cloud user monitors and controls the deployment of its applications on the virtual infrastructure. Cloud providers can dynamically alter the prices charged for leasing the infrastructure while cloud users can alter the costs by changing application parameters and usage levels. However, the cloud user has limited responsibility for resource management, being constrained to generating workload requests and controlling where and when workloads are placed.

The authors distinguish the roles of cloud user from end user. The end user generates the workloads that are processed using cloud resources. The cloud user actively interacts with the cloud infrastructure to host applications for end users. In this sense, the cloud user acts a broker, thus being responsible for meeting the SLAs specified by the end user. Moreover, the cloud user is mostly interested in meeting these requirements in a manner to minimize its own costs of leasing the cloud infrastructure (from the cloud provider) while maximizing its profits.

From a functional perspective, the end user initiates the process by providing one or more workload requests to the workload scheduling component. The requests are relayed to the workload management component provided by the cloud user (broker). The application is submitted to a profiling process that dynamically defines the pricing characteristics, also defining the metrics to be monitored during execution and the objectives (SLAs) to be observed. The cloud user defines the provisioning to be obtained from the cloud provider. The provider receives the requests via a global provisioning and scheduling component that also profiles the requests in order to determine the pricing attributes (this time from cloud provider to cloud user). Moreover, the application is characterized in order to obtain monitoring metrics and objectives from the cloud provider point of view. Finally, the global provisioning and scheduling element submits requests for the local handler, estimating the resource utilization and executing the workloads.

2.1.3 Definition by (MANVI; SHYAM, 2014)

According to (MANVI; SHYAM, 2014), cloud computing resource management comprises nine components:

- **Provisioning:** Assignment of resources to a workload.
- **Allocation:** Distribution of resources among competing workloads.
- **Adaptation:** Ability to dynamically adjust resources to fulfill workload requirements.
- **Mapping:** Correspondence between resources required by the workload and resources provided by the cloud infrastructure.
- **Modeling:** Framework that helps to predict the resource requirements of a workload by representing the most important attributes of resource management, such as states, transitions, inputs, and outputs within a given environment.
- **Estimation:** Guess of the actual resources required for executing a workload.
- **Discovery:** Identification of a list of resources that are available for workload execution.
- **Brokering:** Negotiation of resources through an agent to ensure their availability at the right time to execute the workload.
- **Scheduling:** A timetable of events and resources, determining when a workload should start or end depending on its duration, predecessor activities, predecessor relationships, and resources allocated.

The authors did not explicitly defined the roles or actors related to cloud management activities. The implicit roles in this sense are the cloud provider (responsible for managing the cloud infrastructure) and the cloud user (interested in executing one or more workloads on the cloud infrastructure). QoS is regarded as fundamental part of the resource management

premises. In contrast, the SLAs are not explicitly defined as building block for resource management tasks.

2.1.4 Other Definitions

According to (MUSTAFA et al., 2015), resource management is a process that deals with the procurement and release of resources. Moreover, resource management provides performance isolation and efficient use of underlying hardware. The authors state that the main research challenges and metrics of resource management are energy efficiency, SLA violations, load balancing, network load, profit maximization, hybrid clouds, and mobile cloud computing. No specific remark to cloud roles or to quality of service are made, although the solutions covered by the survey might present QoS related aspects.

For (MARINESCU, 2013), resource management is a core function of cloud computing that affects three aspects: performance, functionality, and cost. In this sense, cloud resource management requires complex policies and decisions for multi-objective optimization. These policies are organized in five classes: admission control, capacity allocation, load balancing, energy optimization, and quality of service guarantees. The admission policies prevent the system from accepting workloads in violation of high-level system policies (e.g., a workload that might prevent others from completing). Capacity allocation comprises the allocation of resources for individual instances. Load balancing and energy optimization can be done either locally or globally, and both are correlated to cost. Finally, quality of service is related to addressing requirements and objectives concerning users and providers. SLA aspects are not explicitly considered in this set of policies.

For (WEINGARTNER; BRASCHER; WESTPHALL, 2015), resource management is related to predicting the amount of resources that best suits each workload, enabling cloud providers to consolidate workloads while maintaining SLAs.

For (MAJUMDAR, 2011), resource management comprises two main activities: matching, which is the process of assigning a job to a particular resource; and scheduling, which is the process of determining the order in which jobs assigned to a particular resource should be executed.

2.1.5 Intercorrelation and Consolidation

Table 1 presents a summary of the resource management definitions. The table presents the works analyzed for the study of definitions of resource management, a summary of the viewpoints from each work, which are the actors identified in each work, and whether aspects related to Quality of Service and Service Level Agreements are mentioned and considered in the works or not. The importance of identifying these aspects is to analyze the similarities and disparities among the works to allow a better understanding of the definitions and the identification of relevant points related to this proposal.

Table 1 – Summary of resource management definitions, actors considered in each case, and QoS/SLA aspects considered in the definitions

<i>Authors</i>	<i>Summary</i>	<i>Actors</i>	<i>QoS</i>	<i>SLA</i>
(SINGH; CHANA, 2015b)	Three phases: provisioning, scheduling, monitoring	Cloud provider (infra. and workload mgmt.) and cloud consumer (end user)	Yes	Yes
(JENNINGS; STADLER, 2015)	Organized in three tiers (one per role) and a total of 15 different stages, including scheduling, provisioning, pricing, profiling, and monitoring.	Cloud provider (infra. mgmt.), cloud user (broker), end user (execute workload)	Yes	Yes
(MANVI; SHYAM, 2014)	Nine components: provisioning, allocation, adaptation, mapping, modeling, estimation, discovery, brokering, and scheduling	No specific actors are identified; implicit assumption of at least two roles: cloud provider and cloud user	Yes	No
(MUSTAFA et al., 2015)	Two tasks: procurement and release of resources. Two objectives: performance isolation and efficient use of hardware. Seven metrics: energy, SLA, load, network load, profit, hybrid clouds, and mobile clouds.	No specific actors are identified; implicit assumption of at least two roles: cloud provider and cloud user	No	Yes
(MARINESCU, 2013)	Three aspects affected: performance, functionality, and cost. Five policy classes: admission control, capacity allocation, load balancing, energy optimization, and QoS.	Cloud provider and cloud user.	Yes	No
(WEINGARTNER; BRASCHER; WESTPHALL, 2015)	Predicting workload to enable workload consolidation while meeting SLAs.	No specific actors	Yes	Yes
(MAJUMDAR, 2011)	Two main activities: matching and scheduling.	No specific actors	Yes	Yes

Among these definitions, (MANVI; SHYAM, 2014) defines nine components for resource management, including provisioning, allocation, adaptation, mapping, modeling, estimation, discovery/selection, brokering, and scheduling. Some works treat resource management and resource scheduling as the same concept. For instance, (WU; WU; TAN,

2015) present a survey focusing on resource scheduling that also comprises several of the components proposed by (MANVI; SHYAM, 2014), such as provisioning, allocation, and modeling.

Three definitions were selected as main references for resource management:

- (SINGH; CHANA, 2015b);
- (JENNINGS; STADLER, 2015); and
- (MANVI; SHYAM, 2014).

These were selected due to their clear definition of steps and components of resource management. Table 2 provides a summary of the phases or steps proposed by each definition.

Table 2 – Explicit phases or steps proposed in each definition

<i>Authors</i>	<i>Explicit Phases or Steps</i>
(SINGH; CHANA, 2015b)	Provisioning, Scheduling, Monitoring
(JENNINGS; STADLER, 2015)	Profiling, Pricing, Provisioning, Estimation, Scheduling, Monitoring
(MANVI; SHYAM, 2014)	Provisioning, Allocation, Adaptation, Mapping, Modeling, Estimation, Discovery, Brokering, Scheduling

While the definition from (MANVI; SHYAM, 2014) proposes more steps than the others, there is a natural correlation between the phases proposed by each definition. Table 3 presents the correlation between the phases from (SINGH; CHANA, 2015b) and the other two. The objective of this table is to fit the steps proposed by (JENNINGS; STADLER, 2015) and by (MANVI; SHYAM, 2014) into the steps from (SINGH; CHANA, 2015b), which represents a simpler classification of resource management tasks.

Table 3 – How (JENNINGS; STADLER, 2015) and (MANVI; SHYAM, 2014) definitions fit the definition from (SINGH; CHANA, 2015b)

<i>(SINGH; CHANA, 2015b)</i>	<i>(JENNINGS; STADLER, 2015)</i>	<i>(MANVI; SHYAM, 2014)</i>
Provisioning	Profiling, Pricing, Provisioning	Discovery, Modeling, Brokering, Provisioning
Scheduling	Estimation, Scheduling	Estimation, Mapping, Allocation, Scheduling
Monitoring	Monitoring	Adaptation

Comparing (JENNINGS; STADLER, 2015) to (SINGH; CHANA, 2015b), the workload profiling (to assess the resource demands), pricing, and provisioning steps defined by Jennings and Stadler fit the provisioning step from (SINGH; CHANA, 2015b), which is essentially the phase to identify the resources for a particular workload based on its characteristics and on the QoS. This includes the selection of resources to execute the workload. These aspects fit the steps of discovery, modeling, brokering, and provisioning from (MANVI; SHYAM, 2014). Note that the brokering aspect is also implicitly included in the definition from Jennings and Stadler, as they define a specific role for the brokering activity (the cloud user; the end user is the actor that has a workload to be executed in the cloud).

The scheduling phase from (SINGH; CHANA, 2015b) are organized in estimation and scheduling by (JENNINGS; STADLER, 2015). (MANVI; SHYAM, 2014) include an

allocation step to these two. In summary, these steps represent the mapping, allocation, and execution of the workload based on the resources selected in the provisioning phase.

Finally, the monitoring phase is present in (SINGH; CHANA, 2015b) and in (JENNINGS; STADLER, 2015). For (MANVI; SHYAM, 2014) the monitoring tasks are implicitly included by the adaptation step, which is related to dynamically adjusting resources to fulfill workload requirements. Because it is necessary to monitor both resource availability and workload conditions in order to provide this feature, this means that this step directly relies on some form of monitoring.

In terms of consolidation, the common point of all definitions is the aspect of managing the life cycle of resources and their association to the execution of tasks. This is the central governing point of cloud resource management which is independent of a specific phase of this life cycle. While it is fundamental to distinguish each phase, they all contribute to two ultimate purposes: enable task execution and optimize infrastructural efficiency based on a set of specified objectives. These are the key points of interest of this work, therefore comprising not only the specific task of scheduling resources (i.e., associating them to a task), but also managing the resource from its initial preparation (e.g., discovery) to its utilization and distribution. As basis to this research the definition of (SINGH; CHANA, 2015b) is adopted as it encompasses all aspects of resource management while keeping the number of categories to a minimum.

2.2 Related Work

Because of its relevance, cloud computing resource management is a topic that not only has a lot of work and research, but also existing surveys and taxonomies. This section presents a broad analysis of the related work and surveys.

2.2.1 Methodology

The method used to identify the surveys and other related work is based on searches performed in three different sets of tools: Google Scholar, Google, and complementary engines. Moreover, two main search queries were used for survey identification: “cloud scheduling survey” (without quotes) and “cloud resource management survey” (also without quotes). Regarding the search engines, the following strategy was adopted:

- For Google Scholar¹ two approaches were adopted: a) results from any year sorted by relevance; and b) results sorted by date.
- For Google both queries were used and up to ten pages were analyzed as well. Different from the Google Scholar results, the hits obtained via Google include white papers and blog posts, representing the view of the industry.

¹ <http://scholar.google.com>

- Complementary engines include IEEE Xplore², ACM Digital Library³, ScienceDirect⁴, Scopus⁵, and CAPES⁶.

Some results were discarded, such as ones addressing mobile cloud computing or other specific scenarios, such as Internet of Things and sensor networks. The focus of this analysis is to identify the surveys and taxonomies for general cloud computing resource management. Surveys covering hybrid clouds and intercloud were included. Surveys related to grid computing and other distributed computing technologies were also considered.

2.2.2 Identified Related Work and Surveys

2.2.2.1 *A Survey of Various Workflow Scheduling Algorithms in Cloud Environment*

(BALA; CHANA, 2011) identifies ten scheduling algorithms:

- (ZHONG; TAO; ZHANG, 2010): Optimized resource scheduling, using an improved genetic algorithm to implemented an automated scheduling policy in order to increase utilization rate of resources.
- (SELVARANI; SADHASIVAM, 2010): Improved cost-based algorithm for task scheduling for efficient mapping of tasks to cloud resources. The algorithm measures

² <http://ieeexplore.ieee.org>

³ <http://dl.acm.org>

⁴ <http://www.sciencedirect.com>

⁵ <https://www.scopus.com>

⁶ <http://www.periodicos.capes.gov.br>

resource cost and computation performance in order to improve computation versus communication ratio.

- (LIU et al., 2010): Time-cost scheduling algorithm for instance-intensive cost-constrained workflows. Instance-intensive workflows are characterized by a large number of relatively simple instances. In this sense, mean execution time becomes an important criterion for scheduling. Typical scenarios include bank check processing, insurance claim processing and many other e-business and e-government scenarios.
- (PANDEY et al., 2010): Particle swarm optimization (PSO) heuristic to schedule applications to cloud resources considering computation cost and data transmission cost. Solution applied to workflow applications.
- (LIN; LU, 2011): Scalable heterogeneous earliest finish time algorithm (SHEFT), a substantial improvement over HEFT (which focuses on optimizing workflow execution time). Moreover, SHEFT enables resources to scale at run time.
- (WU et al., 2013): Market oriented hierarchical scheduling strategy, consisting of service-level scheduling and task-level scheduling.
- (XU et al., 2009): Multiple QoS constrained scheduling for multi-workflows. Objectives include makespan minimization and costs.
- (VARALAKSHMI et al., 2011): Optimal Workflow Scheduling (OWS) algorithm based on QoS constraints to improve execution time.
- (PARSA et al., 2009): Resource-aware scheduling algorithm (RASA) composed of two algorithms: max-min and min-min. Max-min gives priority to larger tasks while min-

min gives priority to smaller tasks. Both are combined to deliver a better final result in terms of scheduling.

- (TOPCUOGLU et al., 2002): Heterogeneous-Earliest-Finish-Time (HEFT), algorithm that focuses on minimizing makespan by calculating the average run time (execution and communication) and defining a ranking system for the tasks, so that tasks with higher priority can be completed at the earliest time.

2.2.2.2 *Towards inter-cloud schedulers: A survey of meta-scheduling approaches*

(SOTIRIADIS et al., 2011) focuses on intercloud schedulers. However, most of the works analyzed are not specific to cloud computing – some address grid computing, and several precede the consolidation of grid computing. The following solutions are analyzed:

- (WEISSMAN; GRIMSHAW, 1996): Wide-area scheduling based on local resource management systems. Each member of the distributed area must instantiate the local manager while the wide-area scheduler provides global scheduling functionality.
- (SCHWIEGELSHOHN; YAHYAPOUR, 1999): Meta-scheduling mechanism that controls several meta-domains. The solution integrates to existing (conventional) schedulers in place.
- (ANAND; GHOSE; MANI, 1999): ELISA (Estimated Load Information Scheduling Algorithm), decentralized dynamic algorithm that aims to increase possibilities of gain load-balancing by estimating usage and job arrival rate.

- (DAVAL-FREROT; LACROIX; GUYENNET, 2000): Federation of distributed resource traders and parallelized job submissions. Traders cooperate to maximize a trading function and negotiate parameters such as response time.
- (SUBRAMANI et al., 2002): Distributed computing scheduling based on adapting to changes in global resource usage. The key idea is to redundantly distributed each job to multiple sites. In this way the possibility of backfilling (jobs moving to the front of a queue in order to fill any space created by a different algorithm) is higher. This solution, however, slightly decreases overall performance.
- (BUTT; ZHANG; HU, 2003): Model for connecting several Condor pools (LITZKOW; LIVNY; MUTKA, 1988). The work focuses on resource discovery using a peer-to-peer routing model. The “flock of Condors” are able to reduce maximum job waiting time in the queue.
- (ANDRADE et al., 2003): Scheduling infrastructure based on bag-of-tasks applications. The infrastructure is a composition of peers that constitute a community.
- (LAI et al., 2005): Market-based resource allocation system based on auctions. Each resource provider runs an auction for its resources. Bid for resources are conducted by meta-schedulers.
- (SHAH et al., 2007): Modified ELISA (MELISA) plus load balancing on arrival. MELISA calculates the load of neighbor nodes considering their job arrival rates, service rate, and node load.

- (IOSUP et al., 2008): Delegated matchmaking to interconnect several grids without requiring a central point of control. If user requirements cannot be satisfied in local level, then the remote resources are transparently included.
- (DIAS; BUYYA; VENUGOPAL, 2008): Interconnects grids via peering arrangements via InterGrid Gateways. Solution improves average response time.
- (LEAL; HUEDO; LLORENTE, 2009): Decentralized model composed of three strategies: static objective, dynamic objective, and advance scheduling. Implementation shows improvement to makespan.
- (FÖLLING et al., 2009): Evolutionary Fuzzy system approach for decentralized grids. Performance enhancement (response time) observed from a stable basis of workload distribution.
- (RODERO et al., 2010): Brokering for multiple grid scenarios. Authors propose a meta-brokers system that behave as gateways. Performance is not penalized significantly.
- (WANG et al., 2010): Bidding system to solve the overloading issue of resource selection. Resource selection heuristic is proposed to minimize turnaround time in a non-reserved bidding-based grid environment. Method does not consider important performance aspects such as job workload, CPU capabilities, job execution deadlines, and dynamic availability of resources.
- (HUANG et al., 2013): Community-aware decentralized dynamic scheduling. Solution aims at overall performance improvement, rather than individual hosts performance.

- (RODERO et al., 2009): Backfilling-based strategy that uses dynamic performance information instead of job requirements and resource characteristics. Overall objective is to minimize execution time (makespan), effectively outperforming first come first served strategies.
- (XU et al., 2011): Considers the commercialization and virtualization characteristics of cloud computing and implements a “justice-based” distribution. The idea is that each individual in the “social system” is able to judge its own gained resources to be fair or not compared to the other elements in the “society”. Execution time is the main metric controlled and improved.

Because these solutions were not designed specifically for cloud computing, (SOTIRIADIS et al., 2011) provide an analysis of which of those solutions are suitable for interclouds. They conclude that only (RODERO et al., 2009) and (XU et al., 2011) are potential candidates in that sense, based on characteristics such as management of unpredictability, heterogeneity of resources, geographical distribution of these resources, variation of job requirements, compatibility of different SLAs, and rescheduling support. According to the authors these are the essential characteristics for a multicloud scheduling solution.

2.2.2.3 *Workflow scheduling in cloud: a survey*

Related work and Taxonomy – The authors provide an extensive study of existing work on cloud scheduling, also include scheduling solutions not initially designed for cloud computing.

The first classification proposed refers to the distinction between static and dynamic scheduling. Static scheduling is based on the characteristics of the workflow (represented as a

directed acyclic graph, DAG). This information is known a priori and all resources have instant availability and stable performance. Scheduling is performed before executing the workflow. In contrast, dynamic scheduling considers the variability of resource availability and execution time. Table 5 provides an overview of static and dynamic scheduling mechanisms.

Table 4 – Distinction between static and dynamic scheduling as defined by (WU; WU; TAN, 2015). References for each example were not added for the sake of simplicity.

<i>Category</i>	<i>Subcategory</i>	<i>Description</i>	<i>Examples of mechanisms</i>
Static workflow scheduling	List heuristic	Scheduling is based on the creation of a list with a sequence of tasks. Tasks are given priorities and then allocated to selected resources.	Modified critical path, mapping heuristic, insertion scheduling heuristic, earliest time finish, heterogeneous earliest-finish-time (HEFT), critical-path-on-a-processor (CPOP), dynamic level scheduling (DLS), dynamic critical path (DCP), predict earliest finish time (PEFT)
	Clustering heuristic	Designed to optimize transmission time between data dependent tasks by aggregating these tasks in a single cluster.	DSC, KB/L, Sarkar, CASS-II (WU; WU; TAN, 2015)
	Duplication heuristic	Duplicate task on the same resource with the target task such that the transmission time between these two tasks is avoided.	TDS (critical parent is duplicated), LWB (lower bound of start-time is approximated), PLW (similar to LWB but with lower complexity), DFRN (duplication first and reduction next), DSH (utilization of idle slot is maximized), BTDH (extension of DSH), LCTD (optimization of linear clustering), CPF (task on critical path is considered at first), PY and TCSD (both similar to LWB)
	Meta-heuristics	Adopted to circumvent the NP-complete characteristic of DAG scheduling.	Genetic algorithms, GRASP (greedy randomized adaptive search procedure), SA (simulated annealing), PSO (particle swarm optimization), ACO (ant colony optimization)
Dynamic workflow scheduling		Developed to handle the unavailability of scheduling information and resource contention.	BMBP (binomial method batch predictor), admission control mechanisms, task throttling

For cloud workflow scheduling the authors define several aspects which are covered by existing scheduling solutions. Table 5 summarizes these categories and their examples.

Table 5 – Summary of workflow scheduling strategies

<i>Category</i>	<i>Subcategory</i>	<i>Examples</i>	<i>Description</i>
<i>Best-effort</i> Optimize one objective while ignoring other factors such as QoS requirements	-	PANDEY et al., 2010	Particle swarm optimization to minimize cost
		BILGAIYAN; SAGNIKA; DAS, 2014	Cat swarm optimization to minimize cost
		CHEN; ZHANG, 2009	Ant colony optimization for QoS constrained scheduling
<i>Deadline-constrained</i> Scheduling based on the trade-off between execution time and monetary cost under a deadline constraint.	<i>Deadline distribution heuristic</i>	YU; BUYYA; THAM, 2005	Partitions tasks in top-bottom direction to implement deadline distribution
		YUAN et al., 2008	Divides deadline by amortizing float time
		YUAN et al., 2009	Partitions all tasks into different paths
		ABRISHAMI; NAGHIBZADEH; EPEMA, 2013	Partial Critical Path (PCP) evolved from a model applied to grids
	<i>Meta-heuristic</i>	YU; BUYYA, 2006	Genetic algorithm to optimize cost with deadline constraint. Solution includes a fitness function and a crossover, mutation, swapping, and replacing operators.
		WU et al., 2010	Revised discrete particle swarm optimization (RDPSO) to minimize cost with deadline constraint
		RODRIGUEZ; BUYYA, 2014	Meta-heuristic optimization algorithm of PSO, considering factors such as performance variation and VM boot time

Table 5 – Summary of workflow scheduling strategies (cont. I)

<i>Category</i>	<i>Subcategory</i>	<i>Examples</i>	<i>Description</i>
<i>Budget-constrained</i> The objective is to finish a workflow as fast as possible at given budget.	<i>One-time</i>	YU et al., 2009	Greedy time-cost distribution, statistically distributes budget to tasks proportionally to their average execution times
		ZHENG; SAKELLARIOU, 2011	Budget-constrained HEFT dynamically distributes budget
		ARABNEJAD; BARBOSA, 2014a	HBCS algorithm, resource with highest aggregation weight is selected
	Backtracking	SAKELLARIOU et al., 2007	LOSS and GAIN approaches. LOSS gets better makespan due to efficient DAG heuristic. GAIN performs better if available budget is close to the cheapest.
		LIN; WU, 2013	Minimize end-to-end delay under budget constraint (MED-CC).
		ZENG; VEERAVALLI; LI, 2012	Backtracking algorithm based on comparative advantage and adjustment steps. Makespan and cost are recalculated every step, increasing cost complexity.
	Meta-heuristic	YU; BUYYA, 2006	Search for time optimal solutions with budget constraint

Table 5 – Summary of workflow scheduling strategies (cont. II)

<i>Category</i>	<i>Subcategory</i>	<i>Examples</i>	<i>Description</i>
Multi-criteria Several objectives are taken into account	<i>Aggregation approach</i>	LI et al., 2011	CCSH (Cost-Conscious Scheduling Heuristic), based on HEFT, to minimize makespan and cost
		GARG; BUYYA; SIEGEL, 2010	Three heuristics aggregating conflicting objectives of makespan and cost
		FARD et al., 2012	List scheduling algorithm for multi-objective DAG scheduling; case study includes makespan, cost, energy, and reliability
		DOGAN; OZGUNER, 2002	Bi-criteria (performance and reliability) optimization using aggression approach
		DOGAN; OZGUNER, 2005	Genetic algorithm for bi-objective dynamic level scheduling
		HAKEM; BUTELLE, 2007	Performance and reliability optimization via list scheduling heuristic
		DONGARRA et al., 2007	Reliable HEFT based on a product of failure rate and performance of a resource
	LEE; SUBRATA; ZUMAYA, 2009	Reduce makespan and increase resource utilization	
	<i>ϵ-approach</i>	PRODAN; WIECZOREK, 2010	Multiple-choice knapsack problem; solution based on dynamic programming; use cases based on cost, reliability, makespan, and data quality
	<i>Pareto approach</i>	BESSAI et al., 2012	Execution time and cost
YU et al., 2007		Multi-objective Evolutionary Algorithms (MOEA) to minimize execution time and cost	
TALUKDER et al., 2009		Multi-objective Differential Evolution (MODE) approach	

Table 5 – Summary of workflow scheduling strategies (cont. III)

<i>Category</i>	<i>Subcategory</i>	<i>Examples</i>	<i>Description</i>
<i>WaaS</i> Workflow as a service, multiple workflow instances submitted to the application	<i>Elastic resource provisioning</i>	BYUN et al., 2011	Partitioned balanced time scheduling (PBTS) that estimates minimum amount of resources to executed workflow within user-specified finish time
		<i>Merge-based multiple DAG scheduling</i>	HÖNIG; SCHIFFMANN, 2006
		ZHAO; SAKELLARIOU, 2006	Four composition approaches to merge DAGs plus two fair scheduling policies
	<i>Hierarchical multiple DAG scheduling</i>	STAVRINIDES; KARATZA, 2011	Dynamic scheduling based on earliest deadline first (EDF), highest level first (HLF), or least space-time first (LSTF)
		TSAI et al., 2012	Combination of clustering and a balance of task start time and fitness of idle time slots
		IVERSON, 1999	Decentralized hierarchical scheduling strategy
		YU; SHI, 2008	Planner-guided scheduling based on HEFT
	<i>Auto-scaling of cloud resources</i>	MALAWSKI et al., 2012	Efficient management of ensembles made of interrelated workflows; complete as many as possible under budget and deadline constraints
		MAO; HUMPHREY, 2011	Each DAG specified with a deadline; earliest deadline first algorithm
		ZHOU; HE; LIU, 2016	Framework for hosting WaaS in IaaS cloud; optimize run time and minimize cost

Table 5 – Summary of workflow scheduling strategies (cont. IV)

<i>Category</i>	<i>Subcategory</i>	<i>Examples</i>	<i>Description</i>
<i>Robust scheduling</i> Able to absorb uncertainties such as performance fluctuation and failure	<i>Makespan minimization</i>	CANON et al., 2008	Robustness and makespan are correlated; well-performing schedules tend to be more robust
		MALEWICZ et al., 2007	Just-in-time (dynamic) scheduling
		CORDASCO et al., 2010	Dynamic scheduling
		SAKELLARIOU; ZHAO, 2004	Low-cost rescheduling policy; risk of extra communication and synchronization overheads
		ZHENG; SAKELLARIOU, 2013	Monte Carlo approach; results show that best schedule always outperforms deterministic one
	<i>Slack maximization</i>	POOLA et al., 2014	Critical-path based heuristic with deadline and budget constraints
		SHI et al., 2006	ϵ -method to address makespan and robustness initially based on HEFT
	<i>Fault-tolerant</i>	HWANG; KESSELMAN, 2003	Characterization of failure handling techniques. Two workflow-level techniques of alternative task and redundancy.
		WANG; CHEN, 2012	Checkpointing and rescheduling to handle task failures
	<i>Spot instance</i>	POOLA et al., 2014a	Spot and on-demand instances; deadline, cost, and reliability
ZHOU; HE; LIU, 2016		Hybrid instance configuration; deadline management	
<i>Hybrid environment</i>	-	BITTENCOURT; MADEIRA, 2011	HCOC: Hybrid Cloud Optimized Cost algorithm
		FARD et al., 2012	Intersection of game theory and computer science

Table 5 – Summary of workflow scheduling strategies (cont. V)

<i>Category</i>	<i>Subcategory</i>	<i>Examples</i>	<i>Description</i>
<i>Data-intensive</i> Data-aware workflow scheduling	-	PARK; HUMPHREY, 2008	Bandwidth allocation technique to speed up file transfers
		BHARATHI; CHERVENAK, 2009	Data staging techniques to reduce execution time
		RAMA et al., 2007	Dynamic method to minimize storage space by removing files at runtime
<i>Energy-aware</i>	-	SHARIFI et al., 2013	PASTA, two-phase power aware solution; fair tradeoff between power and makespan
		VENKA; FRANZ, 2005	DVS (dynamic voltage scaling), resource hibernation, memory optimizations
		PRUHS et al., 2008	Dual-objective (makespan and energy minimization); data transmission not considered
		KIM et al., 2007	DVS-based to optimize energy and deadline
		LEE; ZOMAYA, 2011	Makespan and energy optimization via novel objective function
		MEZMAZ et al., 2011	Multi-objective optimization; hybrid metaheuristic genetic algorithm
		YASSA et al., 2013	Particle swarm optimization for multi-objective scheduling (makespan, cost, energy)
		XIAO; HU; ZHANG, 2013	Data-intensive properties, energy-aware scheduling
MISHRA et al., 2003	Deadline constraint and energy; online dynamic power management		

The works presented in the table cover the aspects defined in the taxonomy proposed by (WU; WU; TAN, 2015), and several of these aspects fit the challenges addressed by MPSF, such as hybrid environments and data-intensive workflows.

2.2.2.4 A survey on resource allocation in high performance distributed computing systems

Hussain et al. start their analysis by presenting a classification of HPC systems distinguishing clusters, grids, and clouds, as depicted in Figure 5.

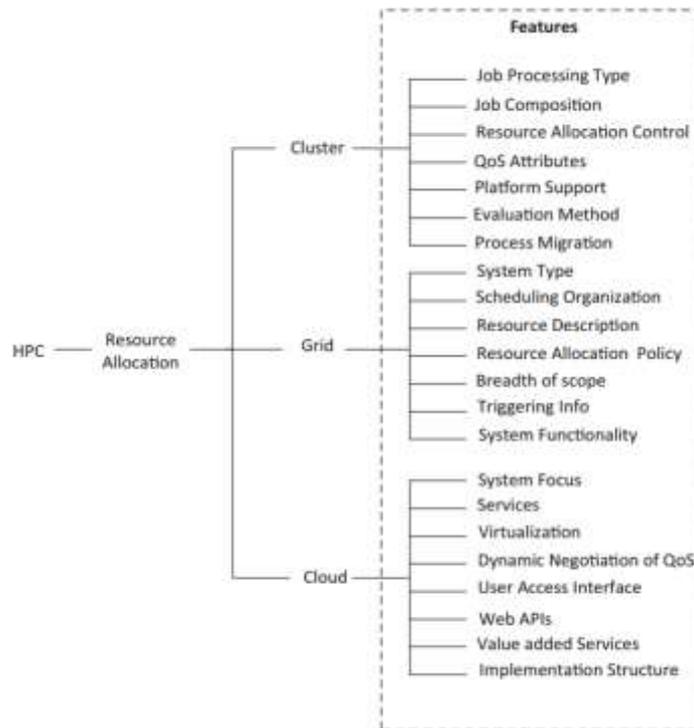


Figure 5 – HPC systems and their attributes according to (HUSSAIN et al., 2013)

The authors state the feasibility of having an HPC cloud as clusters with 50,000 cores had been run on Amazon EC2 for scientific applications. The distinguishing characteristics of cloud HPC are the transparent negotiation of resources, the provisioning of dynamically scalable and often virtualized resources over the Internet, and the low initial infrastructure cost for consumers. The survey focuses on cloud computing systems such as Amazon EC2, Eucalyptus, Google AppEngine, and OpenStack. However, no further details on resource allocation and scheduling are provided or analyzed.

2.2.2.5 *Cloud resource provisioning: survey, status and future research directions & A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges*

(SINGH; CHANA, 2015b) and (SINGH; CHANA, 2016) from Singh and Chana present a comprehensive study of cloud provisioning and scheduling, respectively. In (SINGH; CHANA, 2015b) the authors present a timeline depicting the evolution of provisioning solutions for cloud computing. This timeline is presented in Figure 6.

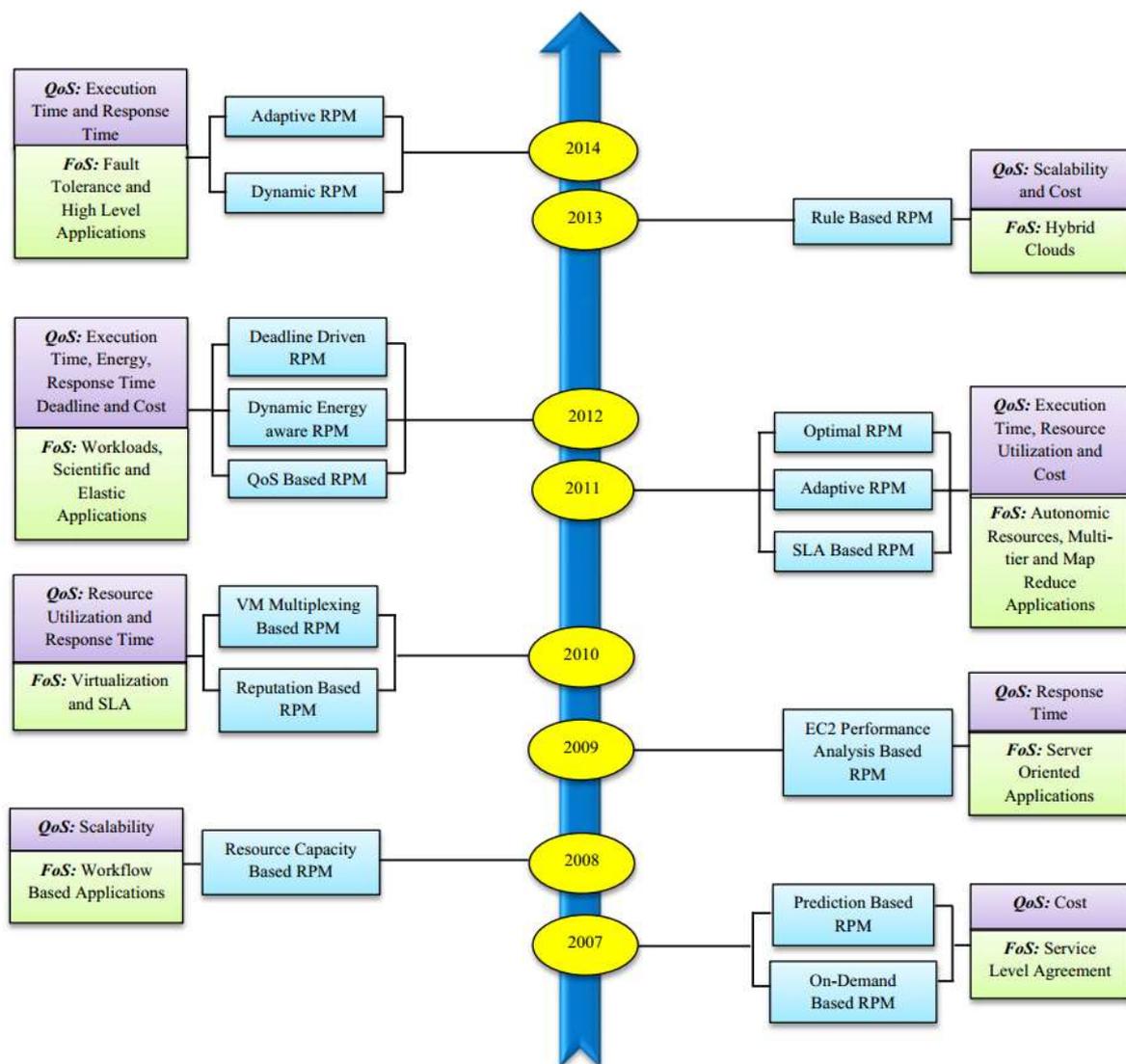


Figure 6 – Resource provisioning evolution by (SINGH; CHANA, 2015b)

The Focus of Study (FoS) of the first solutions in 2007 was either using prediction methods to determine the provisioning solution or using on-demand solutions. The main QoS parameter in both cases was cost. Moving to 2008 the main QoS parameter switched to scalability while the focus moved to workflows and workflow-based applications, with solutions based on the resource capacity. In 2009 the studies focused on the Amazon EC2 service, aiming at providing enough response time for server-oriented applications. In 2010 the focus was on VM multiplexing and reputation based solutions, either with the objective of improving resource utilization and optimizing response time related to virtualization techniques and SLA parameters. In 2011 the focus was on optimized solutions, adaptive solutions, and SLA-based solutions to improve execution time (makespan), resource utilization, and cost. The focus of study was on autonomic resources, multi-tier resources and MapReduce applications.

Moving towards more recent works, in 2012 the main topics were deadline driven provisioning, dynamic energy-aware solutions, and QoS-based solutions usually focusing on multiple objectives. QoS parameters included execution time, energy, response time, deadline, and cost. The focus was back to workloads, including scientific and elastic applications. In 2013 the studies included rule-based solutions to improve scalability and cost for hybrid clouds, marking the beginning of the studies directly addressing more complex cloud environments in terms of hierarchy and organization of resources. Finally, in 2014 the authors highlight the adaptive and dynamic provisioning solutions focusing on execution time and response time for fault-tolerant and high-level applications.

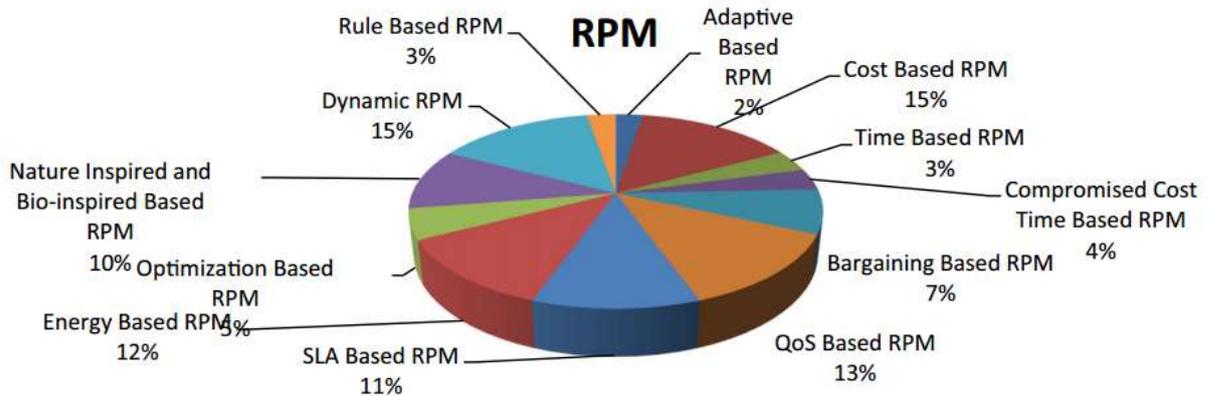


Figure 7 – Resource provisioning mechanisms classified by (SINGH; CHANA, 2015b)

The authors provide a quantitative analysis of resource provisioning mechanisms by classifying them by their type, as shown in Figure 7. According to their findings, the majority of solutions either focus on dynamic provisioning (15% of the works), cost (15%), QoS parameters (13%), energy (12%), and SLAs (11%). Other relevant results include nature-inspired and bio-inspired solutions (10%) and solutions based on bargaining for resources (7%). The other types have a percentage lower than 5%. Considering the QoS parameters, the authors provide the results presented in Figure 8.

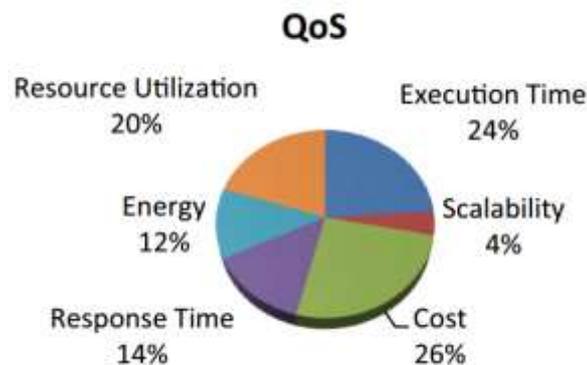


Figure 8 – QoS parameters used in provisioning mechanisms according to (SINGH; CHANA, 2015b)

The distinction between this classification and the categories used to produce the results from Figure 7 is the emphasis on the utilization of a particular parameter to design the provisioning solution versus focusing on QoS and selecting a few specific QoS metrics. In this case the most common metrics are cost and execution time (makespan), constituting 50% of the works. Other metrics include resource utilization (20%), response time (14%), energy (12%), and scalability (4%).

The authors also analyze scheduling solutions (SINGH; CHANA, 2016). The timeline for scheduling solutions built by the authors is depicted in Figure 9.

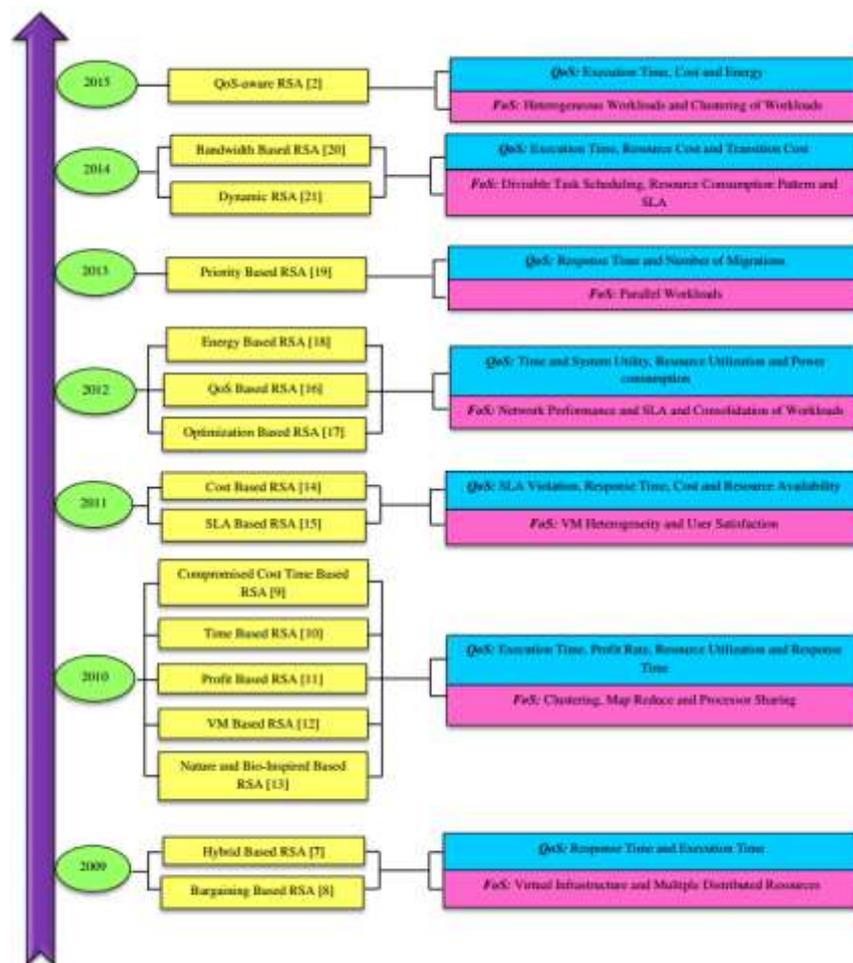


Figure 9 – Evolution of scheduling solutions

The first scheduling solutions in 2009 focused on hybrid approaches (combining response time and execution time, for instance) and bargaining-based solutions. The focus in this case was on the virtual infrastructure and on multiple distributed resources. Moving to 2010 leads to a more diverse range of options, from time-based and profit-based solutions to ones focusing on the virtual machines and on nature-inspired algorithms. The QoS parameters included execution time, profit rate, resource utilization, and response time. These parameters reveal a certain balance between QoS parameters for cloud consumers and for cloud providers. Afterwards, in 2011 the efforts focused on cost-based and SLA-based scheduling solutions, addressing issues related to VM heterogeneity and user satisfaction (e.g., SLA violation, response time). In 2012 the efforts were redirected to energy and optimization, followed by prioritization policies in 2013. In 2014 and 2015 the works focused on dynamic scheduling, bandwidth, and QoS parameters such as execution time, cost, and energy.

2.2.3 Taxonomy Analysis and Consolidation

The surveys analyzed propose or use some form of taxonomy to classify and organize the related work. This subsection analyzes a selection of these taxonomies and then provides a consolidated version used to classify the related work for the purposes of this work.

2.2.3.1 *A Survey of Various Workflow Scheduling Algorithms in Cloud Environment*

(BALA; CHANA, 2011) defines nine categories to classify resource management and scheduling solutions: time, cost, scalability, scheduling success rate, makespan, speed, resource utilization, reliability, and availability. These properties are illustrated by Figure 10.

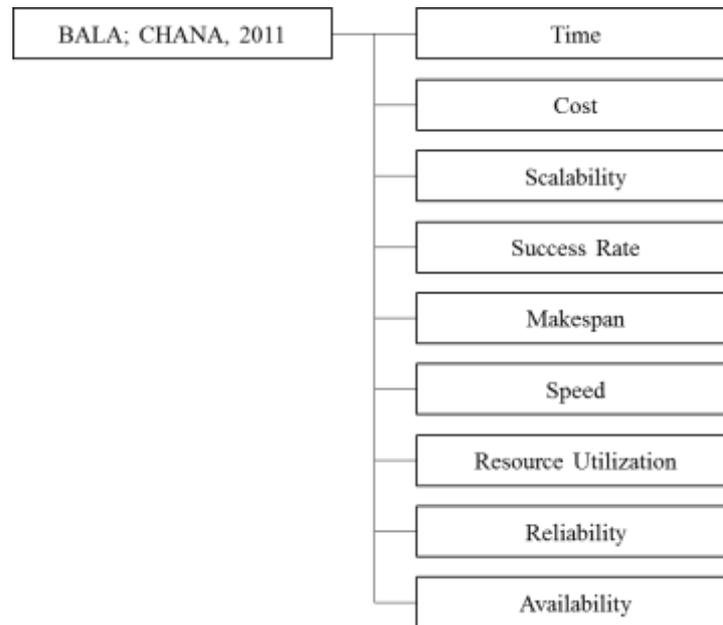


Figure 10 – Properties used by (BALA; CHANA, 2011) to classify the related work

Time, speed, and makespan are directly related. Resource utilization is related to the efficiency of utilization of resources, which is a fundamental aspect of any algorithm. Reliability and availability aspects were not identified in any of the solutions analyzed by the authors.

2.2.3.2 *Towards inter-cloud schedulers: A survey of meta-scheduling approaches*

(SOTIRIADIS et al., 2011) classifies the solutions in terms of flexibility, scalability, interoperability, heterogeneity, local autonomy, load balancing, information exposing, real-time data, scheduling history records, unpredictability management, geographical distribution, SLA compatibility, rescheduling, and intercloud compatibility. These properties are summarized in Figure 11.



Figure 11 – Properties used by (SOTIRIADIS et al., 2011) to classify references

Several properties are relevant for heterogeneous environments, such as local autonomy and geographical distribution. Others are correlated, such as scalability, unpredictability management, and rescheduling.

2.2.3.3 *Workflow scheduling in cloud: a survey*

(WU; WU; TAN, 2015) uses nine categories to classify their references, as illustrated by Figure 12.

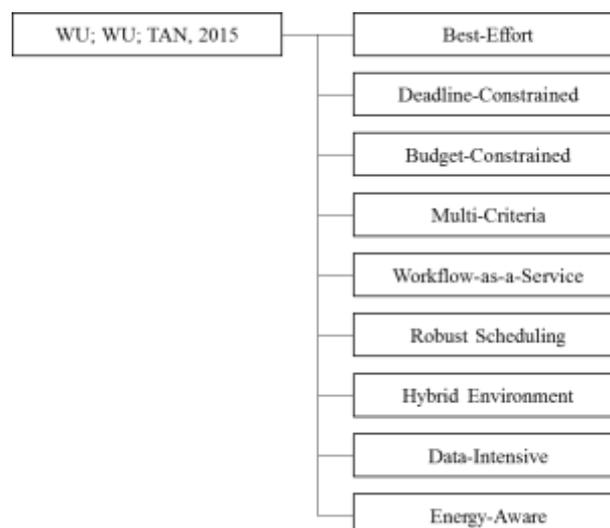


Figure 12 – Categories defined by (WU; WU; TAN, 2015)

The authors also mention other properties such as makespan (which fits the Best-Effort category). Moreover, the multi-criteria category represents the convergence of several objective functions, such as cost and performance. Workflow-as-a-Service (WaaS) is the scheduling of multiple workflows onto a cloud infrastructure. Robust scheduling refers both to reliability and to performance fluctuations, both factors that can affect the performance and consequently the effectiveness of a schedule. Finally, hybrid environments, data-intensive workflows, and energy-aware scheduling represent the novel challenges (WU; WU; TAN, 2015) in terms of cloud scheduling and cloud resource management.

2.2.3.4 *Cloud resource provisioning: survey, status and future research directions*

(SINGH; CHANA, 2015b) defines a taxonomy based on twelve properties, as depicted in Figure 13.

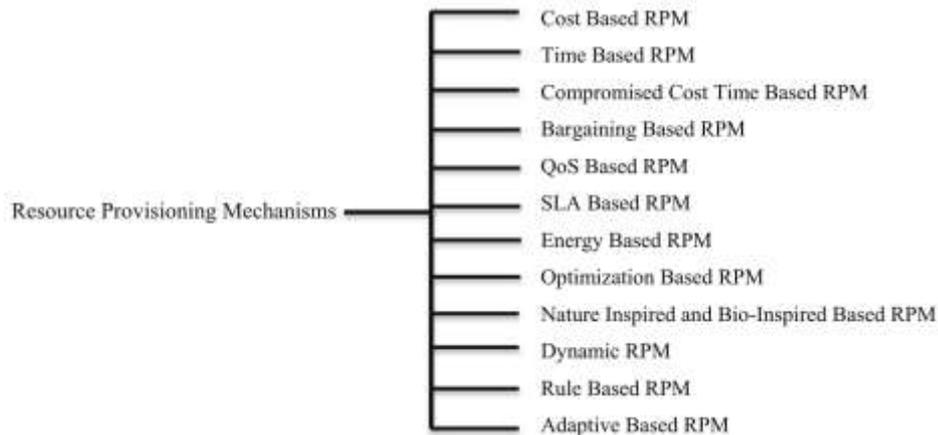


Figure 13 – Taxonomy proposed by (SINGH; CHANA, 2015b) (figure extracted from reference)

Moreover, each category is subdivided into a sub-taxonomy. Figure 14 presents the taxonomy for cost-based resource provisioning mechanisms.

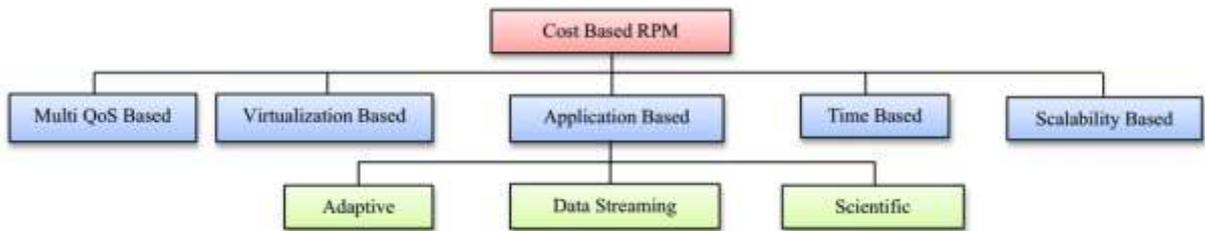


Figure 14 – Taxonomy for Cost Based resource provisioning mechanisms (RPMs)
(extracted from (SINGH; CHANA, 2015b))

The cost-based solutions are organized in multi-QoS-based (multiple cost-related objectives are addressed), virtualization-based (minimize total price of VM acquisition), application based (subdivided into adaptive, data streaming, and scientific applications; in all cases the objective is to minimize cost of application execution), time-based (minimize cost and time), and scalability-based (reduce cost based on auto-scaling capabilities).

Figure 15 presents the taxonomy for time-based solutions. Three subcategories are proposed: deadline based (main one), and deadline combined to budget and to energy, respectively. Note that this category also includes solutions whose objective function is bound by makespan requirements.

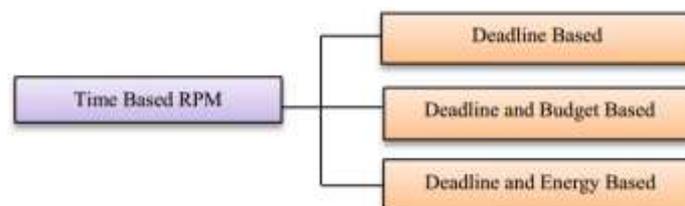


Figure 15 – Taxonomy for time-based solutions (extracted from (SINGH; CHANA, 2015b))

Figure 16 presents the taxonomy for compromised cost-time solutions. In this case there is a need to consider budget and deadline as QoS parameters to execute cost-constrained workflows. Workloads are defined as an abstraction of work executed by one or more (cloud) instances, while workflow is a set of interrelated tasks and their distribution among available resources.



Figure 16 – Taxonomy for compromised cost-time solutions
(extracted from (SINGH; CHANA, 2015b))

Figure 17 presents the taxonomy for bargaining-based solutions, which includes negotiation among resource provider and resource consumers. Subcategories comprise market-oriented based solutions (resources are provided based on QoS requirements of workloads in a cloud market), auction-based (consumer uses bidding policy to choose required resource set), and negotiation-based (user and provider negotiate QoS parameters in the form of a written SLA).

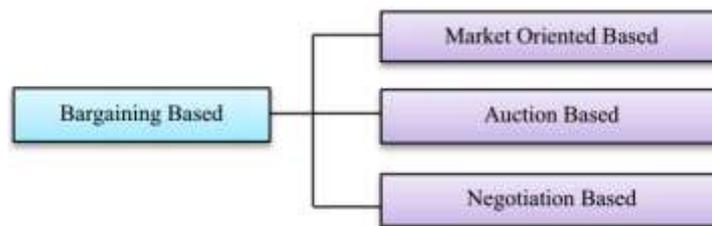


Figure 17 – Taxonomy for bargaining-based solutions (extracted from (SINGH; CHANA, 2015b))

Figure 18 presents the taxonomy for QoS-based solutions, including parameters such as scalability, reliability, resource utilization, energy, security, and availability. These are properties typically considered in other surveys.

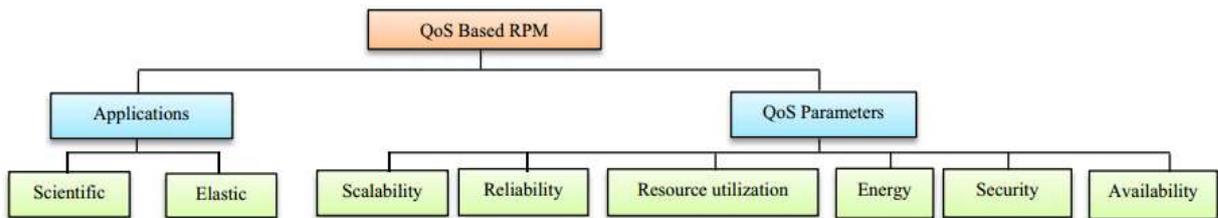


Figure 18 – Taxonomy for QoS-based solutions (extracted from (SINGH; CHANA, 2015b))

Figure 19 presents the taxonomy for SLA-based solutions. SLA-based architectures have been designed so that users and providers interact via user interfaces. Virtualization-based solutions rely on SLA violation rate and SLA deviation metrics for contractual verification. Workload-based solutions are based on SLAs defined relative to operational metrics. Finally, autonomic-based solutions automatically impose penalties to providers in order to compensate consumers (e.g., in case of missing a deadline).

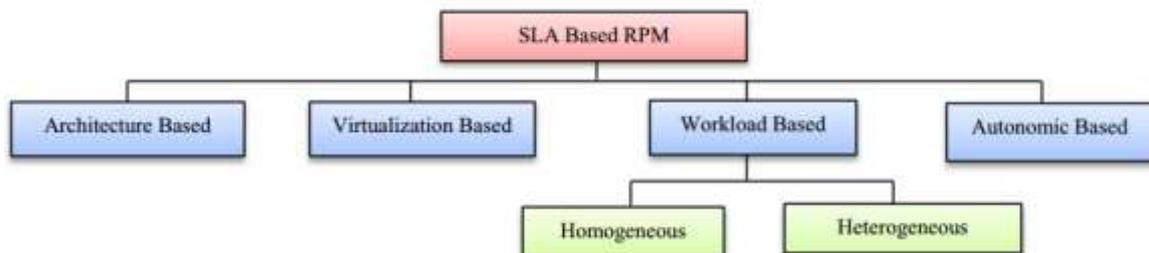


Figure 19 – Taxonomy for SLA-based solutions (extracted from (SINGH; CHANA, 2015b))

Figure 20 presents the taxonomy for energy-based solutions. In each case the energy constraint is combined to another element, such as deadline constraints, virtualization mechanisms, and SLAs for contractual agreement regarding energy consumption.

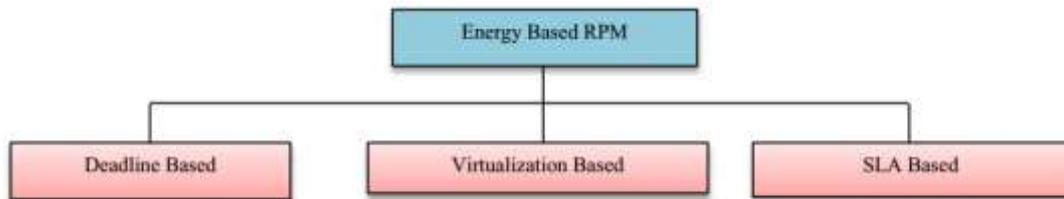


Figure 20 – Taxonomy for Energy-based solutions (extracted from (SINGH; CHANA, 2015b))

Figure 21 presents the taxonomy for optimization-based solutions. Several parameters and combinations of parameters are considered, such as energy, SLAs, deadline, QoS, application properties, task properties, and failure awareness.

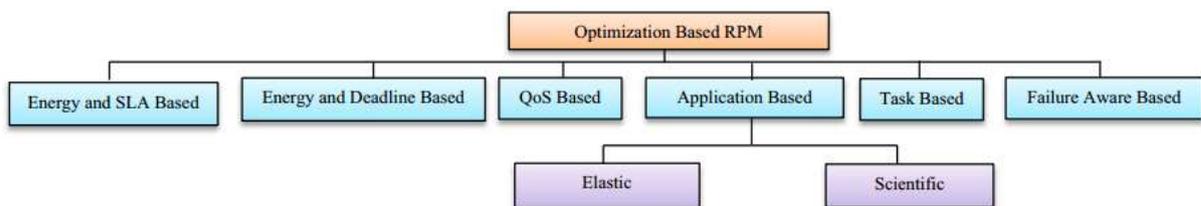


Figure 21 – Taxonomy for optimization-based solutions (extracted from (SINGH; CHANA, 2015b))

Figure 22 presents the taxonomy for nature and bio-inspired solutions. These are solutions that use evolutionary algorithms, genetic-based algorithms, ant colony/firefly/honey bee algorithms, and particle swarm optimization.

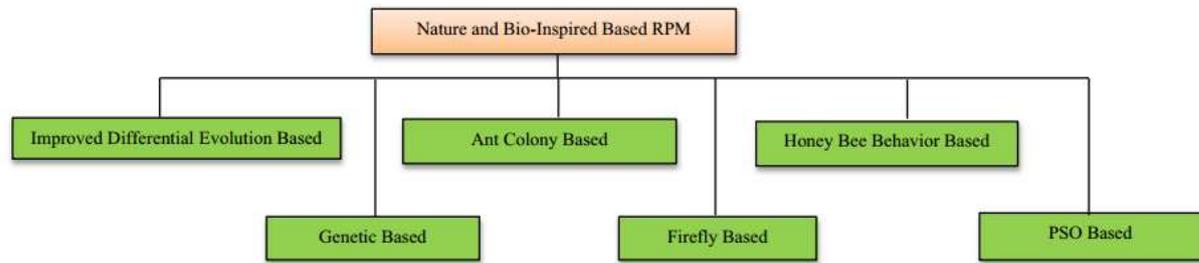


Figure 22 – Taxonomy for nature/bio-inspired solutions (extracted from (SINGH; CHANA, 2015b))

Figure 23 presents the taxonomy for dynamic solutions. Threshold-based provisioning refers to the definition of a threshold for resource utilization; if the threshold is exceeded the resources are dynamically reallocated. Energy-based dynamic solutions continuously adjust load to minimize energy consumption. Other strategies include heterogeneous workloads, application-based dynamic load and resource allocation, auction-based resource allocation, and the utilization of deadlines, priorities, and SLA parameters to dynamically configure the provisioning.

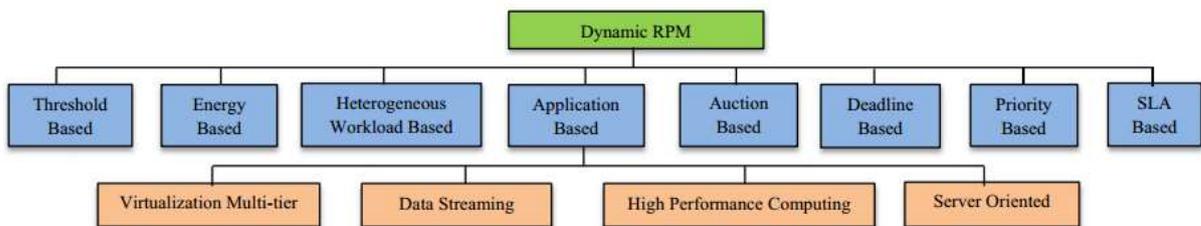


Figure 23 – Taxonomy for dynamic solutions (extracted from (SINGH; CHANA, 2015b))

Figure 24 presents the taxonomy for rule-based solutions. Failure-based solutions usually reserve resources to be used in case of any errors, thus avoiding performance degradation. For hybrid clouds different mechanisms have been proposed to reduce overprovisioning and underprovisioning. Finally, semantic-based solutions define resource ontologies to address homogeneous workloads.



Figure 24 – Taxonomy for rule-based solutions (extracted from (SINGH; CHANA, 2015b))

Figure 25 presents the taxonomy for adaptive-based solutions. Prediction is one of the techniques used to perform an efficient resource provisioning to avoid underutilization (or overutilization) of resources. In parallel, online bin packing mechanisms are used to solve the combinatorial NP-hard problem property of autonomic provisioning of resources.

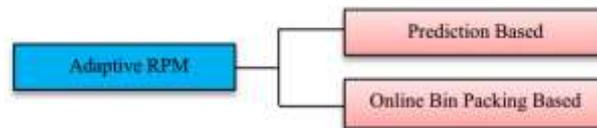


Figure 25 – Taxonomy for adaptive-based solutions (extracted from (SINGH; CHANA, 2015b))

2.2.3.5 Consolidation

The consolidated taxonomy is a result of merging the main properties observed in the surveys and in the analyzed publications to the interests of this work, mainly related to addressing the requirements of heterogeneous environments composed by multiple environments (e.g., hybrid clouds and multicloud scenarios), with data-intensive workflows (a common property of future workflows) and high level of dynamic mechanisms. The result is the taxonomy presented in Figure 26. The properties were selected either by identifying the commonalities between the works analyzed and also based on the interests of this research (data-intensive workflows, hybrid and multicloud scenarios, performance fluctuation, and reliability).

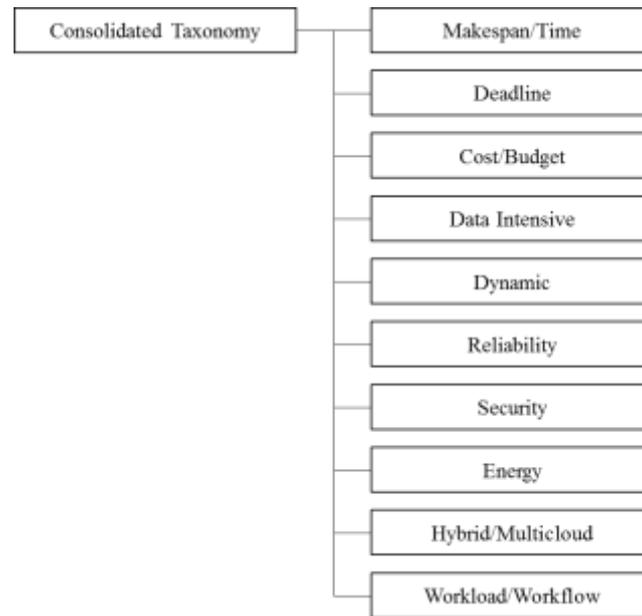


Figure 26 – Consolidated Taxonomy with categories addressing the main properties observed in the surveys and analyzed publications while addressing the interests of this work

The following categories compose the taxonomy:

- The Makespan/Time category encompasses all aspects related to run time and time-based optimization.
- The Deadline category encompasses aspects also related to time but associated to predefined limits to finish a workflow – the central idea is not to finish the execution of a workflow as fast as possible, but simply to address a specific deadline and possibly save resources (i.e., reduce resource allocation) as long as the deadline is met.
- The Cost/Budget category encompasses all aspects related to financial cost and benefits, such as cost minimization and budget limitation.
- The Data Intensive category was selected to identify the works that effectively encompass one or more aspects inherent to data-intensive workflows.

- The Dynamic category allows the identification of solutions that employ some form of dynamic mechanism to continuously adjust the scheduling decision. This is a typical method to address issues related to unpredictability, such as performance fluctuation.
- The Reliability category was selected to identify works that encompass some form of reliability-related aspect, such as selecting nodes in a way to minimize the chances of failure, or providing mechanisms to circumvent failures.
- The Security category was selected to identify works that consider any aspect of security (in the sense of confidentiality).
- The Energy category is related energy-aware scheduling mechanisms.
- The Hybrid/Multicloud category was selected to identify works that address the specific requirements of hybrid clouds and multicloud scenarios.
- The Workload/Workflow category was selected to identify works that address the requirements for scheduling workflows on clouds.

Compared to the other taxonomies, the proposed one encompasses some of the fundamental properties connected to the QoS components that govern the scheduling decisions, such as makespan, cost, deadline, energy, etc. These properties are fully or at least partially covered by the other taxonomies, such as (BALA; CHANA, 2011), with cost, makespan, and reliability; (SOTIRIADIS et al., 2011), with unpredictability management (closely related to dynamic properties and reliability) and rescheduling; (WU; WU; TAN, 2015), with deadline, budget, reliability, and energy; and (SINGH; CHANA, 2015b), with cost, time, and energy. In addition, the proposed taxonomy encompasses some of the attributes of interest to this work, such as hybrid and multicloud aspects, and workflow resource management.

Table 6 – Summary of identified related work classified using the consolidated taxonomy (cont. I)

MK = Makespan/Time; DL = Deadline; CT = Cost; DT = Data Intensive/Transfers;
 DY = Dynamic (re)placement; RL = Reliability; SC = Security; EN = Energy;
 HM = Hybrid/Multicloud; WL = Workload/Workflow;
 . = Not addressed; x = Fully addressed; o = Partially addressed
 * = Selected for further analysis

<i>Work</i>	<i>Highlights</i>	<i>MK</i>	<i>DL</i>	<i>CT</i>	<i>DT</i>	<i>DY</i>	<i>RL</i>	<i>SC</i>	<i>EN</i>	<i>HM</i>	<i>WL</i>
TOPCUOGLU et al., 2002	HEFT (Heterogeneous Earliest Finish Time)	x
* WEISSMAN; GRIMSHAW, 1996	Wide-area scheduling with dynamic load balancing	.	.	.	x	x	.	.	.	o	.
SCHWIEGELSHOHN; YAHYAPOUR, 1999	Integration to conventional schedulers.	o	.
ANAND; GHOSE; MANI, 1999	ELISA, decentralized dynamic algorithm	x	.	.	.	o	.
DAVAL-FREROT; LACROIX; GUYENNET, 2000	Federation of resource traders	o	.
SUBRAMANI et al., 2002	Redundantly distribute job to multiple sites to increase backfilling	o	.
BUTT; ZHANG; HU, 2003	Reduce maximum job waiting time in the queue	x	o	.
ANDRADE et al., 2003	Community of peers for brokering	o	.	.	.	o	.
LAI et al., 2005	Auction-based scheduling.	o	.
SHAH et al., 2007	Load balancing on arrival	o	.	.	.	o	.
IOSUP et al., 2008	Delegated matchmaking, local vs remote usage	.	o	.	.	o	.	.	.	o	.

Table 6 – Summary of identified related work classified using the consolidated taxonomy (cont. II)

MK = Makespan/Time; DL = Deadline; CT = Cost; DT = Data Intensive/Transfers;
 DY = Dynamic (re)placement; RL = Reliability; SC = Security; EN = Energy;
 HM = Hybrid/Multicloud; WL = Workload/Workflow;
 . = Not addressed; x = Fully addressed; o = Partially addressed
 * = Selected for further analysis

<i>Work</i>	<i>Highlights</i>	<i>MK</i>	<i>DL</i>	<i>CT</i>	<i>DT</i>	<i>DY</i>	<i>RL</i>	<i>SC</i>	<i>EN</i>	<i>HM</i>	<i>WL</i>
DIAS; BUYYA; VENUGOPAL, 2008	Improve average response time	o	o	.
LEAL; HUEDO; LLORENTE, 2009	Decentralized model that improves makespan	x	o	.
FÖLLING et al., 2009	Fuzzy approach for decentralized grids	o	o	.
RODERO et al., 2010	Brokering for multiple grids.	o	.
WANG et al., 2010	Bidding system for resource selection	o
HUANG et al., 2013	Community-aware decentralized dynamic scheduling	o	.	.	.	o	.	.	.	o	.
RODERO et al., 2009	Backfilling strategy based on dynamic information	x	.	.	.	x	.	.	.	o	.
XU et al., 2011	Justice-based scheduling	x	.	.	.	o	.	.	.	o	.
SIH; LEE, 1993	Dynamic level scheduling (DLS)	x	.	.	o	x	x
KWOK; HMAD, 1996	Dynamic Critical Path (DCP)	o	.	.	.	x	x
ARABNEJAD; BARBOSA, 2014	Predict Earliest Finish Time (PEFT)	x	.	.	o	x
BILGAIYAN; SAGNIKA; DAS, 2014	Cat Swarm Optimization	.	.	x	x	x
* CHEN; ZHANG, 2009	Ant Colony Optimization	x	x	x	.	.	x	.	.	.	x

Table 6 – Summary of identified related work classified using the consolidated taxonomy (cont. III)

MK = Makespan/Time; DL = Deadline; CT = Cost; DT = Data Intensive/Transfers;
 DY = Dynamic (re)placement; RL = Reliability; SC = Security; EN = Energy;
 HM = Hybrid/Multicloud; WL = Workload/Workflow;
 . = Not addressed; x = Fully addressed; o = Partially addressed
 * = Selected for further analysis

<i>Work</i>	<i>Highlights</i>	<i>MK</i>	<i>DL</i>	<i>CT</i>	<i>DT</i>	<i>DY</i>	<i>RL</i>	<i>SC</i>	<i>EN</i>	<i>HM</i>	<i>WL</i>
YU; BUYYA; THAM, 2005	Deadline partitioning	.	x	o	x
YUAN et al., 2008	Float time amortization	.	x	o	x
YUAN et al., 2009	Path-based deadline partition	.	x	x
ABRISHAMI; NAGHIBZADEH; EPEMA, 2013	Partial Critical Path (PCP)	o	x	o	x
YU; BUYYA, 2006	Genetic algorithm to optimize cost with deadline constraint	.	x	x	x
WU et al., 2010	PSO to minimize cost with deadline constraint	o	x	x	x
* RODRIGUEZ; BUYYA, 2014	PSO considering performance variation and VM boot time	.	x	x	.	x	x
YU et al., 2009	Greedy time-cost distribution	.	.	x	x
ZHENG; SAKELLARIOU, 2011	Budget-constrained HEFT	.	.	x	.	x	x
ARABNEJAD; BARBOSA, 2014a	Aggregation-based budget distribution	.	.	x	x
SAKELLARIOU et al., 2007	LOSS and GAIN approaches	x	.	x	x
LIN; WU, 2013	Minimize end-to-end delay	o	.	x	x

Table 6 – Summary of identified related work classified using the consolidated taxonomy (cont. IV)

MK = Makespan/Time; DL = Deadline; CT = Cost; DT = Data Intensive/Transfers;
 DY = Dynamic (re)placement; RL = Reliability; SC = Security; EN = Energy;
 HM = Hybrid/Multicloud; WL = Workload/Workflow;
 . = Not addressed; x = Fully addressed; o = Partially addressed
 * = Selected for further analysis

<i>Work</i>	<i>Highlights</i>	<i>MK</i>	<i>DL</i>	<i>CT</i>	<i>DT</i>	<i>DY</i>	<i>RL</i>	<i>SC</i>	<i>EN</i>	<i>HM</i>	<i>WL</i>
ZENG; VEERAVALLI; LI, 2012	Backtracking and continuous cost evaluation	o	.	x	.	x	x
LI et al., 2011	CCSH to minimize makespan and cost	x	.	x	x
GARG; BUYYA; SIEGEL, 2010	Optimize makespan and cost	x	.	x	x
* FARD et al., 2012	Multi-objective scheduling	x	.	x	o	.	x	.	x	.	x
DOGAN; OZGUNER, 2002	Performance and reliability optimization	x	.	x	.	x	x	.	.	.	x
HAKEM; BUTELLE, 2007	Performance and reliability optimization	x	x	.	.	.	x
DONGARRA et al., 2007	Reliable HEFT	x	xx
LEE; SUBRATA; ZUMAYA, 2009	Optimize makespan and resource utilization	x	.	.	.	x	x
PRODAN; WIECZOREK, 2010	Dynamic programming	x	.	x	.	.	x	.	.	.	x
BESSAI et al., 2012	Pareto-based; execution time and cost	x	.	x	x
YU et al., 2007	Minimize execution time and cost	x	x	x	x
TALUKDER et al., 2009	Similar to YU et al., 2007	x	x	x	x
BYUN et al., 2011	Deadline optimization based on delaying	.	x	.	.	x	x

Table 6 – Summary of identified related work classified using the consolidated taxonomy (cont. V)

MK = Makespan/Time; DL = Deadline; CT = Cost; DT = Data Intensive/Transfers;
 DY = Dynamic (re)placement; RL = Reliability; SC = Security; EN = Energy;
 HM = Hybrid/Multicloud; WL = Workload/Workflow;
 . = Not addressed; x = Fully addressed; o = Partially addressed
 * = Selected for further analysis

<i>Work</i>	<i>Highlights</i>	<i>MK</i>	<i>DL</i>	<i>CT</i>	<i>DT</i>	<i>DY</i>	<i>RL</i>	<i>SC</i>	<i>EN</i>	<i>HM</i>	<i>WL</i>
ZHAO; SAKELLARIOU, 2006	Merge multiple DAGs	x	x
TSAI et al., 2012	Combination of DAG merging techniques	x	x
IVERSON, 1999	Hierarchical scheduling	o	o
YU; SHI, 2008	Based on HEFT	x	x
* MALAWSKI et al., 2012	Auto-scaling of resources	o	x	x	o	x
MAO; HUMPHREY, 2011	Multiple DAGs; deadline-based	o	x	o	.	x	x
ZHOU; HE; LIU, 2016	Multiple workflows, optimize time and cost	x	x	x	x
MALEWICZ et al., 2007	Dynamic scheduling	x	o	.	.	.	x
CORDASCO et al., 2010	Dynamic scheduling	x	o	.	.	.	x
* SAKELLARIOU; ZHAO, 2004	Rescheduling policies	x	.	o	.	x	x
ZHENG; SAKELLARIOU, 2013	Monte Carlo approach	x	.	o	.	x	x
POOLA et al., 2014	Critical-path heuristic	x	x	x	.	.	x	.	.	.	x
SHI et al., 2006	Makespan and robustness	x	x	.	.	.	x
HWANG; KESSELMAN, 2003	Fault-tolerant scheduling	x	.	.	.	x

Table 6 – Summary of identified related work classified using the consolidated taxonomy (cont. VI)

MK = Makespan/Time; DL = Deadline; CT = Cost; DT = Data Intensive/Transfers;
 DY = Dynamic (re)placement; RL = Reliability; SC = Security; EN = Energy;
 HM = Hybrid/Multicloud; WL = Workload/Workflow;
 . = Not addressed; x = Fully addressed; o = Partially addressed
 * = Selected for further analysis

<i>Work</i>	<i>Highlights</i>	<i>MK</i>	<i>DL</i>	<i>CT</i>	<i>DT</i>	<i>DY</i>	<i>RL</i>	<i>SC</i>	<i>EN</i>	<i>HM</i>	<i>WL</i>
* WANG; CHEN, 2012	Fault-tolerant scheduling	.	x	x	.	o	x	.	.	.	x
* POOLA et al., 2014a	Spot instances	.	x	x	.	x	x	.	.	.	x
* BITTENCOURT; MADEIRA, 2011	Hybrid clouds; iteratively resch. tasks until mksp. < deadline	x	x	o	.	x	.	.	.	x	x
PARK; HUMPHREY, 2008	Bandwidth speedup data-intensive	o	.	.	x	x
BHARATHI; CHERVENAK, 2009	Data staging	x	o	.	x	x
RAMA et al., 2007	Dynamic storage mgmt.	o	.	.	x	x
SHARIFI et al., 2013	Power aware scheduling	x	.	.	.	o	.	.	x	.	x
VENKA; FRANZ, 2005	Dynamic voltage scaling	.	x	o	.	o	.	.	x	.	x
PRUHS et al., 2008	Makespan and energy	x	x	.	x
KIM et al., 2007	Energy and deadline	.	x	.	.	o	.	.	x	.	x
LEE; ZOMAYA, 2011	Makespan and energy	x	x	.	x
MEZMAZ et al., 2011	Makespan and energy	x	.	.	.	o	.	.	x	.	x
YASSA et al., 2013	Particle swarm optimization	x	.	x	.	o	.	.	x	.	x

Table 6 – Summary of identified related work classified using the consolidated taxonomy (cont. VII)

MK = Makespan/Time; DL = Deadline; CT = Cost; DT = Data Intensive/Transfers;
 DY = Dynamic (re)placement; RL = Reliability; SC = Security; EN = Energy;
 HM = Hybrid/Multicloud; WL = Workload/Workflow;
 . = Not addressed; x = Fully addressed; o = Partially addressed
 * = Selected for further analysis

<i>Work</i>	<i>Highlights</i>	<i>MK</i>	<i>DL</i>	<i>CT</i>	<i>DT</i>	<i>DY</i>	<i>RL</i>	<i>SC</i>	<i>EN</i>	<i>HM</i>	<i>WL</i>
XIAO; HU; ZHANG, 2013	Data-intensive, energy-aware	x	.	.	x	o	.	.	x	.	x
MISHRA et al., 2003	Dynamic, deadline, energy	.	x	.	.	x	.	.	x	.	x
ZHANG et al., 2007	Forecast prototype and SLA compensation	.	.	x
ZHANG et al., 2007a	Historical information, forecasting	.	.	x
BERL et al., 2010	Energy efficiency	o	.	.	x	.	.
XIAO et al., 2010	Reputation-based QoS provisioning	o	o	x
TIAN; CHEN, 2011	MapReduce on public clouds	.	x	x
IQBAL et al., 2011	Multi-tier applications	o	.	o
* VECCHIOLA et al., 2012	Deadline-driven, scientific applications, hybrid clouds	.	x	x	o
ZHANG et al., 2012	Energy-aware, scheduling delay	.	o	.	.	o	.	.	x	.	.
CALHEIROS et al., 2012	Aneka platform; QoS-driven, hybrid	.	x	.	.	o	.	.	.	x	.
GREWAL; PATERIYA, 2013	Rule-based, hybrid cloud	.	o	o	x	.
BELLAVISTA et al., 2014	Fault-tolerance	.	.	x	.	o	x
KOUSIOURIS et al., 2014	Behavioral-based estimation	.	.	o	.	x

Table 6 – Summary of identified related work classified using the consolidated taxonomy (cont. VIII)

MK = Makespan/Time; DL = Deadline; CT = Cost; DT = Data Intensive/Transfers;
 DY = Dynamic (re)placement; RL = Reliability; SC = Security; EN = Energy;
 HM = Hybrid/Multicloud; WL = Workload/Workflow;
 . = Not addressed; x = Fully addressed; o = Partially addressed
 * = Selected for further analysis

<i>Work</i>	<i>Highlights</i>	<i>MK</i>	<i>DL</i>	<i>CT</i>	<i>DT</i>	<i>DY</i>	<i>RL</i>	<i>SC</i>	<i>EN</i>	<i>HM</i>	<i>WL</i>
HWANG; KIM, 2012	Cost minimization, deadline	.	x	x
MAO; LI; HUMPHREY, 2010	Deadline, budget, auto-scaling	.	x	x	.	o
GAO et al., 2013	Energy, deadline	.	x	o	x	.	.
GREKIOTI; SHAKHLEVICH, 2013	Bag of tasks, time and cost	x	.	x
DASTJERDI; BUYYA, 2012	Negotiation / bargaining	.	x	x
ZAMAN; GROSU, 2011	Auction-based, cloud-provider viewpoint
ROSENBERG et al., 2009	QoS-aware, cost and execution time	x	.	x
SIMAO; VEIGA, 2013	SLA-based cost model; power	o	.	x	o	.	.
GARG et al., 2011	Heterogeneous workloads	.	x
YOO; KIM, 2013	Cost management	.	.	o
KERTESZ et al., 2011	SLA management, improve resource utilization	.	o	o	o
SIMARRO et al., 2011	Multi-cloud, cost optimization	.	.	x	x	.
FRINCU; CRACIUN, 2011	Multi-objective, cost constraints	.	.	x	x	.
* MAHESHWARI et al., 2016	Multi-cloud, enhanced workflow model,	o	x	x	x	x	x

Table 6 consolidates the information of 113 works related to resource management and task scheduling, with the majority focusing on cloud computing and a few works on distributed systems, such as (IVERSON, 1999) and (SIH; LEE, 1993). The majority of the works focus on aspects related to cost and time, such as makespan deadline-based solutions. Among them, makespan is addressed by 44 works (39%), deadlines are addressed by 31 works (27%), and cost is addressed by 43 works (38%). In contrast, none of solutions address security aspects related to confidentiality, such as safe zones to execute code and to store sensitive data.

Regarding support for workflows and workloads, 64 works (57%) provide some level of support to execute workflows using the resource management solution proposed. However, when combined to aspects related to dynamic placement and replacement of resources and tasks, only 19 (17%) provide support for both aspects (dynamic execution of workflows). Combining workflow support to data-intensive workflows leads to only 8 works (7%). Finally, combining workflow support to hybrid and multicloud scenarios, only 2 works (2%) address both aspects. None of the works combine workflow support, data-intensive loads, hybrid and multicloud scenarios, dynamic scheduling and rescheduling, and reliability aspects.

Data-intensive loads are explicitly supported by only 9 works (8%). Hybrid and multicloud scenarios are supported by 7 works (6%). This analysis reveals that while there are works addressing these aspects in separate, none provide explicit support for all aspects regarded as challenges addressed by the proposed solution, MPSF.

Further analysis of the selected work (marked with asterisk in the table) is presented in **Appendix A**. This further analysis was essential to allow the identification of strengths of each work, as well as points to improve and to address in MPSF.

2.3 Gaps and Challenges

This section analyzes the gaps and challenges identified throughout the investigation of surveys and related work on cloud resource management.

2.3.1 Data-Intensive Workflows

Regarding data-intensive workflows, (PANDEY et al., 2010) states that they represent a special class of workflows where the size and/or quantity of data is large. As a direct result, transfer costs are significantly higher and more prominent. While the authors do address data transfers in their resource model, several aspects of data access are not acknowledged. For instance, accesses to the same resource leads to a communication cost of zero. Transfer costs are calculated based on average bandwidth between the nodes, without regards to I/O contention, multiples accesses to the same resource, containers and VMs co-located in the same node sharing network and I/O resources, among other factors. This is also observed in other works such as (LIN; LU, 2011), (RODRIGUEZ; BUYYA, 2014), (FARD et al., 2012), and (BITTENCOURT; MADEIRA, 2011). Other models consider transfers as part of computation time, such as (MALAWSKI et al., 2012). This is depicted as a fundamental challenge by (WU; WU; TAN, 2015), which states that “in most studies, data transfer between tasks is not directly considered, data uploading and downloading are assumed as part of task execution”. (WU; WU; TAN, 2015) complement by stating that this may not be the case for modern applications and workflows – in fact, data movement activities might dominate both execution time and cost. For the authors it is essential to design the data placement strategies for resource provisioning decision-making. Moreover, employing VMs deployed in different regions intensifies the data transfer costs, leading to an even more complicated issue. This is correlated to having more

complex cloud environments in terms of resource distribution, such as hybrid and multicloud scenarios.

2.3.2 Hybrid and Multicloud Scenarios

Regarding hybrid and multicloud scenarios, (WU; WU; TAN, 2015) states that it is necessary hybrid environments, heterogeneous resources, and multicloud environments. (SINGH; CHANA, 2016) also highlights the importance of hybrid and multicloud scenarios for future deployments of large-scale cloud environments and reach performance comparable to large-scale scientific clusters. On the other hand, most of the scheduling solution still do not address hybrid clouds nor multiclouds. The few ones that do implement mechanisms that use the public part of a hybrid cloud to lease additional resources if necessary – the hybrid component of the setup is treated as a supporting element, not as protagonist. For example, (BITTENCOURT; MADEIRA, 2011) and (VECCHIOLA et al., 2012) propose solutions that only allocate resources from the hybrid cloud (the public part of it) if the private part is not able to handle the workflow execution. Multicloud support is even more scarce or not explicit. Several of the proposed solutions could be adopted or adapted to multicloud environments, but there still is a lack of experimental results to match the predicted importance of such large-scale setups.

The motivation for multicloud environments vary from having more raw performance to match other large-scale deployments to having more options in terms of available services. (SIMARRO et al., 2011), for instance, states that resource placements across several cloud offers are useful to obtain resources at the best cost ratio. The same approach is adopted by (FRINCU; CRACIUN, 2011) and (SENTURK et al., 2016). Regarding the execution of large-

scale applications on similar scale systems, (MAHESHWARI et al., 2016) suggest a multi-site workflow scheduling technique to enhance the range of available resources to execute workflows. While their approach does consider data transfers and the costs of sending data over expensive (slower) links that connect different geographically distributed sites, their approach does not consider 1) performance fluctuations during execution of the workflow, which would suggest the implementation of rescheduling and rebalancing mechanisms; 2) reliability mechanisms to cope with performance fluctuations due to failures; and 3) the influence of contention in the general I/O operations, such as sequential accesses to the same data inputs.

2.3.3 Rescheduling and Performance Fluctuations

Performance fluctuations caused by multi-tenant resource sharing is one of the major components that must be included in the definition of uncertainties associated to scheduling operations (WU; WU; TAN, 2015). The authors complement: “The most important problem when implementing algorithms in real environment is the uncertainties of task execution and data transmission time”. Moreover, most works assume that a workflow has a definite DAG structure while actual workflows have loops and conditional branches. For instance, the execution control in several scientific workflows is based on conditions that are calculated every iteration, meaning that branches are essential to determine whether the pipelines must be stopped or not. In this sense, rescheduling techniques are usually adopted to correct potential deviations from an original guess of the performance of a workflow on a system (LEE; SUBRATA; ZUMAYA, 2009, WU; WU; TAN, 2015).

2.4 Problem Definition

The problems derived from the gaps and challenges are:

- Lack of mechanisms to address **data-intensive** workflows, especially considering that future trends point to the direction of I/O workflows with intensive data movement and with reliability-related mechanisms highly dependent on I/O as well.
- Lack of mechanisms to address the particularities of large-scale cloud setups with more complex environments in terms of resource heterogeneity and distribution, such as **hybrid and multicloud scenarios**, which are expected to be the main drivers for large-scale utilization of cloud – scientific workflows being one important instance.
- Lack of mechanisms to address the **fluctuations** in workflow progress due to performance variation and reliability, both phenomena that can be partially or even fully addressed by implementing controlled rescheduling policies.

These points summarize the problems to be addressed by the proposal presented in this work. By designing an integrated solution that addresses these points the objective is to provide one step towards the definition of an actual infrastructure for supporting these complex setups, apart from all limitations that are inherent to the design.

2.5 Chapter Considerations

This chapter provided an investigation of existing works in cloud resource management. The investigation started in Section 2.1 by providing several definitions and associated concepts on the subject, covering the rationale presented by several authors from the academia. Three

main works were selected as they provided a clear definition of distinct steps regarding cloud resource management. Among these works the common point is the association of management components to each phase of the resource lifecycle, such as resource discovery, allocation, scheduling, and monitoring. Moreover, the ultimate objective in all cases is to enable task execution while optimizing infrastructural efficiency.

The next step in this investigation was to identify relevant surveys covering aspects of cloud resource management. Section 2.2 presented the results of this analysis, leading to the identification of several surveys and over 100 works on cloud resource management, as summarized by Table 6. Each survey was analyzed to identify the works mentioned and also the taxonomies or classification methodologies used. These taxonomies were consolidated into a set of characteristics and properties later used to classify the works presented in Table 6.

The next step of this investigation was the identification of gaps and challenges obtained during the research. Section 2.3 presented the gaps and challenges identified in the surveys and related work investigated in its previous sections. The issues were organized in challenges related to data-intensive workflows, including lack of proper modeling of transfers, or modeling of transfers as part of computation; hybrid and multicloud scenarios, comprising large-scale deployments and more complex setups in terms of resource distribution; rescheduling and performance fluctuations, essentially addressing the lack of mechanisms to adequately cope with the inherent performance fluctuation of large scale cloud deployments, and the effects of multi-tenancy and resource sharing.

All these gaps and challenges were finally used in Section 2.4 to define the problems tackled by this work. This comprises the definition of mechanisms to address data-intensive workflows, hybrid and multicloud scenarios, and performance fluctuations.

3 MPSF

The Multiphase Proactive Scheduling Framework (MPSF) is a resource management and scheduling framework for handling the execution of workflows in distributed cloud environments (GONZALEZ; MIERS; CARVALHO, 2016). The framework defines multiple scheduling phases that proactively assess the cloud system and resources to dynamically optimize resource distribution among active workflows. The scheduling process is based on a high-level characterization of resources and workflows. Computational and transfer costs are described as a function of the input data to be processed by each worker node. The workflow is described as a composition of phases with different costs, considering both the time to process the data as well as the time to send this data and retrieve the results.

This chapter is organized as follows. Section 3.1 presents the solution requirements based on the problem definition presented in 2.4. Section 3.2 presents the overview of the solution addressing the requirements presented in Section 3.1. Section 3.3 presents the performance models used for scheduling and rescheduling purposes. Section 3.4 presents how the requirements were covered. Section 3.5 concludes this chapter. The other aspects that complement the framework are presented in the appendices, as specified in Section 1.4.

3.1 Solution Requirements

This section revisits the problems identified in Section 2.4, then it defines the solution requirements and maps these requirements to the problems.

3.1.1 Problems Revisited

As presented in Section 2.4, the problems (P) to be addressed are:

- **P1** *Data-Intensive Workflows*
- **P2** *Hybrid and Multicloud Scenarios*
- **P3** *Rescheduling and Performance Fluctuations*

These problems are summarized in Table 7.

Table 7 – Summary of Problems (P) addressed in this work.

<i>#</i>	<i>Problem</i>	<i>Description</i>
P1	Data-Intensive Workflows	Lack of mechanisms to address workflows with intensive data movement and with reliability mechanisms that highly rely on I/O performance.
P2	Hybrid and Multicloud Scenarios	Lack of mechanisms to address properties of large-scale complex cloud setups, such as data transfer costs and resource heterogeneity.
P3	Rescheduling and Performance Fluctuations	Lack of mechanisms to address performance fluctuations during workflow execution.

3.1.2 Solution Requirements

Based on these problems and properties of a cloud resource management solution, the solution requirements (SR) from a high-level perspective are:

- **SR1** *Workflow model*: MPSF must define a workflow model able to describe the properties of each workflow phase as well as the properties of the links between each phase. This includes resource requirements and associated costs both to the phases and to the links connecting each phase.
- **SR2** *Resource model*: MPSF must define a resource model able to describe the resources that compose a cloud environment and the attributes that allow the connection of multiple environments, either constituting a hybrid setup or a multcloud setup.
- **SR3** *Performance model*: MPSF must define a performance model that captures the characteristics of workflow execution on cloud environments, including the particularities of data distribution over multiple nodes considering I/O contention effects and the properties of hybrid and multcloud scenarios also when distributing data and results.
- **SR4** *Framework*: MPSF must define a framework comprising scheduling phases that address the complete resource lifecycle, triggers that determine when scheduling operations must be executed, thresholds that control when these operations are executed and when decisions must be taken, auxiliary functionalities that provide tools for actually executing a scheduling decision, and operational functions that implement the scheduling and other resource management features.

- **SR5 Architecture:** MPSF must define a software architecture that enables the implementation of the framework and the models proposed. Moreover, it must be possible to integrate this architecture to existing computing frameworks for distributed processing.

A summary of these requirements is presented in Table 8.

Table 8 – Summary of Solution Requirements (SR)

<i>#</i>	<i>Solution Requirement</i>	<i>Description</i>	<i>Related Problems</i>
SR1	Workflow model	Describe properties of workflow phases and links	P1, P3
SR2	Resource model	Describe resources considering multiple cloud environments	P1, P2, P3
SR3	Performance model	Capture the properties of data-intensive workflows	P1, P2, P3
SR4	Framework	Define a framework addressing the full resource lifecycle and required controls	P1, P2, P3
SR5	Architecture	Define an architecture that integrates all framework components and enable the implementation of the solution	P1, P2, P3

3.1.3 Mapping Solution Requirements to Problems

The solution requirements (SR) defined in Section 3.1 are mapped to the problems (P) presented in Section 3.1.1 as presented in Table 9.

Table 9 – Mapping between problems (P) and solution requirements (SR)

<i>P</i>	<i>SR</i>	<i>Mapping</i>
P1	SR1	The workflow model must provide elements to describe the sources of input data for the workflow, as well as how the partial and final results are moved around the nodes. This promotes adequate resource distribution considering I/O contention effects.
	SR2	The resource model must provide the description of inter-node and intra-node costs of sending data; this information is also fundamental to calculate data distribution costs based on workflow properties.
	SR3	The performance model must adequately consider the cost of distributing data to the nodes, accounting for the effects of sending data in sequence to a series of nodes and containers which might share the same node.
	SR4	The framework must define scheduling phases and thresholds not only according to computational requirements of workflow phases, but also based on the data movement patterns and requirements. For instance, thresholds must be defined in order to avoid rescheduling when it is beneficial from a computational point of view but inefficient from a data and I/O standpoint.
	SR5	The architecture must define components that implement the controls based on data movement to address the requirements of data-intensive workflows.
P2	SR2	The resource model must define the data transfer costs of moving information between two distinct sites in a hybrid or multicloud environment.
	SR3	The performance model must consider data distribution also when resources are separated by costly links, such as long-distance networks, typically observed in hybrid and multicloud scenarios.
	SR4	The framework must define mechanisms specific for hybrid and multicloud setups, such as integrated metrics and thresholds considering the whole setup.
	SR5	The architecture must define components to address implementation-related issues comprising heterogeneous resources and large-scale management.
P3	SR1	The workflow model must provide the costs of each phase and the transitions between phases to inform the framework to calculate the cost of rescheduling tasks.
	SR2	The resource model must provide the description of resources for each cloud environment in a hybrid/multicloud setup, also considering the link capacity that connects each environment.
	SR3	The performance model must provide scheduling results that considers the cost of sending data across different sites in a hybrid/multicloud setup.
	SR4	The framework must provide mechanisms for large-scale management of resources from different sites in a hybrid/multicloud scenario.
	SR5	The architecture must provide components associated to the management framework to implement the controls for hybrid/multicloud scenarios.

3.2 Solution Overview

MPSF is a cloud resource management framework based on four essential elements: the workflow characterization, the user requirements, the description of resources, and the models used to determine the resource allocation. Figure 27 presents an overview of these elements.

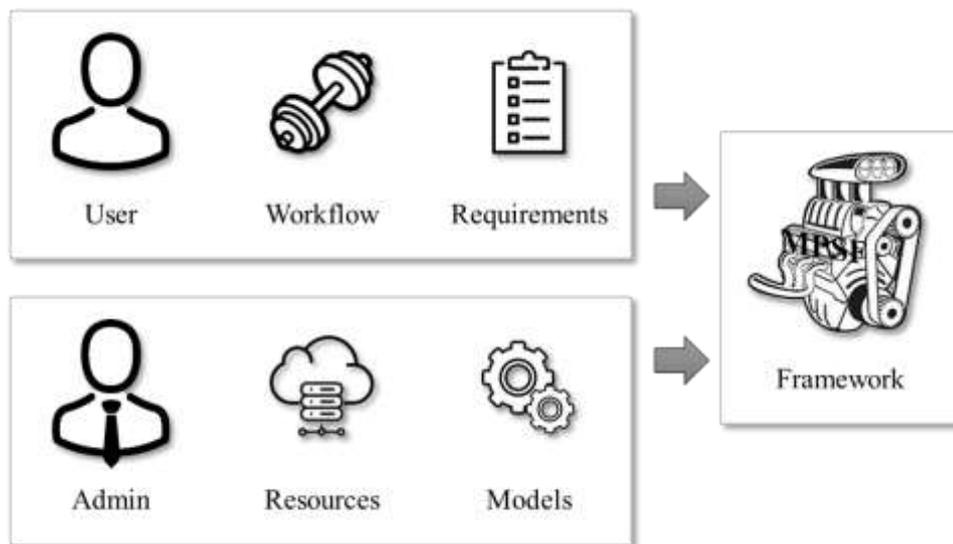


Figure 27 – Overview of MPSF conceptual components

Two actors are identified in Figure 27. The cloud user wants to run a particular workflow on the cloud infrastructure, and the cloud administrator (admin) is responsible for managing this infrastructure as well as for configuring the scheduling framework. The cloud user provides a specification of the workflow and a list of requirements to be addressed and observed during the execution of the workflow. Note that this workflow specification could be the final characterization used by MPSF in its scheduling operations or it could simply be the binaries that compose the workflow and instructions on how to run it. In this case MPSF is able to iteratively execute the workflow in order to determine the characterization of the workflow, defining transfer and computation costs as a function of input data.

MPSF also relies on a description of the available resources in the cloud infrastructure. Resources are organized in a hierarchical structure composed by containers, nodes, groups, and clusters. Finally, MPSF depends on the definition of models that provide the methodology to determine, for instance, the expected run time of a workflow or application based on the input data. These models are also used to calculate the amount of input data that each worker node or container must receive. The distribution is calculated to optimize both transfers and execution, initially focusing on minimizing makespan (total duration of all processes related to the same task). MPSF offers support to any model for these purposes, as long as the final result generated by this model (and its implementation) is the distribution of input data for nodes and containers.

3.2.1 Features

In order to facilitate the translation of solution requirements (SR) presented in Section 3.1 into features (F) that must be supported by the framework to address these requirements, Table 10 provides an overview of the features offered by the framework comprising each of the solution requirements.

Table 10 – Features (F) provided by MPSF

<i>SR</i>	<i>F</i>	<i>Description</i>
1	1	<ul style="list-style-type: none"> • Provide a workflow model structure based on a rich directed acyclic graph (DAG) with more information and properties of each workflow phase and for the connections between these phases.
1	1.1	<ul style="list-style-type: none"> – For workflow phases, each phase must represent a relevant computational operation performed over data. Regarding distributed computing applications and frameworks, this comprises the following minimal set of operations:
1	1.1.1	<ul style="list-style-type: none"> • One operation to represent the entry points of a workflow, as well as the phases wherein data must be reassessed and wherein the scheduling decision could be reevaluated.
1	1.1.2	<ul style="list-style-type: none"> • One operation to represent the distribution of data to multiple processing nodes and containers, with an accurate description of the operations required to divide data and the costs involved.
1	1.1.3	<ul style="list-style-type: none"> • One operation to represent the actual computation phases executed by the workflow, such as transformation and processing functions.
1	1.1.4	<ul style="list-style-type: none"> • One operation to represent the aggregation of preliminary and final results, defining how data can be merged into a single set of values.
1	1.1.5	<ul style="list-style-type: none"> • One operation to represent the phases wherein data sets are produced for preliminary and final result purposes, as well as to indicate points for inspection and general validation.
1	1.2	<ul style="list-style-type: none"> – For connections between phases, the connections must include costs and a description of how much data is transferred based on the inputs received.
1	2	<ul style="list-style-type: none"> • Provide support for branching and multi-path execution. Workflow characterization, scheduling, and execution must consider the multiple paths and branching logic of a workflow.
1	3	<ul style="list-style-type: none"> • Provide support for nesting and workflow composition, effectively implementing the concept of building blocks that can be combined to implement more complex applications.

(cont. I)

<i>SR</i>	<i>F</i>	<i>Description</i>
1	4	<ul style="list-style-type: none"> Describe transfer and computation costs based on the amount of data being handled, or based on information collected by the framework based on the amount of data typically processed.
1	4.1	<ul style="list-style-type: none"> Provide different representation possibilities for representing cost, such as fixed values collected by the framework, based on functions, or using complexity notation.
1	4.2	<ul style="list-style-type: none"> Provide framework automation to collect these costs if the workflow characterization does not provide this information.
1	4.3	<ul style="list-style-type: none"> Even if the information is provided, provide automation in the sense of validating the characterization.
1	4.4	<ul style="list-style-type: none"> Allow the characterization of the cost in terms of data structures being handled and manipulated by the workflow.
1	5	<ul style="list-style-type: none"> Provide transparent integration from workflow description into implementation and execution of the workflow, so that phases and connections are transparently transformed into the execution of applications or data movement across the system.
1	6	<ul style="list-style-type: none"> Extend the concept of data-based cost into other metrics, such as memory and storage footprint; define the primitives so that other metrics can be calculated, such as energy consumption (even if these metrics are not considered to be as part of the scope of this work).
2	1	<ul style="list-style-type: none"> Define a resource hierarchy comprising resource levels in a cloud-local space and also comprising multiple cloud setups.
2	2	<ul style="list-style-type: none"> Define an abstraction to identify nodes and containers (based on some sort of compartmentalization strategy, such as virtualization or operating system level containers) that are allocated to the same task or workflow.
2	3	<ul style="list-style-type: none"> Define an abstract hierarchy including both the primary resource hierarchy (based on physical and virtual resources) and the abstract components that identify a working group.
2	4	<ul style="list-style-type: none"> Provide the data structures to describe the resources and properties related either to performance or to users requirements for the execution of workflows.
3	1	<ul style="list-style-type: none"> Define a performance model that adequately considers the data contention effects for distribution of data among nodes.
3	2	<ul style="list-style-type: none"> Define a performance model that considers the performance fluctuations and provides hints on how to prevent degradation.

(cont. II)

<i>SR</i>	<i>F</i>	<i>Description</i>
3	3	<ul style="list-style-type: none"> Define a performance model that enables rescheduling of resources and tasks based on the performance variations observed during execution of the workflow.
4	1	<ul style="list-style-type: none"> Define a framework with clear phases covering every aspect of resource lifecycle, comprising discovery, preallocation, allocation, reallocation, and deallocation.
4	2	<ul style="list-style-type: none"> Define triggers that control when scheduling and rescheduling decisions must be calculated or recalculated.
4	3	<ul style="list-style-type: none"> Define thresholds that control when scheduling and rescheduling operations must be actually executed.
4	4	<ul style="list-style-type: none"> Define auxiliary functionalities that provide tools for the scheduling and resource management operations, such as to support discovery and allocation.
5	1	<ul style="list-style-type: none"> Provide the architectural components that implement the features for workflows, resources, performance models, and framework mechanisms.
5	2	<ul style="list-style-type: none"> Provide architectural integration points with other computing frameworks for distributed processing.

These are the features to be provided by MPSF. These features include the definition of a workflow model to adequately represent the workflows to be executed in the infrastructure, the creation of a resource hierarchy to support the execution of these workflows, the definition of performance models to support data-intensive workflows and performance fluctuations. To integrate all these components, the list of features include the definition of a framework and an architecture that orchestrates their utilization.

3.3 Performance Model

This section presents the performance model used in MPSF to calculate data and consequently load distribution. Section 3.3.1 presents an introduction to the problem of performance optimization, comparing the typical approach adopted by the existing solutions (based on simple DAGs) versus the approach proposed in this work based on rich characterization of workflows and resources. Section 3.3.2 presents the simple model for data distribution, considering that transfers can be executed in parallel without any degradation due to contention and concurrency. Section 3.3.3 provides an example on the utilization of this method, and Section 3.3.4 provides an experimental verification of the method.

Based on this analysis, Sections 3.3.5 and 3.3.6 provide the definition of an enhanced model that considers sequential transfers and the experimental verification of the model. The model is much more accurate but the observed performance is not much better than the one obtained in the simple model. Then, based on the properties of workflows and workflow phases, Section 3.3.7 proposes the final model that distinguishes shared and chunk data. Shared data is common among multiple iterations of the workflow while chunk data needs to be sent every iteration. The dynamics of these types of data are used to defined sub-models that optimize data distribution among nodes.

Finally, Section 3.3.8 presents the probabilistic model based on a Bayesian system of multiple hypotheses and multiple evidences to address the performance fluctuations observed during workflow execution without imposing a drastic cost to the system in terms of tests and general assessment.

3.3.1 Problem Introduction

Load distribution is a classical problem of optimization of one or more variables. In this case the variable to be optimized is time, in particular the total run time of a workflow. The approach typically adopted by workflow scheduling solutions is to represent the workflow as a directed acyclic graph (DAG) with costs associated to the vertices (tasks) and edges (transfers). Compared to MPSF's workflow representation, the DAG does not allow the definition of cycles (e.g., loops), which simplifies the modeling aspect. In MPSF there is the definition of cycles and also splits and merges, which must be considered in the calculations. Finally, the ultimate aspect to be considered in the modeling is the distribution of data, in particular the sequential aspect of it and the derived contention issues.

3.3.2 Simple Model for Data Distribution

The first phase of a workflow typically comprises some mechanism to read the input data that will be processed by the compute elements of the workflow. In some cases this data is generated using some random generator of pertinent values for the execution of the workflow, or for testing it. Fact is, I/O plays a key role in the beginning of the workflow, either if it is file based, network based, or simply in-memory computation.

A straightforward approach to model the data distribution is to assume that all data is consistently and equally distributed to all worker elements (either nodes, containers, or any other element from the same execution space). Figure 28 illustrates the data distribution from node 0 to nodes 1, 2, and 3.

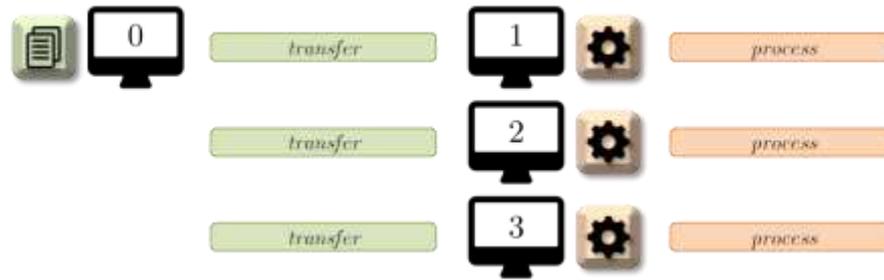


Figure 28 – Illustration of data distribution

Figure 28 shows node 0 distributing data to other three nodes. It is assumed that the data is distributed at the same time to the other nodes. There is an implicit assumption that it is possible to share the communication channel to distribute the data. Considering that the amount of data to be transferred is D , the transfer bandwidth between node 0 and node i is r_i , and the processing bandwidth of node i is p_i , the time of transferring data and then processing this data in each node is:

Equation 1 – Time to transfer and process data according to simple model

$$t_i = \frac{d_i}{r_i} + \frac{d_i}{p_i}$$

Where d_i is the amount of data sent to node i .

The optimization step is summarized by minimizing the total duration (makespan) of the tasks. This is accomplished by **equaling** the t_i times:

Equation 2 – Optimization rule based on equaling times

$$t_i = t_j, \forall i, j \in N$$

Where N is set of worker elements that compose the execution space being used to execute the task. Note that this expression leads to the definition of several equivalent expressions. To build a determined system N independent equations are required. These equations can be built using the following rule:

Equation 3 – Expressions used to build the equations for determined system

$$t_i = t_{i+1}, \forall i < n, n = |N|$$

Where $n = |N|$ (number of elements of N). This leads to expressions such as:

Equation 4 – Equations derived from equaling the time expressions

$$\begin{aligned} t_1 &= t_2 \\ t_2 &= t_3 \\ &\dots \\ t_{n-1} &= t_n \end{aligned}$$

Which leads to a total of $n - 1$ expressions. The final expression to make this system determined is based on the sum of all data chunks, which must correspond to the total data (which is known):

Equation 5 – Final expression used to build the determined system

$$\sum_{i=1}^n d_i = D$$

For the example of Figure 28 the following system would be conceived:

Equation 6 – Equation system using equaled times and sum of data

$$\begin{array}{rcl}
 t_1 = t_2 & \rightarrow & \frac{d_1}{r_1} + \frac{d_1}{p_1} = \frac{d_2}{r_2} + \frac{d_2}{p_2} \quad \text{(I)} \\
 t_2 = t_3 & \rightarrow & \frac{d_2}{r_2} + \frac{d_2}{p_2} = \frac{d_3}{r_3} + \frac{d_3}{p_3} \quad \text{(II)} \\
 & & d_1 + d_2 + d_3 = D \quad \text{(III)}
 \end{array}$$

A determined system of equations with three variables: d_1 , d_2 , and d_3 . This system can be organized to be solved via Gauss-Seidel method in the form $Ax = b$:

Equation 7 – Matrix representation of system of equations

$$\begin{array}{c}
 \left| \begin{array}{ccc}
 \frac{1}{p_1} + \frac{1}{r_1} & -\left(\frac{1}{p_2} + \frac{1}{r_2}\right) & 0 \\
 0 & \frac{1}{p_2} + \frac{1}{r_2} & -\left(\frac{1}{p_3} + \frac{1}{r_3}\right) \\
 1 & 1 & 1
 \end{array} \right| \cdot \left| \begin{array}{c}
 d_1 \\
 d_2 \\
 d_3
 \end{array} \right| = \left| \begin{array}{c}
 0 \\
 0 \\
 D
 \end{array} \right| \\
 \Downarrow & & \Downarrow \\
 A & \cdot & x = b
 \end{array}$$

Simplifying the numerical factors in matrix A:

Equation 8 – Simplification applied to numerical factors

$$\begin{array}{c}
 \frac{1}{p_i} + \frac{1}{r_i} = \alpha_i \\
 \left| \begin{array}{ccc}
 \alpha_1 & -\alpha_2 & 0 \\
 0 & \alpha_2 & -\alpha_3 \\
 1 & 1 & 1
 \end{array} \right| \cdot \left| \begin{array}{c}
 d_1 \\
 d_2 \\
 d_3
 \end{array} \right| = \left| \begin{array}{c}
 0 \\
 0 \\
 D
 \end{array} \right|
 \end{array}$$

Solving the system for d_3 leads to the following expressions:

Equation 9 – Solving system

$$\alpha_1 d_1 - \alpha_2 d_2 = 0 \rightarrow d_1 = \frac{\alpha_2}{\alpha_1} d_2 \quad (\text{I})$$

$$\alpha_2 d_2 - \alpha_3 d_3 = 0 \rightarrow d_2 = \frac{\alpha_3}{\alpha_2} d_3 \quad (\text{II})$$

- Substituting (II) in (I):

$$d_1 = \frac{\alpha_2}{\alpha_1} \cdot \frac{\alpha_3}{\alpha_2} d_3 = \frac{\alpha_3}{\alpha_1} d_3 \quad (\text{III})$$

- Substituting (II) and (III) in the sum of D :

$$d_1 + d_2 + d_3 = D \Rightarrow$$

$$\frac{\alpha_3}{\alpha_1} d_3 + \frac{\alpha_3}{\alpha_2} d_3 + d_3 = D \quad (\text{IV})$$

- Assuming $d_3 = \frac{\alpha_3}{\alpha_3} d_3 \rightarrow$ apply to (IV)

$$\frac{\alpha_3}{\alpha_1} d_3 + \frac{\alpha_3}{\alpha_2} d_3 + \frac{\alpha_3}{\alpha_3} d_3 = D \quad (\text{V})$$

- Let $\mu_i = \frac{1}{\alpha_i} \rightarrow$ apply to (V)

$$\frac{\mu_1}{\mu_3} d_3 + \frac{\mu_2}{\mu_3} d_3 + \frac{\mu_3}{\mu_3} d_3 = D \Rightarrow$$

$$d_3 = \frac{D}{\frac{\mu_1}{\mu_3} + \frac{\mu_2}{\mu_3} + \frac{\mu_3}{\mu_3}} \Rightarrow$$

$$d_3 = \frac{\mu_3}{\mu_1 + \mu_2 + \mu_3} \cdot D \quad (\text{VI})$$

Expression VI in Equation 9 reveals that the amount of data to be sent to node 3, d_3 , is proportional to the weighted average of the factor μ_3 compared to all factors μ_i calculated for

each node i . The factor μ_i is the inverse of α_i , which is the sum of the inverses of the transfer rate and processing rate of node i .

Equation 10 – Performance factor

$$\alpha_i = \frac{1}{r_i} + \frac{1}{p_i}$$

$$\mu_i = \frac{1}{\alpha_i} = \frac{1}{\frac{1}{r_i} + \frac{1}{p_i}} = \frac{r_i \cdot p_i}{r_i + p_i}$$

The final expressions to calculate the data distribution using the simple model are:

Equation 11 – Expressions to calculate data distribution for simple model

$$\mu_i = \frac{r_i \cdot p_i}{r_i + p_i} \quad \text{performance factor}$$

$$d_i = \frac{\mu_i}{\sum_{j=1}^n \mu_j} \cdot D \quad \text{data chunk}$$

These expressions are used first to calculate the performance factor of each node, which depends on the transfer and processing rates. Then the second expression is used to determine the amount of data that each node must receive based on a weighted average calculated using the performance factors.

3.3.3 Exemplar Utilization of Simple Model

A theoretical example of utilization of the model considers a simple problem of distributing data to three nodes for further processing. The parameters of the problem are:

Table 11 – Parameters for theoretical problem

	Node	<i>r</i>	<i>p</i>
<i>D</i> = 100	1	1	1
	2	1	3
	3	2	2

The value of *D* is expressed as size (e.g. *bytes*) while the performance values *r* and *p* are expressed as size per time (e.g., *bytes per second*). Using the expressions from Equation 11 the performance factors are:

Equation 12 – Performance factors calculated for exemplar problem

$$\mu_1 = \frac{r_1 \cdot p_1}{r_1 + p_1} = \frac{1 \cdot 1}{1 + 1} = \frac{1}{2}$$

$$\mu_2 = \frac{r_2 \cdot p_2}{r_2 + p_2} = \frac{1 \cdot 3}{1 + 3} = \frac{3}{4}$$

$$\mu_3 = \frac{r_3 \cdot p_3}{r_3 + p_3} = \frac{2 \cdot 2}{2 + 2} = \frac{4}{4}$$

Then, using the second expression of Equation 11 leads to the amount of data to be distributed per node:

Equation 13 – Solving system for example

$$\sum_{j=1}^n \mu_j = \frac{2}{4} + \frac{3}{4} + \frac{4}{4} = \frac{9}{4}$$

$$d_1 = \frac{\mu_1}{\sum_{j=1}^n \mu_j} \cdot D = \frac{2/4}{9/4} \cdot 100 = 22.2$$

$$d_2 = \frac{\mu_2}{\sum_{j=1}^n \mu_j} \cdot D = \frac{3/4}{9/4} \cdot 100 = 33.3$$

$$d_3 = \frac{\mu_3}{\sum_{j=1}^n \mu_j} \cdot D = \frac{4/4}{9/4} \cdot 100 = 44.4$$

Verifying the results:

Equation 14 – Verification

$$t_1 = \frac{22.2}{1} + \frac{22.2}{1} = 44.4 \text{ s}$$

$$t_2 = \frac{33.3}{1} + \frac{33.3}{3} = 44.4 \text{ s}$$

$$t_3 = \frac{44.4}{2} + \frac{44.4}{2} = 44.4 \text{ s}$$

Duration of transfer plus computation for the nodes match each other values. Based on the initial optimization assumption, the distribution is correct and minimizes makespan for this sequence of tasks.

3.3.4 Experimental Verification of Simple Model

The next step is to experimentally verify the validity model and its applicability. Two sets of nodes were used for this experiment, one set based on the Intel Xeon X5687 and another based on the Intel Xeon E3-1230. Each set of nodes is located in two layers of switches, as illustrated by Figure 29.

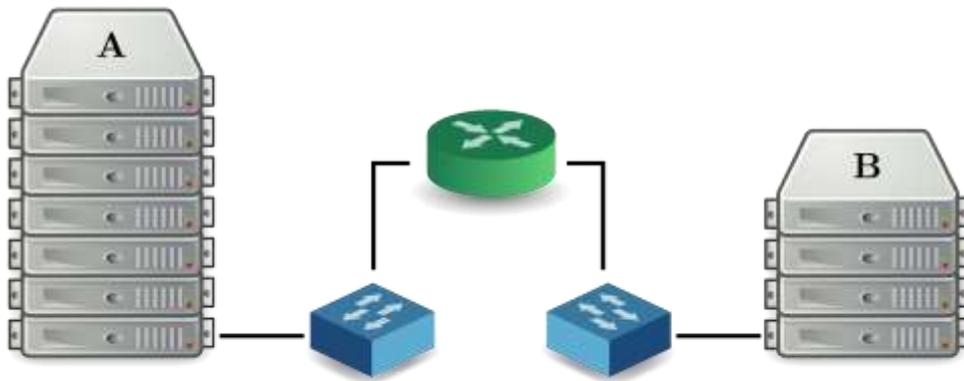


Figure 29 – Experimental setup to validate model

The input file is located in one node from cluster A. Two machines from each cluster are allocated as worker nodes for this experiment. The test application implements a moving average algorithm. Transfer bandwidth from input to a node from cluster A is higher than to cluster B. The values for processing rates also differ, as observed in Table 12. Cluster A is based on the Intel Xeon X5687 while Cluster B is based on the Intel Xeon E3-1230.

Table 12 – Machine parameters for experimental validation

	Node	r (MB/s)	p (MB/s)
$D = 100$	A_1	48.1	81.0
	A_2	48.1	81.0
	B_1	37.3	94.3
	B_2	37.3	94.3

The transfer rate r_i is also considering the time to read the input in a shared manner (dividing read bandwidth by four) and the time to write in the destination. All rates were experimentally measured by executing 100 tests individually for each node and each operation (either transfer or processing). In other words, to measure r_i and p_i for each node i 100 complete runs were executed for the node by itself. Then the time spent on each operation (transfer and computation) was measured and the final rates calculated.

Equation 15 – Solving system for example

$$\sum_{j=1}^n \mu_j = \sum_{j=1}^n \frac{r_j \cdot p_j}{r_j + p_j} = 2 \cdot \frac{48.1 \cdot 81.0}{48.1 + 81.0} + 2 \cdot \frac{37.3 \cdot 94.3}{37.3 + 94.3} = 113.8$$

$$d_A = \frac{\mu_A}{\sum_{j=1}^n \mu_j} \cdot D = \frac{30.2}{113.8} \cdot 100\% = 26.5\%$$

$$d_B = \frac{\mu_B}{\sum_{j=1}^n \mu_j} \cdot D = \frac{26.7}{113.8} \cdot 100 = 23.5\%$$

This means that each node from cluster A should receive 26.5% of the input while each node from cluster B should receive 23.5% of the input.

Verification:

Equation 16 – Verification

$$t_A = \frac{26.5}{48.1} + \frac{26.5}{81.0} = 0.88 \text{ s}$$

$$t_2 = \frac{23.5}{37.3} + \frac{23.5}{94.3} = 0.88 \text{ s}$$

The expected time distribution for these tasks is presented in Figure 30. Note that all tasks finish at the same time. In the figure, t_r is the transfer time and t_p is the processing time for each node. Values were calculated for an input of 1 GB.

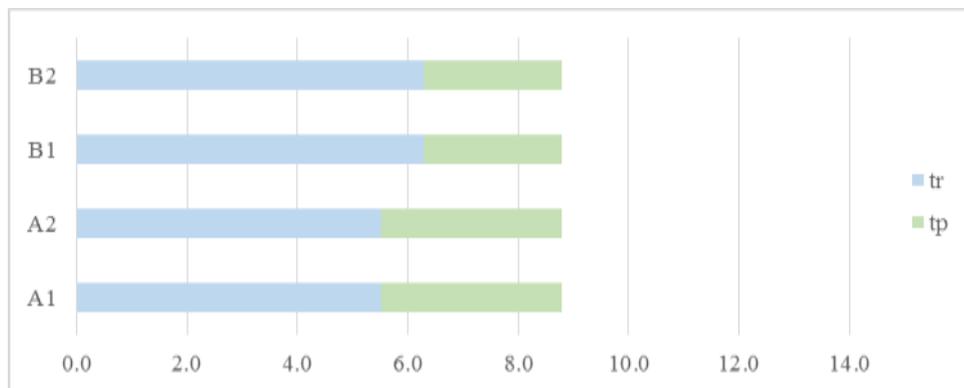


Figure 30 – Expected time distribution for tasks

The actual validation is executing the tasks and verifying how close the proposed scheduling is from reality. The results are presented in Table 13. Experiments were executed using a file of 1 GB (10^9 bytes).

Table 13 – Results obtained for experimental verification (one run)

Node	t_r	t_p
A_1	6.7	3.4
A_2	7.9	3.8
B_1	9.2	2.4
B_2	11.4	2.5

Based on these results, the time distribution is depicted in Figure 31.

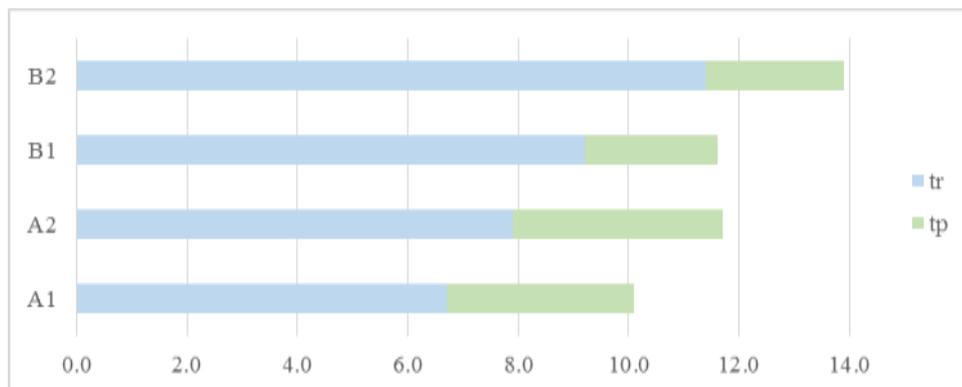


Figure 31 – Observed time distribution

This represents one run. Running more repetitions does stabilize the performance, but at a much slower figure than the predicted by the model. The model predicted a makespan of 8.8 s while the actual execution led to a makespan of 13.9 s, an error margin of 58%. The dashed line shows the initially expected makespan. Table 14 presents the error margin for each value predicted by the model.

Table 14 – Error margin for each predicted value

Node	t_r (model)	t_r (real)	error	t_p (model)	t_p (real)	error
A_1	5.5	6.7	22%	3.3	3.4	4%
A_2	5.5	7.9	43%	3.3	3.8	16%
B_1	6.3	9.2	46%	2.5	2.4	4%
B_2	6.3	11.4	81%	2.5	2.5	0%

The error margins for transfers are particularly higher and increase in the same order that the transfers are initiated. Although all transfers are started at the same time, the ones started before have a slight performance advantage over the ones started after. This shows that the initial assumption that accounting for multiple input transfers by simply dividing available read bandwidth by the number of destinations leads to imprecise results. Multiple concurrent reads in the same node, especially when the same file is involved, leads to contention effects that severely hinder performance.

3.3.5 Enhanced Model for Data Distribution

Using the simple model for data distribution provides an initial estimate of the data distribution, but even for a simple case of four tasks and a simple program the makespan estimation error was almost 60%. The error was mostly due to the imprecision in the prediction of the initial distribution based on the assumption that the read bandwidth can be equally shared among multiple processes reading the same set of inputs without major concurrency and contention issues.

The enhancement made to this model is to substitute this assumption by a simpler one: the initial distribution of inputs, which can be based on one or even on a set of inputs (e.g., files), is first done sequentially to the worker elements. Instead of the data distribution presented in Figure 28, the distribution and subsequent execution would follow the behavior presented in Figure 32.

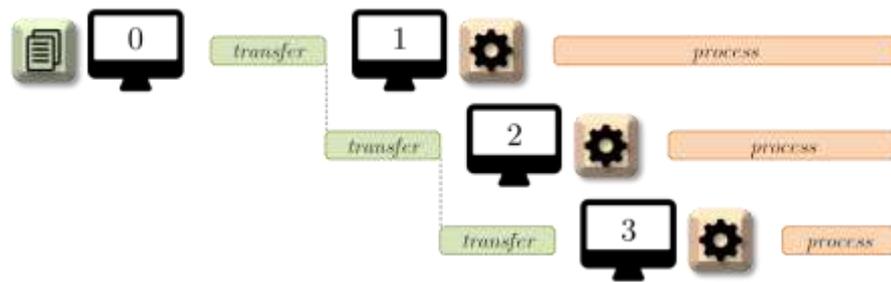


Figure 32 – Data distribution in the enhanced model

Now the Equation 1 has to take in account the previous transfers executed:

Equation 17 – Time to transfer and process data according to enhanced model

$$t_1 = \frac{d_1}{r_1} + \frac{d_1}{p_1}$$

$$t_2 = \frac{d_1}{r_1} + \frac{d_2}{r_2} + \frac{d_2}{p_2}$$

...

$$t_i = \left(\sum_{j=0}^{i-1} \frac{d_j}{r_j} \right) + \frac{d_i}{r_i} + \frac{d_i}{p_i} \Rightarrow$$

$$t_i = \left(\sum_{j=0}^i \frac{d_j}{r_j} \right) + \frac{d_i}{p_i} \quad (\text{I})$$

Essentially expression (I) from Equation 17 shows that the time of transfer plus processing for node i is the sum of all previous transfers plus the transfer to node i plus the processing on node i . The advantage of this model is that there is no concurrency during the data distribution and, in theory, the final performance is similar, considering that in the previous model the bandwidth to read the input files was divided by the number of worker nodes.

Applying the same optimization assumption as presented in Equation 2 and using the sequential equations as presented in Equation 3 leads to the set of equations as described in Equation 4. Adding the equation representing the sum of data chunks (equal to the total amount of data, D) leads to the equation system defined in Equation 18.

Equation 18 – Equation system using equaled times and sum of data and enhanced model

$$\begin{aligned}
 t_1 = t_2 & \rightarrow \cancel{\frac{d_1}{r_1}} + \frac{d_1}{p_1} = \cancel{\frac{d_1}{r_1}} + \frac{d_2}{r_2} + \frac{d_2}{p_2} \Rightarrow \\
 & \frac{d_1}{p_1} = \frac{d_2}{r_2} + \frac{d_2}{p_2} \quad \text{(I)} \\
 t_2 = t_3 & \rightarrow \cancel{\frac{d_1}{r_1}} + \cancel{\frac{d_2}{r_2}} + \frac{d_2}{p_2} = \cancel{\frac{d_1}{r_1}} + \cancel{\frac{d_2}{r_2}} + \frac{d_3}{r_3} + \frac{d_3}{p_3} \Rightarrow \\
 & \frac{d_2}{p_2} = \frac{d_3}{r_3} + \frac{d_3}{p_3} \quad \text{(II)} \\
 & d_1 + d_2 + d_3 = D \quad \text{(III)}
 \end{aligned}$$

The following substitution is applied to the equations:

Equation 19 – Substitution of variables

$$R_i = \frac{1}{r_i} \quad P_i = \frac{1}{p_i}$$

Leading to the following equations:

Equation 20 – Equation system after substitutions

$$P_1 d_1 = R_2 d_2 + P_2 d_2 \quad (\text{I})$$

$$P_2 d_2 = R_3 d_3 + P_3 d_3 \quad (\text{II})$$

$$d_1 + d_2 + d_3 = D \quad (\text{III})$$

Isolating to find d_3 :

Equation 21 – Equation system after substitutions and isolation

$$d_1 = \frac{R_2 + P_2}{P_1} \cdot d_2 \quad (\text{I})$$

$$d_2 = \frac{R_3 + P_3}{P_2} \cdot d_3 \quad (\text{II})$$

$$d_1 + d_2 + d_3 = D \quad (\text{III})$$

- Substituting d_1 and d_2 in (III):

$$\frac{R_2 + P_2}{P_1} \cdot \frac{R_3 + P_3}{P_2} \cdot d_3 + \frac{R_3 + P_3}{P_2} \cdot d_3 + d_3 = D \Rightarrow$$

$$d_3 = \frac{D}{\frac{R_2 + P_2}{P_1} \cdot \frac{R_3 + P_3}{P_2} + \frac{R_3 + P_3}{P_2} + 1}$$

- Substituting $\beta_i = \frac{R_{i+1} + P_{i+1}}{P_i}$: (IV)

$$d_3 = \frac{D}{\beta_1 \cdot \beta_2 + \beta_2 + 1} \quad (\text{V})$$

Combining expressions (V), (I), (II), and (IV):

Equation 22 – Final equations for three variables

$$d_3 = \frac{D}{\beta_1 \cdot \beta_2 + \beta_2 + 1}$$

$$d_2 = \beta_2 \cdot d_3 \qquad \beta_i = \frac{R_{i+1} + P_{i+1}}{P_i} = \frac{\frac{1}{r_{i+1}} + \frac{1}{p_{i+1}}}{\frac{1}{p_i}}$$

$$d_1 = \beta_1 \cdot \beta_2 \cdot d_3$$

The verification of the results will be based on the following parameters:

Table 15 – Parameters for theoretical problem

	Node	<i>r</i>	<i>p</i>
<i>D</i> = 100	1	1	1
	2	2	1
	3	3	1

Calculating the β factors:

Equation 23 – Beta factors

$$\beta_1 = \frac{\frac{1}{r_2} + \frac{1}{p_2}}{\frac{1}{p_1}} = \frac{\frac{1}{2} + \frac{1}{1}}{\frac{1}{1}} = \frac{3/2}{1/1} = \frac{3}{2}$$

$$\beta_2 = \frac{\frac{1}{r_3} + \frac{1}{p_3}}{\frac{1}{p_2}} = \frac{\frac{1}{3} + \frac{1}{1}}{\frac{1}{1}} = \frac{4/3}{1/1} = \frac{4}{3}$$

Then, calculating the d_i values:

Equation 24 – Data chunks

$$d_3 = \frac{D}{\beta_1 \cdot \beta_2 + \beta_2 + 1} = \frac{100}{3/2 \cdot 4/3 + 4/3 + 1} = \frac{100}{2 + 4/3 + 1} = \frac{100}{13/3} = 23.1$$

$$d_2 = \beta_2 \cdot d_3 = \frac{4}{3} \cdot 23.1 = 30.8$$

$$d_1 = \beta_1 \cdot d_2 = \frac{3}{2} \cdot 30.8 = 46.2$$

Verifying the execution times:

Equation 25 – Execution times

$$t_3 = \frac{d_1}{r_1} + \frac{d_2}{r_2} + \frac{d_3}{r_3} + \frac{d_3}{p_3} = \frac{46.2}{1} + \frac{30.8}{2} + \frac{23.1}{3} + \frac{23.1}{1} = 92.4$$

$$t_2 = \frac{d_1}{r_1} + \frac{d_2}{r_2} + \frac{d_2}{p_2} = \frac{46.2}{1} + \frac{30.8}{2} + \frac{30.8}{1} = 92.4$$

$$t_1 = \frac{d_1}{r_1} + \frac{d_1}{p_1} = \frac{46.2}{1} + \frac{46.2}{1} = 92.4$$

The results match, satisfying the optimization condition of equal times. Expanding this system to four variables would lead to the following expressions:

Equation 26 – Final equations for four variables

$$d_4 = \frac{D}{\beta_1 \cdot \beta_2 \cdot \beta_3 + \beta_2 \cdot \beta_3 + \beta_3 + 1}$$

$$d_3 = \beta_3 \cdot d_4$$

$$d_2 = \beta_2 \cdot d_3 = \beta_2 \cdot \beta_3 \cdot d_4$$

$$d_1 = \beta_1 \cdot d_2 = \beta_1 \cdot \beta_2 \cdot \beta_3 \cdot d_4$$

These equations can be expressed using the generic form presented in Equation 27.

Equation 27 – General form with intermediary variables

$$d_i = \frac{\rho_i}{\pi} D$$

Where:

$$\rho_i = \prod_{j=i}^{n-1} \beta_j$$

$$\pi = \sum_{p=1}^{n-1} \prod_{q=p}^{n-1} \beta_q = \sum_{p=1}^{n-1} \rho_p$$

$$\beta_j = \frac{\frac{1}{r_{j+1}} + \frac{1}{p_{j+1}}}{\frac{1}{p_j}}$$

Or in its full form without intermediary variables:

Equation 28 – Full form of enhanced model

$$d_i = \frac{\prod_{j=i}^{n-1} \frac{1}{r_{j+1}} + \frac{1}{p_{j+1}}}{\sum_{p=1}^{n-1} \prod_{q=p}^{n-1} \frac{1}{r_{q+1}} + \frac{1}{p_{q+1}}} D$$

These expressions are based on the fact that to calculate d_i the denominator is fixed and based on the values of β and the numerator depends on the index i . For the last index, d_n , the numerator is 1, while for the first index the numerator is the product of all values of β . The simplified expression from Equation 27 resembles a weighted average in the sense that the $1 + \pi$ factor is the global weight while the ρ_i factor is the local weight.

3.3.6 Experimental Verification of Enhanced Model

The experimental verification for the enhanced model is based on the same experimental platform presented in Figure 29. Table 16 presents the machine parameters adjusting the values of r considering that now the read bandwidth of the input node is not shared by four processes.

Table 16 – Machine parameters for experimental validation

	Node	Node	r (MB/s)	p (MB/s)
$D = 100$ MB	1	A_1	86.2	81.0
	2	A_2	86.2	81.0
	3	B_1	56.8	94.3
	4	B_2	56.8	94.3

Calculating the betas:

$$\beta_1 = \frac{\frac{1}{r_2} + \frac{1}{p_2}}{\frac{1}{p_1}} = \frac{\frac{1}{86.2} + \frac{1}{81.0}}{\frac{1}{81.0}} = \frac{0.0116 + 0.0123}{0.0123} = 1.9431$$

$$\beta_2 = \frac{\frac{1}{r_3} + \frac{1}{p_3}}{\frac{1}{p_2}} = \frac{\frac{1}{56.8} + \frac{1}{94.3}}{\frac{1}{81.0}} = \frac{0.0176 + 0.0106}{0.0123} = 2.2927$$

$$\beta_3 = \frac{\frac{1}{r_4} + \frac{1}{p_4}}{\frac{1}{p_3}} = \frac{\frac{1}{56.8} + \frac{1}{94.3}}{\frac{1}{94.3}} = \frac{0.0176 + 0.0106}{0.0106} = 2.6604$$

Calculating the rhos:

$$\rho_1 = \prod_{j=1}^3 \beta_j = \beta_1 \beta_2 \beta_3 = 1.9431 \cdot 2.2927 \cdot 2.6604 = 11.8519$$

$$\rho_2 = \prod_{j=2}^3 \beta_j = \beta_2 \beta_3 = 2.2927 \cdot 2.6604 = 6.0995$$

$$\rho_3 = \prod_{j=3}^3 \beta_j = \beta_3 = 2.6604$$

$$\rho_4 = \prod_{j=4}^3 \beta_j = 1.000$$

Calculating the denominator:

$$\pi = \sum_{p=1}^{n-1} \prod_{q=p}^{n-1} \beta_q = \rho_1 + \rho_2 + \rho_3 + \rho_4$$

$$\pi = 11.8519 + 6.0995 + 2.6604 + 1.000 =$$

$$\pi = 21.6118$$

Calculating now the chunks:

$$d_1 = \frac{\rho_1}{\pi} D = \frac{11.8519}{21.6118} 100 = 54.84$$

$$d_2 = \frac{\rho_2}{\pi} D = \frac{6.0995}{21.6118} 100 = 28.22$$

$$d_3 = \frac{\rho_3}{\pi} D = \frac{2.6604}{21.6118} 100 = 12.31$$

$$d_4 = \frac{\rho_4}{\pi} D = \frac{1.000}{21.6118} 100 = 4.63$$

Verifying the distribution (now for 100 MB):

$$t_1 = \frac{d_1}{r_1} + \frac{d_1}{p_1} = \frac{54.84}{86.2} + \frac{54.84}{81.0} =$$

$$0.6362 + 0.6770 = 1.3 \text{ s}$$

$$t_2 = \frac{d_1}{r_1} + \frac{d_2}{r_2} + \frac{d_2}{p_2} = \frac{54.84}{86.2} + \frac{28.22}{86.2} + \frac{28.22}{81.0} =$$

$$0.6361 + 0.2926 + 0.3484 = 1.3 \text{ s}$$

$$t_3 = \frac{d_1}{r_1} + \frac{d_2}{r_2} + \frac{d_3}{r_3} + \frac{d_3}{p_3} = \frac{54.84}{86.2} + \frac{28.22}{86.2} + \frac{12.31}{56.8} + \frac{12.31}{94.3} =$$

$$0.6361 + 0.2926 + 0.2167 + 0.1305 = 1.3 \text{ s}$$

$$t_1 = \frac{d_1}{r_1} + \frac{d_2}{r_2} + \frac{d_3}{r_3} + \frac{d_4}{r_4} + \frac{d_4}{p_4} = \frac{54.84}{86.2} + \frac{28.22}{86.2} + \frac{12.31}{56.8} + \frac{4.63}{56.8} + \frac{4.63}{94.3} =$$

$$0.6361 + 0.2926 + 0.2167 + 0.0815 + 0.0491 = 1.3 \text{ s}$$

The expected time distribution is presented in Figure 33 (adjusted to 1 GB).

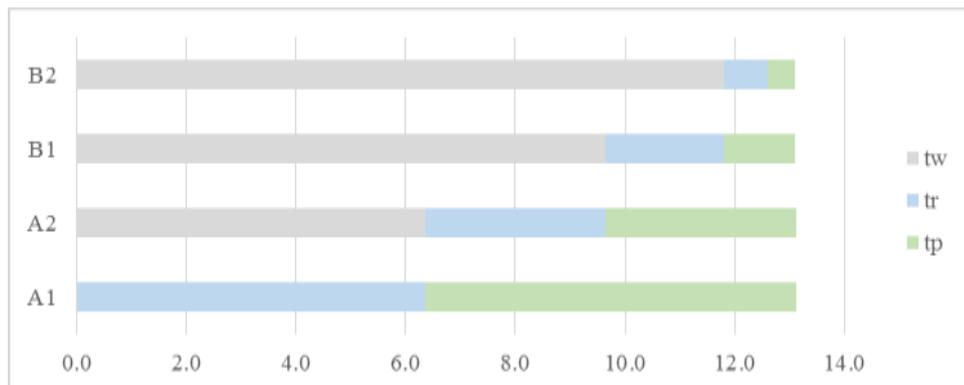


Figure 33 – Expected time distribution for enhanced model

The results of the actual experiments are presented in Table 17.

Table 17 – Results obtained for experimental verification (one run)

Node	t_r	t_p
A_1	6.5	7.0
A_2	3.4	3.4
B_1	2.2	1.4
B_2	1.0	0.5

These results lead to the time distribution presented in Figure 34.

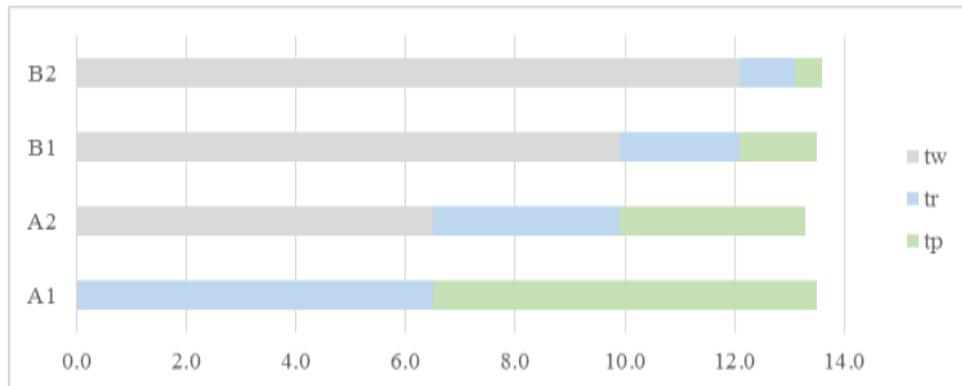


Figure 34 – Time distribution observed for experimental evaluation (t_w is waiting time)

Different from the results for the simple model, the time distribution is much closer to the expected. The expected time was 13.1 s and the slowest time measured is 13.6 s, an error of 3.8% -- much lower than the obtained for the simple model.

3.3.7 Final Model for Data Distribution

The enhanced model provides a much more accurate prediction of the time to transfer the input data and process it in the worker nodes. However, the duration of the whole execution is still close to the simple model – 13.9 s for the simple model versus 13.6 s for the enhanced model, both measured times. The time distribution presented in Figure 34 shows a lot of idle time, especially nodes B1 and B2, which have more processing capacity.

3.3.7.1 Shared Data and Chunk Data

The final model is a conjunction of the simple model and the enhanced model plus special methods to distribute data. First of all the input data is organized in shared data and chunks, as presented in Figure 35. The figure shows an example of multiplication of matrices A and B . One possible distributed implementation is to send one row of matrix A and the complete matrix B for each worker element. In this case, the matrix A row is a chunk and matrix B is shared data, because for every iteration of the algorithm one row from A and the whole matrix B are processed.

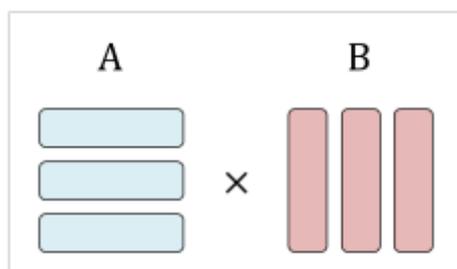


Figure 35 – Visual example of matrix multiplication data structures

Figure 36 shows an example of chunk and shared data in three iterations. This example represents how a multiplication of two matrices 3×3 would be executed in one node. In each iteration a chunk of the matrix A is sent to the worker. Moreover, before starting the computation the shared matrix B is sent to the worker.

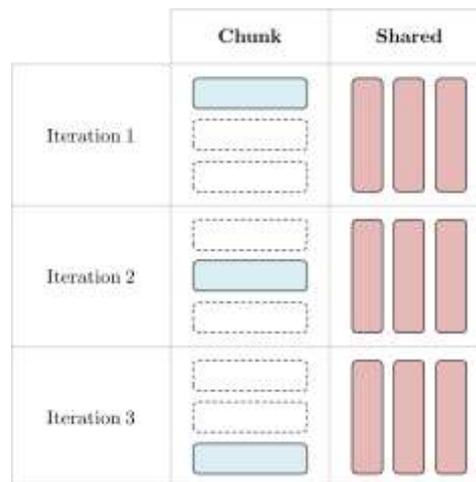


Figure 36 – Example of chunk and shared data

Note that the distinction between what is chunk and what is shared data relies on the workflow implementation and characterization. For instance, it is also possible to implement the matrix multiplication only with chunk data, as depicted in Figure 37.

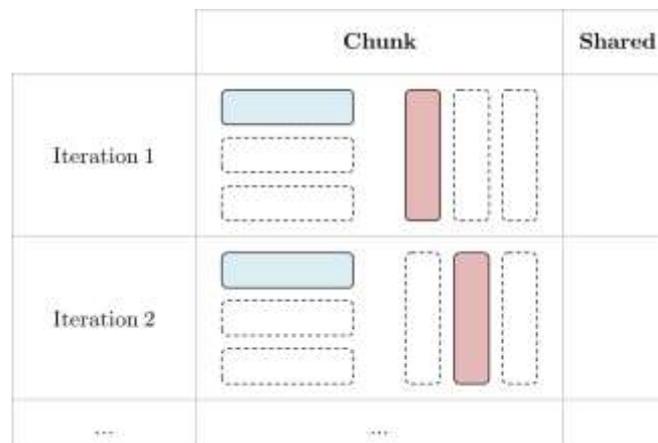


Figure 37 – Example of matrix multiplication implementation using only chunk data

3.3.7.2 *Distribution of Shared Data*

With this distinction between chunk and shared data it is possible to implement different strategies to distribute this data. Shared data is required by multiple iterations while chunk data is required only for a single iteration. Moreover, shared data is required by multiple nodes. For matrix multiplication with fixed matrix B all nodes require it. In this sense, the shared data can be distributed in an exponential fashion, similar to the spread of a virus using a selective neighbor pattern based on time cost.

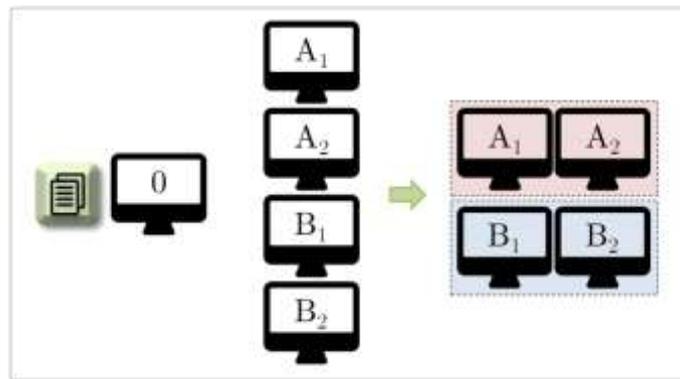


Figure 38 – Distribution of shared data: node clustering

The first step of the distribution of shared data is the clustering phase, which is presented in Figure 38. The nodes that are going to receive data are grouped based on their proximity, which is either calculated via resource description information or determined via network tests regarding available bandwidth. Once the clusters are defined the input node 0 starts by sending data to one node from each cluster. Figure 39 illustrates this behavior.

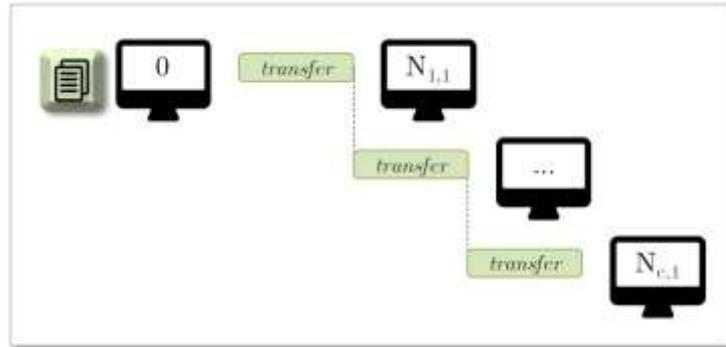


Figure 39 – Input sending to one node from each cluster.

In this figure the nodes are identified as $N_{c,n}$, where c is the cluster number (from 1 to C , C being the number of clusters) and n is the node number in the cluster (from 1 to N_c , N_c being the number of nodes from cluster c).

After one of the cluster nodes has the inputs, this node can start sending the shared input to a neighbor node (i.e., a node of the same cluster). Note that sending the shared input in a sequence does not lead to a performance loss due to idling, as observed in the enhanced model. The difference in this case is that concurrent transfers do degrade performance and all nodes must have the shared data to start computation. Sending the shared input in sequence minimizes contention effects and maximizes communication channel utilization.

Considering that the amount of data each node receives is D_s (data shared) and that this amount is the same for all nodes, the time for a node $N_{c,1}$ to wait and then receive data is:

Equation 29 – Time for a first node to receive the shared data

$$t_s(N_{1,1}) = \frac{D_s}{r_0}$$

$$t_s(N_{2,1}) = \frac{D_s}{r_0} + \frac{D_s}{r_1}$$

$$t_s(N_{c,1}) = \sum_{m=1}^c \frac{D_s}{r_m}$$

Figure 40 shows the distribution of shared data to a specific cluster c before the first spread. This is showing the transfer from the input node 0 to the first node of the cluster, $N_{c,1}$.

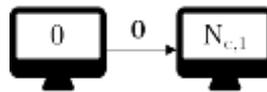


Figure 40 – Distribution of shared data to a cluster; before first spread

Figure 41 shows the scenario after the first spread. Node $N_{c,1}$ sends the shared data to node $N_{c,2}$, which is selected based on the bandwidth between the first node and the other nodes of the cluster. The index 1 on top of the arrow identifies that this transfer occurs in the first spread of the algorithm.



Figure 41 – Distribution of shared data to a cluster; after first spread

Figure 42 shows the scenario after the second spread. Node $N_{c,1}$ sends data to $N_{c,3}$ and node $N_{c,2}$ sends data to $N_{c,4}$.

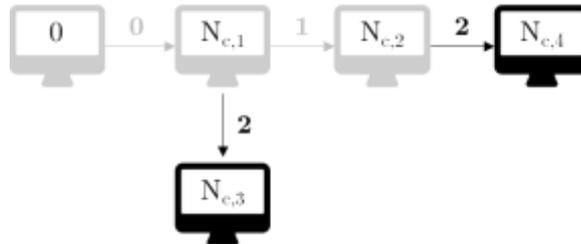


Figure 42 – Distribution of shared data to a cluster; after second spread

Figure 43 shows the scenario after the third spread

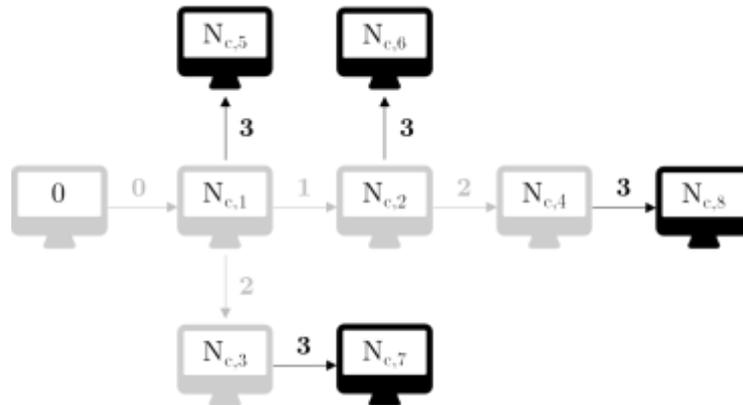


Figure 43 – Distribution of shared data to a cluster; after third spread

The time for a node $N_{c,n}$ to receive data can be computed as:

Equation 30 – Time for a node to receive the shared data

$$t_s(N_{c,1}) = \sum_{m=0}^c \frac{D_s}{r_m}$$

$$t_s(N_{c,2}) = t_s(N_{c,1}) + 1 \cdot \frac{D_s}{r_c}$$

$$t_s(N_{c,3}) = t_s(N_{c,4}) = t_s(N_{c,1}) + 2 \cdot \frac{D_s}{r_c}$$

$$t_s(N_{c,5}) = t_s(N_{c,6}) = t_s(N_{c,7}) = t_s(N_{c,8}) = t_s(N_{c,1}) + 3 \cdot \frac{D_s}{r_c}$$

$$t_s(N_{c,n}) = t_s(N_{c,1}) + \lceil \log_2 n \rceil \cdot \frac{D_s}{r_c}$$

Where:

- $t_s(N_{c,1})$ is the time for the first node of the cluster to receive data;
- $\lceil x \rceil$ is the ceiling function, which returns the smallest integer greater or equal than x ; for example, $\lceil 1.0 \rceil = 1$, $\lceil 1.5 \rceil = 2$; and
- r_c is the transfer rate among nodes of the same cluster c ; this value is assumed to be the same or roughly the same among nodes of the same cluster.

The time for the last node of the cluster is:

Equation 31 – Time for a node to receive the shared data

$$t_s(N_{c,N}) = T_{s,c} = t_s(N_{c,1}) + \lceil \log_2 N_c \rceil \cdot \frac{D_s}{r_c} \Rightarrow$$

$$t_s(N_{c,N}) = T_{s,c} = T_{0,c} + T_{i,c}$$

The formula was simplified using the components $T_{0,c}$ to indicate the time for the first node of the cluster to receive data from the input node, and the component $T_{i,c}$ that indicates the time to propagate the input internally in the cluster. Note that the second component T_c in this equation is independent of the order the clusters were organized.

The total duration of the distribution of shared data is calculated by:

Equation 32 – Time for a node to receive the shared data

$$T_s = \max \left(\left(\sum_{i=1}^c T_{0,i} \right) + T_{i,c}, \forall c < |C| \right)$$

For instance, a setup with two clusters leads to the following:

Equation 33 – Example with two clusters

$$T_s = \max(T_{0,1} + T_{i,1}, T_{0,1} + T_{0,2} + T_{i,2})$$

The expression of Equation 33 leads to the definition of several equations depending on the order the clusters are organized. For instance, if there are C clusters then there are $C!$ possible configurations of clusters. For each combination it is necessary to calculate C components, then calculate the maximum value among these components. Thus, for a large number of clusters this computation may become expensive. However, the number of clusters directly depend on the complexity of the network hierarchy and associated different costs, which should not be numerous. Moreover, the clustering phase relies on a static threshold and a dynamic threshold to create the clusters. The dynamic threshold is used to create clusters based on different bandwidth values, while the static threshold is used to avoid the creation of clusters for small variations, even if the dynamic threshold determines that the variation is relevant. For instance, Figure 44 shows the clustering based on the dynamic threshold only. It detects three clusters, one with nodes with bandwidth around 10 MB/s, another with around 20 MB/s, and another around 100 MB/s.

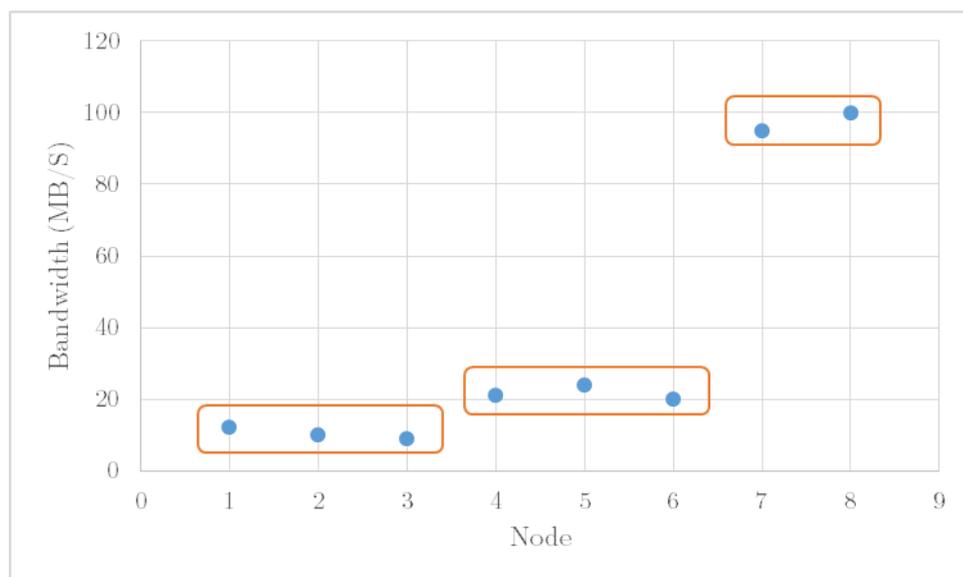


Figure 44 – Graph showing clustering based on dynamic threshold

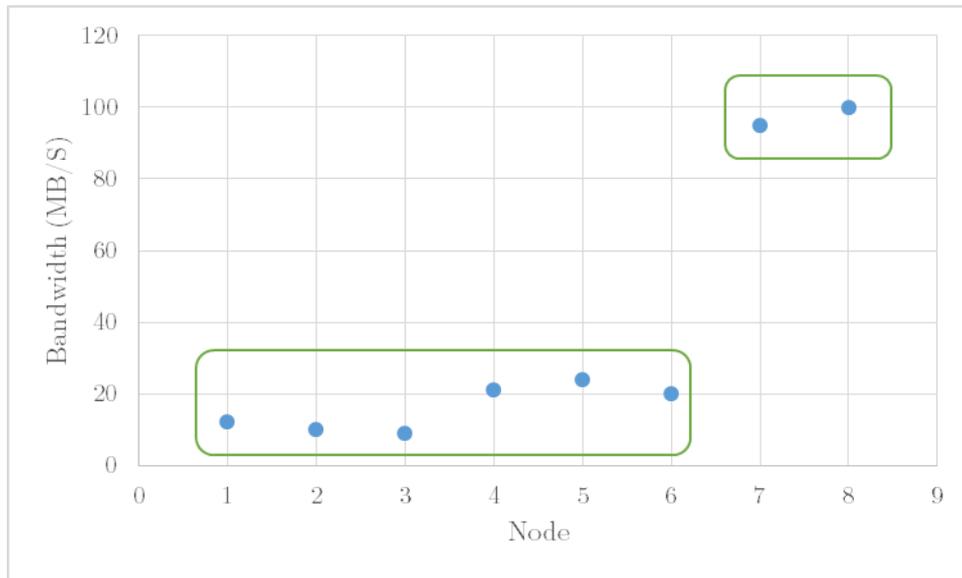


Figure 45 – Graph showing clustering after applying static threshold

Then, applying the static threshold leads to the results of Figure 45. A threshold with granularity of 50 MB/s was defined, thus the initial clusters of 10 MB/s and 20 MB/s were naturally joined together. These thresholds are defined to detect and also reduce the number of clusters, which consequently simplifies the calculations done by the framework to distribute the shared data.

Still on the distribution of shared data, note that if the value of $T_{0,c}$ is similar for different clusters, $T_{0,c} = T_0$ and the equation can be simplified to:

Equation 34 – Time for a node to receive the shared data when bandwidths are similar

$$T_{s,c} = c \cdot T_0 + T_{i,c}$$

This assumption is valid when the nodes are clustered by the dynamic threshold but joined back together by the static threshold. Moreover, the values are also similar when the cluster is created by simply dividing nodes of the same cluster (*sub-clustering*). This technique is applied when there is imbalance between the number of nodes of each cluster, or when there

are too many nodes in the same cluster. For instance, if all nodes belong to the same network, the bandwidths will be roughly the same and thus all nodes will be organized in a single cluster. To implement the clustering and division technique these nodes are subdivided in smaller groups. This avoids idling the input node for too long. For a cluster with N nodes the time to distribute the shared data from a single input node considering parallel transfers and assuming perfect transfers (i.e., no contention) is:

Equation 35 – Shared data distribution with a single input node and perfect transfers

$$T_{s,parallel} = (N) \cdot \frac{D}{B}$$

Although bandwidth is shared (equally, in the ideal model), the transfers occur in parallel. In contrast, with a method of partitioning and spreading the time is:

Equation 36 – Shared data distribution with spreading method

$$T_{s,spreading} = (\log_2 N) \cdot \frac{D}{B}$$

The spreading technique does not rely on multiple accesses to the same resource, drastically reducing the possibility of contention issues. In addition, the formula for the parallel technique assumes an ideal behavior of multiple I/O accesses, which has experimentally shown a deviation of almost 50% of the expected value. Even so, the spreading technique clearly outperforms parallel transfers.

The expression from Equation 36 can also be used to calculate the total duration of distribution of shared data if all nodes belong to the same cluster. In practical terms multiple clusters are created in hybrid and multicloud scenarios, wherein the bandwidth between the input node and the worker nodes may vary a lot depending on the network link between clouds.

Consequently, the number of clusters would not be much higher than ten, at least for current distributed cloud setups. Lowering the thresholds, however, can increase the number of clusters. Nevertheless, applying formula from Equation 32 should not be computationally expensive for current cloud setups.

3.3.7.3 *Distribution of Chunk Data*

Chunk data is an enabler, in the sense that once the worker node receives a chunk it has enough data to start computation (assuming that shared data was distributed as part of the beginning of workflow execution, marked by the *Input* primitive). Different from shared data, however, chunk data is not common throughout iterations, thus it is not possible to implementing a similar spreading technique to reduce final distribution cost.

Chunk data is distributed in a weighed round-robin fashion based on the enhanced model presented in Section 3.3.5. However, instead of sending all data to one node and then moving to the next one, the chunk data is distributed based on specific amount of chunks while attaining to the ratio calculated by the enhanced model. This attempts to minimize long periods of idle time and it favors workflows wherein it takes longer to process a chunk of data than sending it to the node. Transmission of chunks usually follows a linear pattern, meaning that the cost of a transfer usually varies within $\mathcal{O}(n)$ of data size. If the workflow phase implements an algorithm of higher complexity than the transfer cost, MPSF calculates the optimal number of chunks to distribute at each iteration wherein processing this set of chunks takes longer than sending them and also waiting for the other nodes to receive their chunks. By sending this number of chunks the node will not be idle in terms of processing. The expression from Equation 37 represents this condition.

Equation 37 – Condition for zero idle time

$$t_p \geq t_r + t_w = \sum_{all} t_r \rightarrow t_{idle} = 0$$

Where t_p is the chunk process time, t_r is the chunk transfer time, and the t_w is the waiting time for the other chunk transfers. The sum of the node transfer and the other transfers is simply the sum of all transfers. If t_p is shorter than this sum then the idle time is zero.

On the other hand, if $t_p < t_r + t_w$ then the node will experience idle time. The enhanced model defines a distribution wherein the amount of idle time drastically increases based on the order that each nodes receives data. For instance, if the system is composed by a set N of n nodes with the same processing and transfer rates, the following distribution would be observed:

Equation 38 – Distribution for exemplar scenario

$$\frac{\frac{1}{r_{i+1}} + \frac{1}{p_{i+1}}}{\frac{1}{p_i}} = \frac{\frac{1}{r_{j+1}} + \frac{1}{p_{j+1}}}{\frac{1}{p_j}} \therefore \beta_i = \beta_j, \forall i, j \in N$$

$$\rho_i = \beta^{n-i} \rightarrow d_i = \frac{\beta^{n-i}}{\pi} D \rightarrow d_i = \frac{\beta^n}{\pi} D \cdot \frac{1}{\beta^i}$$

This is an exponentially decaying distribution. For instance, if $p_i = r_i = v$, each subsequent node receives half of the data of the previous node.

Equation 39 – Distribution for exemplar scenario

$$\beta_i = \frac{\frac{1}{r_{i+1}} + \frac{1}{p_{i+1}}}{\frac{1}{p_i}} = \frac{\frac{1}{v} + \frac{1}{v}}{\frac{1}{v}} = 2 \rightarrow d_i = \frac{\beta^n}{\pi} D \cdot \frac{1}{2^i}$$

However, one must notice that the idle time is only due to how data is distributed. Figure 46 compares the data distribution methods.

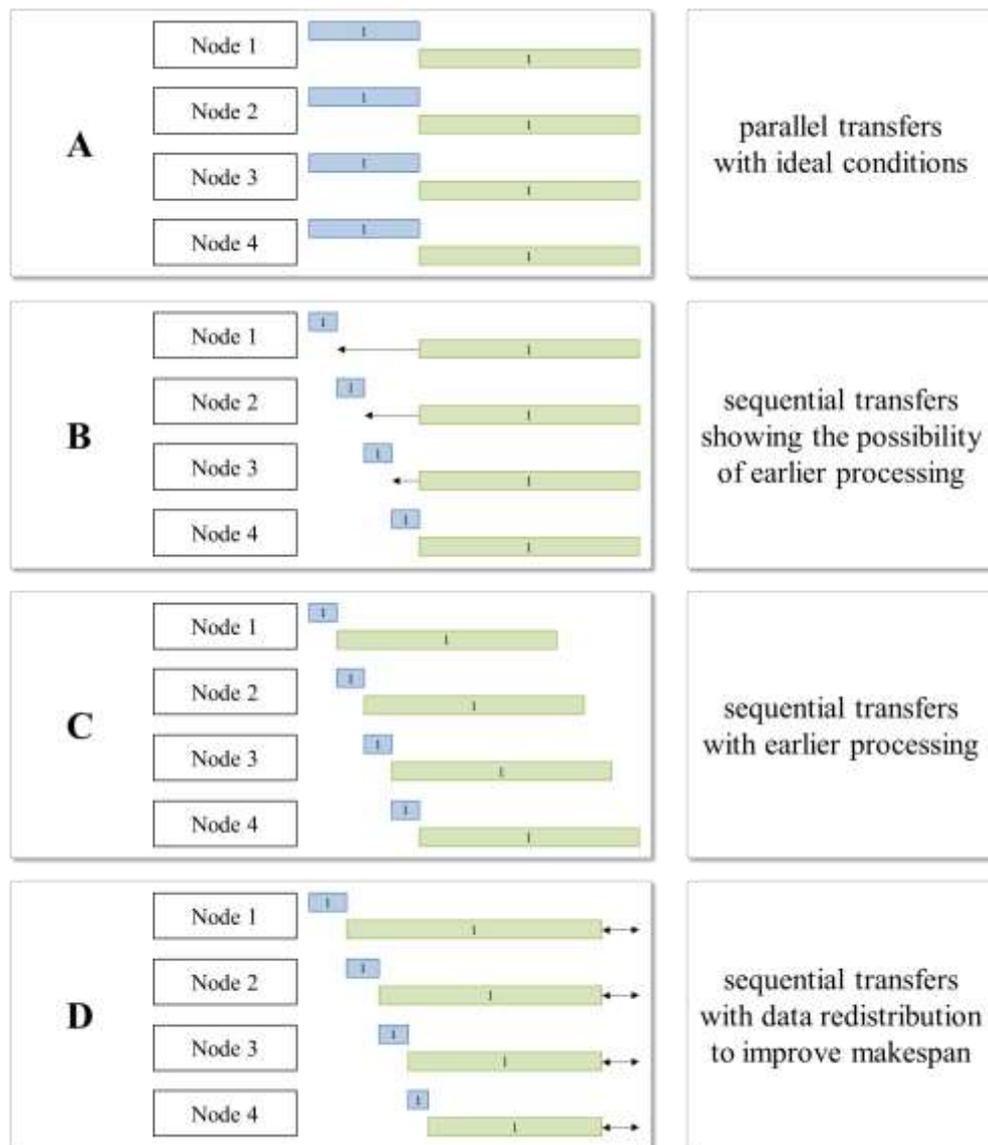


Figure 46 – Data distribution techniques. For each node, first bar is transfer and second bar is processing.

Scenario A represents the simple data distribution with shared transmission channel. Note that this representation is assuming a perfect environment, without any contention effects. Scenario B represents the exact same amount of data and processing, but sending to one node at a time. Because the communication is not shared anymore, the read rate does not have to be

split among multiple readers. Moreover, computation does not have to wait for all transfers to complete, it may start as soon as each transfer has finished. This leads to scenario C, in which computation immediately follows the transfers for each node. While the last node finishes roughly at the same time as in scenario A, the other nodes finish before. This suggests that each of these nodes could receive more input in the meantime, or that the input distributed among the nodes could be better distributed so that the final time for all nodes is the same. This is the logic for the enhanced model, which is represented by scenario D.

3.3.7.4 Equalization for $t_p \geq t_r + t_w$

In the examples of Figure 46 the processing times are always longer than the transfer times. As stated by Equation 37, there is a chunk size that leads to a t_p greater than t_r plus t_w . In this sense, Figure 47 expands the scenario D presented in Figure 46. Because the time processing is longer than the time for the transfers, it is possible to start the next transfers (e.g., of the second iteration) before the computation finishes. The perfect chunk size can be calculated as a weighed distribution based on the processing rate p_i of each node, as presented in Equation 40.

Equation 40 – Optimal chunk distribution for $t_p \geq t_r + t_w$

$$\frac{d_i}{p_i} = \sum_{j=1}^n \frac{d_j}{r_j}, \forall i \in N \Rightarrow \frac{d_i}{p_i} = \frac{d_j}{p_j}, \forall i, j \in N \therefore \text{weighted distribution}$$

$$d_i = \frac{p_i}{\sum_{j=1}^n p_j} D$$

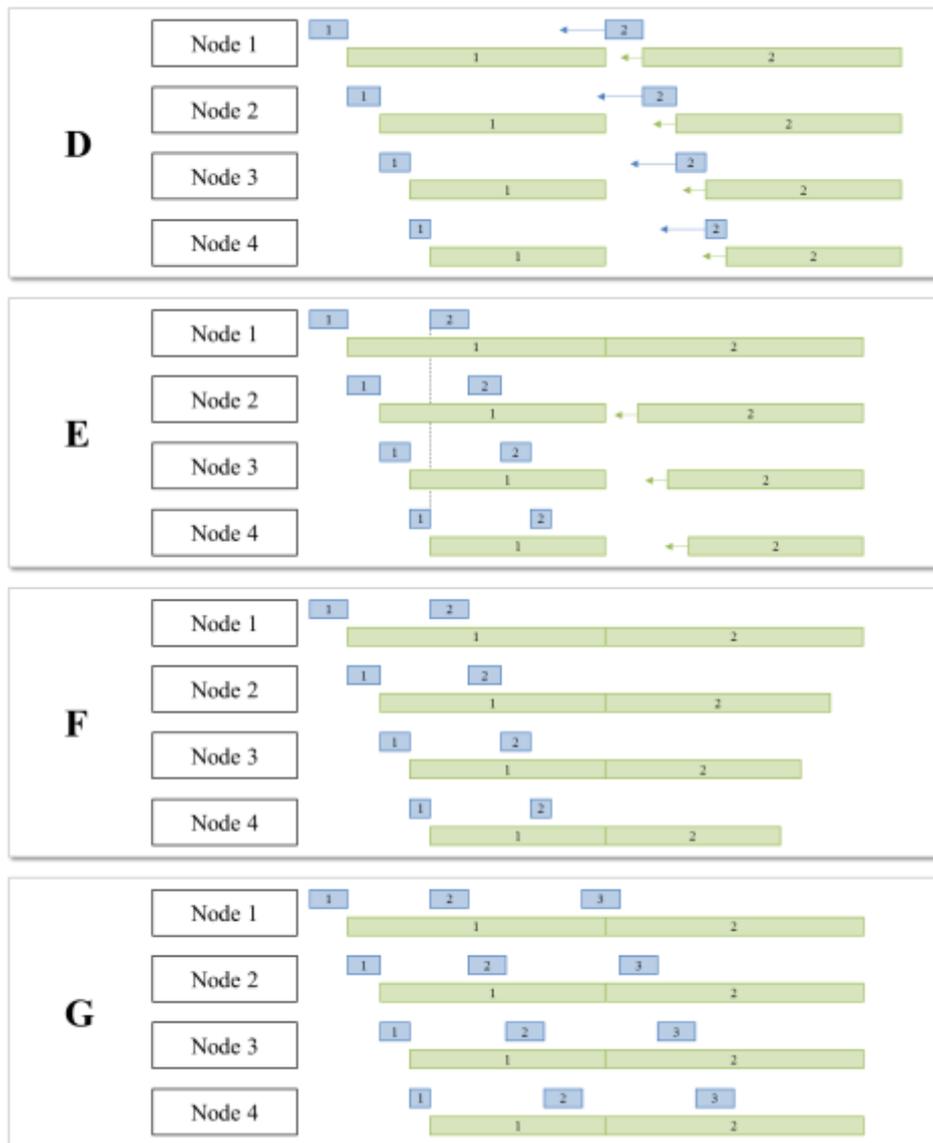


Figure 47 – Chunk distribution when $t_p \geq t_r + t_w$

The scenarios presented in Figure 47 describe the equalization of chunks, which is appropriate when the time to process is longer than the transfers. Scenario D shows the original distribution using the enhanced method. Once it is detected that transfers are faster than processing, these transfers can be concatenated in sequence, as shown in scenario E. The transfers for the second iteration begin right after the transfers for the first iteration are finished. By the time that computation has finished, the data for the next iteration is already available.

This availability of data also allows computation to start earlier, which is also depicted in scenario E. However, the computation for the subsequent nodes does not have to follow the same time distribution as defined in scenario D. In other words, the difference between starting times of nodes 1 and 2 in scenario D, for instance, does not have to be the same in scenario E. Thus, scenario F shows all nodes starting computation at the same time, as all nodes already have the necessary data to process. However, because computation depends on the size of the input, now the computation in the first node is unbalanced, as it was balanced before based on the premise that computation had to be harmonized with transfer times. To solve this imbalance, either more input has to be distributed to the other nodes or less input has to be distributed to the initial nodes. The optimal chunk ratio is defined by the expression in Equation 40, which is a weighted average based on the processing capacity of each node. Consequently, when the time to process is greater than the time for the transfers, an optimal chunk size is calculated in a way that chunk distribution is equalized via the processing rates of each node. Note that this equalized distribution could be used from the beginning.

3.3.7.5 *Pipelining and Phase Combination for $t_p < t_r + t_w$*

The scenarios presented in Figure 46 and Figure 47 depict the data and time distribution when the time to process is greater than the time for transfers. In contrast, Figure 48 presents the time distribution (transfers and processing) when the time to process is shorter than the transfers. The transition from scenario A to B is going from parallel transfers (again, on a perfect environment without contention effects) to sequential transfers. Because the read bandwidth is not shared in scenario B, the final time for the transfers is the same.

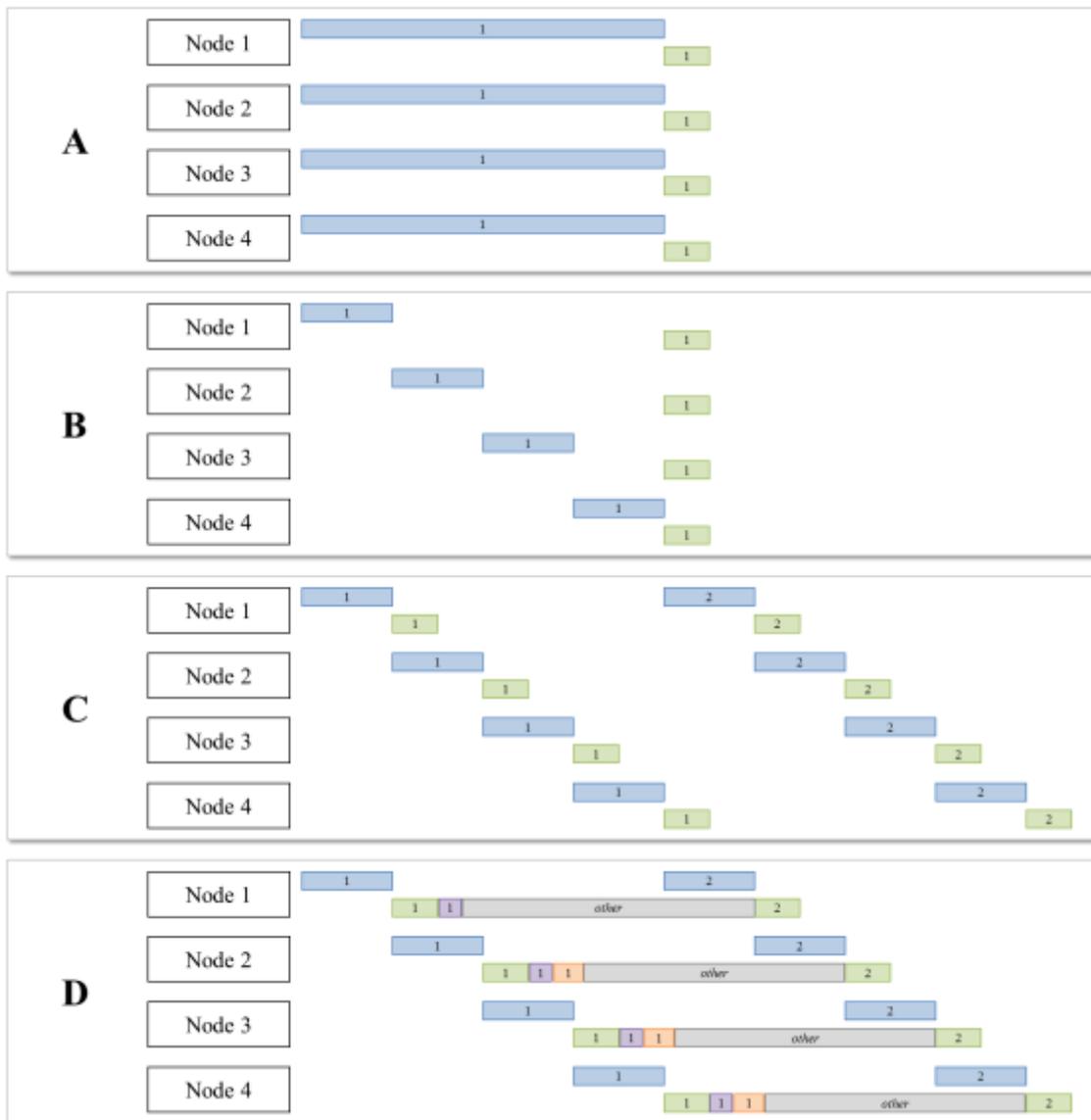


Figure 48 – Chunk distribution for $t_p < t_r + t_w$

The transition from scenario B to scenario C shows the processing starting right after each transfer is completed. Scenario C also shows the time components of the next iteration. There is a considerable idle period for each node in terms of processing, and there is no possibility of equalization as the time to transfer will always be longer than processing. However, two strategies can be implemented to optimize the utilization of the resources: pipelining and workflow phase combination.

Pipelining refers to concatenating subsequent operations of the workflow. Scenario D shows an example of concatenation of three sequential tasks. The first task depends on the input data coming from the input node, thus this task relies on the initial transfer which is controlled via the enhanced model for chunk distribution. However, the other two tasks depend on the outputs of the first task. For instance, the first task could apply a filter to the results, and the second task is a merge of these results to produce a partial result based on the chunks processed. In this sense, these tasks, which are represented by different workflow phases, can be combined and executed in the form of a pipeline of instructions.

Workflow phase combination refers to the combination of tasks of different workflows, particularly tasks related reliability (e.g., replication) and tasks that can be optimized via equalization. In this sense, workflow phases with t_p relatively longer can be combined to workflow phases with t_p relatively shorter. The combination can be implemented via best effort to simplify calculations, in the sense that the tasks active in a particular node or set of nodes are analyzed and any possibilities of phase combination among these tasks are identified. Any tasks that fit the free slots (identified as *other* in the figure) are reassigned.

3.3.8 Probabilistic Model for Performance Fluctuations

Performance fluctuations affect makespan particularly for long-duration workflows, or workflows with long-duration phases. The initial load distribution is calculated based on parameters obtained a priori, such as properties of resources and their availability. To account for fluctuations one possibility is to implement tests and other forms of verification to quantitatively analyze the discrepancy between the original performance figures and the current ones.

3.3.8.1 Correction Factors

To integrate this performance analysis to the actual data distribution it is possible to define correction factors that are applied to the data distribution in order to address the fluctuations identified in the analysis. Table 18 illustrates the utilization of these factors.

Table 18 – Distribution after applying correction factors

Node	d_i	f_i	d'_i	d'_i (norm)
1	40%	50%	20%	50%
2	60%	33%	20%	50%

The corrections factors f_i are used to redistribute the input data d_i . In the initial distribution node 1 would receive 40% of the input and node 2 would receive 60%. The correction factors determine that node 1 should receive only 50% of that amount as the is reasonably occupied. In contrast, node 1 should receive only 33% of that amount as the node is severely occupied. This leads to a final distribution of 20% to each node. Normalizing the values to 100% leads to a distribution of 50% for each node. The whole process of determining the correction factors and applying them to the data distribution is depicted in Figure 49.

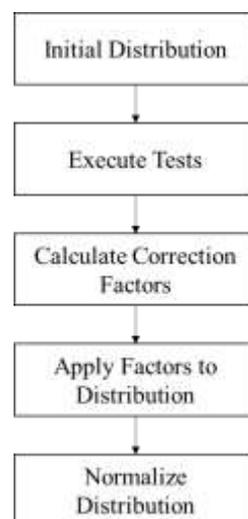


Figure 49 – Process of applying correction factors to data distribution

3.3.8.2 Bayesian System of Multiple Hypotheses and Multiple Evidences

The correction factors are calculated using Bayesian probability based on multiple hypotheses and multiple evidences. The hypotheses are based on the load level of a node. Table 19 presents an example of discretization of levels.

Table 19 – Load level and respective hypothesis

H_i	f_i	Description
1	100%	Node is free
2	75%	Light load, lots of available resources
3	50%	Medium load; node can still receive some more load
4	25%	Heavy load; node is almost unusable
5	0%	Node is completely busy

Each row from this table represents a hypothesis identified as H_i . Each hypothesis is associated to a correction factor f_i , which should be applied to the data distribution according to the load level of the node.

The next step is to determine the system of hypotheses and evidences to be used to compute whether how busy a node is. An example of such system is presented in Table 20.

Table 20 – Bayesian System of multiple hypotheses and evidences

	e_1^1	e_2^1	e_3^1	e_1^2	e_2^2	e_3^2	e_1^3	e_2^3	e_3^3	<i>a priori</i>
H_1	0.30	0.10	0.05	0.50	0.05	0.01	0.80	0.04	0.01	0.10
H_2	0.45	0.30	0.15	0.30	0.25	0.10	0.15	0.20	0.04	0.50
H_3	0.20	0.50	0.15	0.15	0.55	0.10	0.04	0.60	0.05	0.35
H_4	0.04	0.05	0.60	0.01	0.11	0.70	0.01	0.15	0.80	0.04
H_5	0.01	0.05	0.05	0.01	0.04	0.09	0.00	0.01	0.10	0.01

The e_r^t values represent the multiple evidences that can be collected in the system. For each e_r^t value, the t index identifies a **test** and the r index identifies a **result** or outcome of that test. Table 21 specifies each test and the results.

Table 21 – Specification of tests and results

t	Description	r	Description
e^1	Simple test, low cost	e_1	Light load
e^2	Normal test, medium cost	e_2	Medium load
e^3	Complex cost, high cost	e_3	Heavy load

For instance, if test 1 is executed and it returns that a node has medium load, this is represented by e_2^1 . The idea is to define multiple tests that can be executed to assess the conditions of the system. Simpler tests are easier and faster to execute, but they are also less accurate than more complex tests. This fact is observed in Table 20. The combination of a row and a column represents the probability of each hypothesis given that the test returned a particular result – in other words, $P(H_i | e_r^t)$, the probability of hypothesis H_i given evidence e_r^t . For example, the combination of row H_2 with column e_2^1 is 0.30. This means that if test 1 is executed in a node and it returns that the node has medium load, there is a 30% chance that the node actually has a light load, thus the factor of 75% should be applied to the distribution. Similarly, if test 1 is executed and it returns evidence 2 (medium load), there is a 10% chance that the node is completely free, 50% chance that the node is under medium load, 5% chance that the node is under heavy load, and 5% chance that the node is completely busy.

Note that the fact that the node might be completely busy even if the test returns that the node has a medium load suggests that this test is not extremely accurate. In parallel, if test 3 returns evidence 2, e_2^3 , the chance of the node being completely busy is 1%.

3.3.8.3 Calculating the Correction Factors

The correction factors are calculated based on the outcome of the test executed. Tests can be executed in order of complexity or based on the desired accuracy. Moreover, two stopping criteria are adopted: one based on the accuracy obtained by executing a particular test, and another based on the accuracy delta between two subsequent tests executed.

For instance, if test 1 was executed and it resulted in evidence 2, the probabilities for each hypothesis (node condition) are valid:

Table 22 – Probabilities of hypotheses based on the test and the result

	e_2^1
H_1	0.10
H_2	0.30
H_3	0.50
H_4	0.05
H_5	0.05

The final probabilities are calculated by combining these values to the probability of each hypothesis a priori, which is indicated in Table 20. The calculations are presented in Table 23.

Table 23 – Calculations for probability of hypotheses

$H'_1 = P(H_1 e_2^1) \cdot H_1 = 0.10 \cdot 0.10 = 0.0100$
$H'_2 = P(H_2 e_2^1) \cdot H_2 = 0.30 \cdot 0.50 = 0.1500$
$H'_3 = P(H_3 e_2^1) \cdot H_3 = 0.50 \cdot 0.35 = 0.1750$
$H'_4 = P(H_4 e_2^1) \cdot H_4 = 0.05 \cdot 0.04 = 0.0020$
$H'_5 = P(H_5 e_2^1) \cdot H_5 = 0.05 \cdot 0.01 = 0.0005$

The new values for the hypotheses have to be normalized. A factor α is calculated:

Equation 41 – Normalization of hypotheses

$$\frac{1}{\alpha} = \sum_{i=1}^H H'_i$$

$$H'_i = H_i \cdot \alpha$$

Where H is the number of hypotheses considered in the model. In this example:

Equation 42 – Normalization factor for example

$$\frac{1}{\alpha} = 0.0100 + 0.1500 + 0.1750 + 0.0020 + 0.0005 = 0.3375$$

$$\alpha = 2.9630$$

The final values for the hypotheses are:

Table 24 – Final values for hypotheses

$$H'_1 = 0.0100 \cdot 2.9630 = 0.0296$$

$$H'_2 = 0.1500 \cdot 2.9630 = 0.4445$$

$$H'_3 = 0.1750 \cdot 2.9630 = 0.5185$$

$$H'_4 = 0.0020 \cdot 2.9630 = 0.0059$$

$$H'_5 = 0.0005 \cdot 2.9630 = 0.0015$$

3.3.8.4 Using the Correction Factors

Based on these results, the predominant hypothesis is H_3 , thus the node is considered to be under medium load and a factor of 50% should be applied to the final data distribution. Table 25 compares the values of the hypotheses before and after the test.

Table 25 – Comparison of hypotheses values before and after test

	a priori	after test
H_1	0.1000	0.0296
H_2	0.5000	0.4445
H_3	0.3500	0.5185
H_4	0.0400	0.0059
H_5	0.0100	0.0015

Note that using the a priori distribution would lead to the H_2 hypothesis, thus to a correction factor of 75%. After executing the test the decision system verifies that the node is busier than expected, and a factor of 50% should be applied. In addition, this system can be used in different ways in order to combine the probabilities calculated. For instance, it is possible to calculate a weighted average between all probabilities in order to calculate an aggregate correction factor. Table 26 presents the calculations and the final result of this approach.

Table 26 – Utilization of probabilities to calculate weighted average for correction factor

	value	factor	weight
H_1	0.0296	1.00	0.0296
H_2	0.4445	0.75	0.3334
H_3	0.5185	0.50	0.2593
H_4	0.0059	0.25	0.0015
H_5	0.0015	0.00	0.0000
			0.6238

Using a weighted average leads to a correction factor of 0.6238 versus the correction factor of 0.7500 based on the approach that adopts the highest probability as the correct one. The strategy adopted depends on the requirements of the system and on the configurations made to the model and to the framework.

3.3.8.5 Building and Updating the Bayesian System

The Bayesian system can be built with pre-defined values or with fixed values. The pre-defined values can be defined via expert knowledge or based on the properties of the tests and the possible outcomes.

Either with pre-defined values or fixed values, the Bayesian system is able to update its own values based on the results and the knowledge accumulated by the system. After each iteration two dimensions are updated: the *a priori* values for the hypotheses, and also the accuracy of the internal probabilities. The *a priori* values are calculated at each iteration to define the load level of the node. In this sense, these values from Table 20 would be substituted by Table 24. Note that the internal values are common to all nodes, while the *a priori* values are specific to each node.

The internal values are updated globally for all nodes that use the same set of tests to assess their loads.

Equation 43 – Method to update internal probabilities

$$P'(H_i | e_r^t) = w_0 \cdot P_0(H_i | e_r^t) + w_i \cdot \frac{m_{H_i | e_r^t}}{\sum m}$$

Where P_0 is the initial probability assigned to each internal value and w_0 is a weight used to combine this initial static value to the dynamic values obtained during the utilization of the system. The dynamic values $m_{H_i | e_r^t}$ are based on simple counters or **markers** for each combination of hypothesis and evidence. The dynamic probabilities are calculated by dividing the number of marks for a particular combination by the total marks. For example, if a particular combination of test and result led to two occurrences of hypothesis 3 and one occurrence of

hypothesis 4, the probabilities for each case are 66% and 33%, respectively, while the others are 0%. The variable w_i is the weight for the values obtained in the iterations, typically equal to the number of iterations already performed. Thus, setting w_0 to a high value (e.g., 100) will give the initial static probabilities a heavier weight while the system starts to be used. As it gains more knowledge due to utilization, the weight w_i increases and the probabilities for each combination become more important in the calculation, finally overshadowing the initial static probabilities when $w_i \gg w_0$.

3.3.9 Comparison to Literature

The performance model provides several mechanisms to address data-intensive workflows by adequately distributing the load based on the properties of the workflows and their phases. Moreover, the probabilistic model provides a mechanism to handle performance fluctuations in the system without having to resort to expensive assessments of the system while workflows are being executed. While the mechanisms can be compared in terms of design and performance to other proposals for cloud scheduling, no other proposals have encompassed these aspects in an integrated way, addressing the particular properties of workflow phases regarding the duration of computation and data distribution. Moreover, the utilization of Bayesian probability improves the accuracy of data distribution by taking into account the variation of performance during the execution of workflows. Combined to the rescheduling policies defined by the framework, this improves both makespan and also the prediction capabilities of the solution.

The experimental verification for the final model for data distribution and load balancing is presented in Chapter 4.

3.4 Requirement Coverage

Table 27 presents a summary of the architectural requirements and how they were covered by the proposed components.

Table 27 – Architectural requirements and coverage

<i>#</i>	<i>Requirement</i>	<i>Coverage</i>
P1-SR5	The architecture must define components that implement the controls based on data movement to address the requirements of data-intensive workflows.	Data movement is covered by the Workflow Characterization component, which provides adequate description of data transfers and respective costs; by the Performance Modeling component, which implements the Performance Models that encompass the requirements of data-intensive workflows; and by the Task Execution Management component, that effectively executes the allocation task and addresses the requirements of data-intensive phases and tasks.
P2-SR5	The architecture must define components to address implementation-related issues comprising heterogeneous resources and large-scale management.	The particularities of heterogeneous resources and large-scale management are covered by the Environment Monitoring, which provides continuous assessment of resources to obtain their performance and reliability status; the Resource Discovery, which facilitates the modifications in the environment; the Execution Space Management, which implements the mechanisms to define and use execution spaces that are able to transparently integrate resources from different environments; the Resource Characterization, which automates the process of mapping performance metrics to resources; and Task Execution Management, which provides a comprehensive set of mechanisms to execute workflows in large-scale systems.
P3-SR5	The architecture must provide components associated to the management framework to implement the controls for hybrid/multicloud scenarios.	Hybrid and multicloud scenarios are covered by several components of the architecture, such as resource discovery, resource characterization, execution space management, task execution management, performance and reliability modeling, and requirements framework.

Table 27 – Architectural requirements and coverage (cont. I)

<i>#</i>	<i>Requirement</i>	<i>Coverage</i>
P4-SR5	The architecture must define components to implement the phases, triggers, and thresholds defined by the scheduling framework.	This requirement is covered by the execution space management, reallocation management, and trigger and threshold management. These components implement the controls to manage reallocation activities.
AUX1	The architecture must support mechanisms for resource discovery, enabling automated detection of resources added to the cloud environment being managed by MPSF.	The resource discovery architectural component is responsible for automating the processing of discovering resources and initiating the task of characterizing the resource.
AUX2	The architecture must support automation of resource characterization.	The resource characterization architectural component is responsible for automating the process of analyzing the resource performance based on fixed benchmarks (absolute) and based on the workflows and tasks to be executed.
AUX3	The architecture must support automation of workflow characterization, providing mechanisms that automate the process of definition of costs and other properties to workflow phases.	The workflow characterization component is responsible for automating the process of gradually executing the workflow with different input parameters to assess the sensitivity of the workflow. Then, using a set of regression functions, the performance behavior of the workflow is described as function that is further used to predict the makespan of tasks.

3.5 Chapter Considerations

This chapter presented details of MPSF, Multiphase Proactive Scheduling Framework, the proposed framework for resource management and workflow execution. Section 3.1 started by defining the solution requirements for MPSF. First it revisited the problems defined in Chapter 2. Three classes of problems were defined: data-intensive workflows, hybrid and multicloud scenarios, and rescheduling and performance fluctuations. Five solution requirements were defined based on these problems: 1) a workflow model to describe the properties of workflows to address the specific requirements of data-intensive workflows and workflows phases; 2) a resource model to describe the properties to allow addressing small-scale and large-scale deployments, including hybrid and multicloud scenarios; 3) a performance

model that encompasses the particularities of data-intensive workflows by properly modeling data distribution among tasks and nodes; 4) a framework that defines scheduling phases to allocate and reallocate resources and tasks; and 5) an architecture that defines component to implement all these functionalities.

Section 3.2 presented an overview of MPSF. The framework receives the characterization of the workflow to be executed, the requirements to be observed during the execution of the workflow, the resources to be used, and the models to be used to calculate data distribution. Based on these elements, a set of features presented in Table 10 define the main functionalities desired for the solution.

Section 3.3 presented the performance model used in MPSF to calculate resource distribution. The main issue solved by this part of the framework is how to distribute the load, which is typically represented by input data, among the nodes of the execution space assigned to execute the task. The investigation presents several models, going from a simpler models with assumptions that are not completely valid, to an enhanced and then a final version of the model that is able to accurately predict the makespan of workflow phases and workflows by adequately calculating the data distribution. Moreover, the performance model defines a Bayesian system to gradually improve the resource distribution.

Finally, Section 3.4 presented how the requirements were covered by the proposed components.

4 EXPERIMENTS AND ANALYSIS

Experiments were conducted to evaluate MPSF compared to other approaches regarding resource management. Section 4.1 presents the experimental platform and scenarios considered in the experiments. Section 4.2 presents details about the implementation, as well as which components were selected for implementation. Section 4.3 presents the benchmarks selected to execute the tests. Section 4.4 presents the other methods used to compare and analyze MPSF. Section 4.5 presents the results of the experiments. Section 4.6 provides an analysis of the results. Section 4.7 concludes the chapter.

4.1 Experimental Platform and Scenarios

Three systems were used in the experiments to evaluate the performance of MPSF. A summary of the specifications of these systems is presented in Table 28. Two systems are located in the United States: **Gauss**, with ten nodes based on the Intel Xeon X5687 processor, and **Vulcan**, with sixteen nodes based on the IBM POWER8 processor. The third system is located in Brazil: **Mamona**, with seven nodes based on the Xeon E3-1230 processor. The Vulcan nodes provide much more resources per node in terms of compute capacity (up to 160 VCPUs versus 16 from Gauss and 8 from Mamona) and memory (512 MB per node versus 32 GB from Gauss and 16 GB from Mamona).

Table 28 – Specification of systems used in the experiments

	Gauss	Vulcan	Mamona
Location	United States	United States	Brazil
Nodes	10	16	7
Processor	Xeon X5687	IBM POWER8	Xeon E3-1230
CPU Clock	3.6 GHz	3.5 GHz	3.2 GHz
Cores/Socket	4	10	4
Threads/Cores	2	8	2
Sockets/Node	2	2	1
Total VCPUs	16	160	8
Memory/Node	32 GB	512 GB	16 GB
Memory/VCPU	2.0 GB	3.2 GB	2.0 GB
HDD/Node	4 TB	2 TB	500 GB
Switch	Mellanox	Mellanox	Netgear
	SX6025	SB7700	GSM7328SO
	56 Gb/s	100 Gb/s	10 Gb/s

All machines were configured to run Ubuntu Trusty 14.04. The cloud environments were set up using **Docker**⁷, enabling fast deployment of containers compared to virtual machines. The resources were partitioned in advance by allocating one container per VCPU due to the similarity in the amount of memory per VCPU among the three systems.

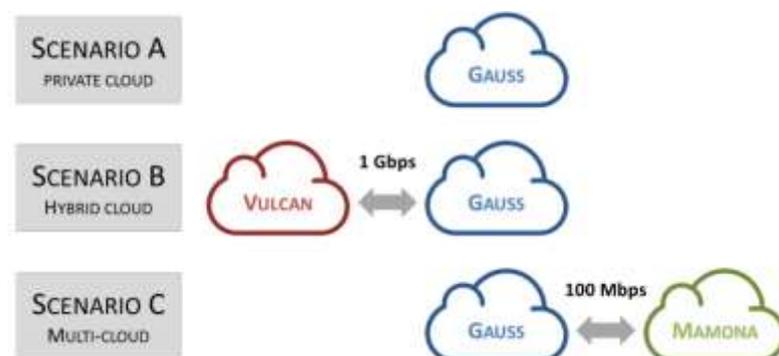


Figure 50 – Scenarios used in the experiments

⁷ <https://www.docker.com/>

Three scenarios were built for the execution of experiments, as illustrated by Figure 50. **Scenario A** is a private cloud built using nodes from system Gauss. **Scenario B** is a hybrid cloud combining the private cloud from Scenario A and a public cloud built using nodes from system Vulcan. Finally, **Scenario C** is a multicloud scenario combining the resources from scenario A to Mamona, a remote private cloud in Brazil.

4.2 Implementation

MPSF was implemented using Python as main programming language, although some support modules were built using Shell scripting (for *bash*). A simple command line interface was created to allow issuing commands to MPSF. From server-side an application was implemented to receive requests related to new workflows and to configure user-related parameters, such as requirements to run workflows on a specific set of nodes.

Regarding the components of the architecture, the environment monitoring was implemented using system tests such as *ping* and *iperf*. Moreover, tools were created to implement the performance tests or to integrate existing benchmarks, such as LINPACK and STREAM.

Resource discovery was implemented using a unicast-based protocol considering that the addresses of information servers are known. Only one information server was deployed in each environment. For resource characterization the aforementioned tests were automated to collect system properties. Resource characterization based on workflows was not implemented as automated workflow characterization was implemented already.

Task management comprised a simple pipeline to utilize the models (performance and reliability) and then to distribute tasks and coordinate the execution of phases. Finally,

reallocation was implemented using fixed triggers and thresholds, without implementing dynamic values for these controls.

4.3 Selected Benchmarks

Three benchmarks were selected for the experiments: AXPY (vector addition), GEMM (matrix multiplication), and WAVG (weighted average). These benchmarks were selected due to high compute intensity, because they represent kernels of important scientific applications, and also because of intense data utilization, especially for larger inputs. Each benchmark is described as follows.

4.3.1 AXPY

Implementation of the scaled vector addition in the form $y = \alpha x + y$, where x and y are vectors with N elements each and α is a constant. The calculation for one vector element from y is obtained by computing $y_i = \alpha x_i + y_i$, thus, the splitter can distribute the vector elements in a way that each data fragment is the tuple (i, x_i, y_i) , generating N tuples. The α constant is optimized to be sent only once per container, instead of including it in each tuple. Considering that each container m receives N_m tuples, the total number of elements sent to each container is $3 \cdot d_m + 1$. The output generated by each container is a partial representation of the y vector containing N_m elements with newly calculated values. Each container sends tuples in the form (i, y_i) to the merger, leading to a total of $2 \cdot N_m$ elements. The merge function simply aggregates the vector elements in order to generate the final result. Intermediary merging (i.e., partially merging results and relaying the aggregate to a main merger) is not advantageous

in this case as it does not save any network resources. The visual representation of this workflow is depicted in Figure 51.

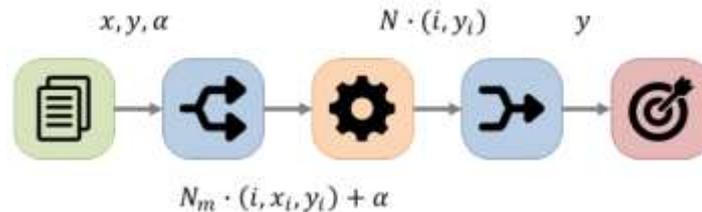


Figure 51 – Visual representation of AXPY

Vectors x and y are sent to the splitter, generating N fragments each containing an index and the x_i and y_i values to compute the result. Each compute component receives N_m fragments and the value of α , generating N_m tuples containing the index and the final value of y_i . The merger receives all N tuples generated by the compute components, each tuple containing an index and the final y_i value. These values are combined and the merger creates the final y vector. Depending on the size of the initial input and on the sizes of the partials generated by the splitter, MPSF is able to determine if I/O can occur via network and memory only or if disks are required.

4.3.2 GEMM

Implementation of the general matrix to matrix multiplication in the form $C = \alpha AB + \beta C$, where A , B , and C are matrices and α and β are constants. To simplify the implementation the matrices were set square matrices $N \times N$, and α and β were set to 1.0. The most basic component the splitter is able to generate contains enough information to calculate the value of a single cell of the resulting matrix: $C_{ij} = A_i \cdot B_j + C_{ij}$, where A_i is matrix A line i and B_j is matrix B column j . Thus, each fragment generated by the splitter is a tuple

(i, j, A_i, B_j, C_{ij}) composed by $2N + 3$ elements. In this process the splitter generates N^2 fragments. Each container receives N_m tuples and the output generated by each fragment processed by a container is a single cell of the resulting matrix, C_{ij} . The merger receives all parts and assembles the resulting matrix, and similar to AXPY, intermediary merging is not advantageous. The visual representation of this workflow and the interactions between each phase are depicted in Figure 52.

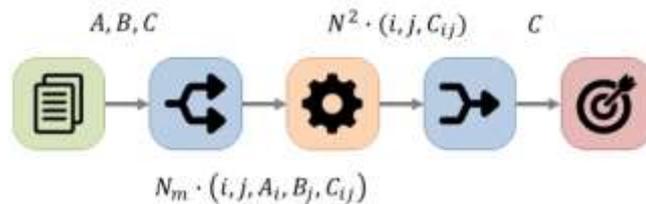


Figure 52 – Visual representation of GEMM

Matrices A , B , and C are sent to the splitter, producing N^2 tuples with enough information to compute a single cell of the resulting matrix. Finally, the merger receives N^2 result fragments, creating the final matrix C .

4.3.3 WAVG

Implementation of a weighted average algorithm given a vector of N values v_i and their associated weights w_i . The splitter receives both vectors and generates N fragments composed by the tuple (v_i, w_i) . Each compute component m receives N_m fragments and calculates a local weighted average, generating a result fragment represented by a tuple (avg_m, ws_m) with the local average and the sum of the weights. All result fragments are sent to the merger, which

relays these fragments to a second compute component that calculates the final average combining the partial results. This workflow is depicted in Figure 53.

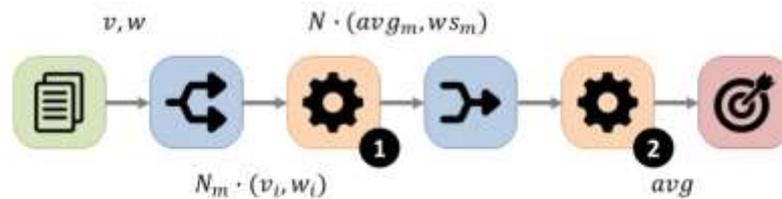


Figure 53 – Visual representation of WAVG

Vectors v and w contain the values and their weights. A second compute calculates the final average. The final average calculation was implemented in a dedicated compute component only to separate the responsibilities – it is perfectly possible to implement a merger that also calculates the final average.

4.4 Compared Methods

The methods compared in the experiments were:

- 1) **Single/Serial:** Execution on a single thread;
- 2) **Single/Parallel:** Single node using all cores;
- 3) **Naive/VCPU:** Naïve distribution strictly based on processing power and greedy over VCPUs (i.e., tries to use all VCPUs available);
- 4) **Naive/Core:** Naïve distribution greedy over cores;
- 5) **Naive/Node:** Naïve distribution greedy over nodes;

6) MPSF no-resc.: Partial implementation of MPSF without the rescheduling components; and

7) MPSF: The full implementation of MPSF.

The Naïve methods implement the resource management and scheduling mechanisms present in most works that focus on time to complete the tasks, such as makespan and deadline. These Naïve methods implement a basic form of resource distribution based on a weighted average to transfer and process data considering the workflow described in the characterization, without the models for data distribution proposed for MPSF. Thus, these Naïve methods represent a large class of scheduling solutions that focus on time to complete tasks without further mechanisms to address performance fluctuations and considering effects such as data and I/O contention.

4.5 Results

4.5.1 Scenario A: Private Cloud (Gauss)

The results for AXPY on Scenario A (private cloud) are summarized in Table 29. Experiments were repeated ten times and the results represent the average for each case.

Table 29 – Results (run time in seconds) for AXPY on Scenario A with vector size N of 500 million elements (8 GB of total footprint), 5 billion elements (80 GB), and 50 billion elements (800 GB).

Algorithm	500M	5G	50G
Single/Serial	4.96	68.15	-
Single/Parallel	0.67	22.78	-
Naïve/VCPU	0.22	3.09	62.05
Naïve/Core	0.21	3.01	49.87
Naïve/Node	0.32	4.12	54.10
MPSF no-resc.	0.14	1.19	28.15
MPSF	0.15	1.21	26.82

These results clearly show the importance of considering the I/O component during the resource allocation phase. The best naive approach in these experiments was the one based on cores (i.e., create one task per core and split input data accordingly). The run time for the 500M vectors (i.e., vectors with 500 million elements) was 0.22 s while the run time for MPSF without rescheduling was 0.14 s. Notice that subdividing the input from node-level to core-level led to an improvement of 34.4%, while doing so from core-level to VCPU-level actually deteriorated performance by 4.8% due to I/O contention. This contention becomes even more visible for larger vectors, such as the runs for 5G (5 billion) and 50G (50 billion) elements.

For 50G the memory footprint (800 GB) is larger than the total memory available in the system (320 GB), thus requiring intense disk I/O. Because the naive algorithms do not consider the I/O components in the initial resource allocation the performance is severely compromised. On the other hand, MPSF is able to gradually split the input based on the time to transfer data from the input node to the other nodes. Compared to the best naive approach, for 50G vectors MPSF provides a speedup of 1.86x. Moreover, the rescheduling mechanism provides an improvement of 4.7% by handling performance fluctuations in the system.

Table 30 - Results (run time in seconds) for GEMM on Scenario A with matrix side sizes N of 10k (total memory footprint of $3 \cdot (10k)^2 \cdot 8B = 2.4$ GB), 20k (9.6 GB), and 40k (38.4 GB).

Algorithm	10k	20k	40k
Single/Serial	82.34	667.31	-
Single/Parallel	12.10	188.40	-
Naïve/VCPU	1.49	13.12	252.13
Naïve/Core	1.52	14.39	128.93
Naïve/Node	8.48	102.26	979.90
MPSF no-resc.	1.38	13.17	122.35
MPSF	1.41	13.25	117.41

The results for GEMM on scenario A are summarized in Table 30. Matrix sizes are presented as the number of elements in each dimension (N). For the single node allocation techniques it is possible to run GEMM with 10k and 20k matrices without using file I/O and the run time varies according to number of operations (cube of the size increase, leading to 8x in number of operations which is similar to the run time increase observed). For multi-node configurations the performance increase due to parallelization is evident. In matrix multiplication the amount of data moved around is much smaller compared to vector addition, thus the transfer component is less expressive than the compute component. In terms of matrix size, the transfer component varies by the square of the increase in size and it is inherently smaller than the vector sizes for the AXPY experiments.

For the 20k matrix the Naive/VCPU distribution provides a slightly better result over MPSF, as the I/O contention effects are not expressive and due to the management overhead generated by the framework. However, for the 40k matrix the larger amount of data to be transferred favors MPSF, and the rescheduling mechanism also leads to some improvement by handling the performance fluctuations during the execution of the workflow.

Table 31 - Results (run time in seconds) for WAVG on Scenario A with vector size N of 500 million elements (8 GB of total footprint), 5 billion elements (80 GB), and 50 billion elements (800 GB).

Algorithm	500M	5G	50G
Single/Serial	28.51	898.07	-
Single/Parallel	4.12	124.84	-
Naïve/VCPU	0.37	11.10	497.28
Naïve/Core	0.58	17.40	716.88
Naïve/Node	2.91	97.78	3,920.20
MPSF no-resc.	0.40	12.01	362.81
MPSF	0.41	12.59	384.97

Finally, the results for WAVG on Scenario A are summarized in Table 31. For vectors of size 500M and 5G the core-based and VCPU-based naive approaches lead to relatively good distributions, even outperforming MPSF for 500M. However, for larger data sizes the I/O component becomes relevant and only MPSF is able to attain to the natural increase in number of operations (around three times the increase in data size). Compared to the best result obtained via naive approaches, MPSF provides a performance gain of 27.0%, with a speedup factor of 1.37x.

4.5.2 Scenario B: Hybrid Cloud (Gauss and Vulcan)

Scenario B combines the resources from Gauss, consolidated as a private cloud, and the resources from Vulcan, consolidated as a public cloud. In addition, Vulcan also supports non-cloud-specific operations such as batch jobs, affecting the status of its resources and leading to more noticeable performance fluctuations. For this scenario MPSF was configured to either allocate one container per VCPU or one container per core, depending on the benefit of running tasks on memory, without having to use disk I/O. The benchmark selected for the experiments was GEMM. The input is located in the Gauss cluster, and the results must be stored there too.

In other words, if the Vulcan nodes are used, the results must be sent back to Gauss. Table 32 shows the results comparing GEMM for scenario A and B.

Table 32 - Results (run time in seconds) for GEMM.
Table compares results for 40k matrices running on Scenarios A and B.

Algorithm	A	B
Naïve/Core	128.93	88.26
MPSF no-resc.	122.35	70.68
MPSF	117.41	71.12

The average compute rate measured by the framework during execution of the workflow is around 153.6 MB/s for Gauss and 1,474.8 MB/s for Vulcan. Moreover, the effective bandwidth (also collected by the framework during execution and considering fluctuations) between the clusters is 128.12 MB/s. The data distribution reported by the framework is around 56% for Gauss and 44% for Vulcan. During execution no rescheduling activities were detected, thus the performance difference between the MPSF implementations is due to overall performance variations and the small overhead due to rescheduling. The improvement over the Naive/Core approach is of 19.9%, with a 1.25x speedup. Note that MPSF is not greedy in the sense of trying to use all available containers. Instead, it computes the performance gain of using an extra container versus the total number of containers already allocated. If this improvement is lower than a configured threshold, the container is not used.

4.5.3 Scenario C: Multicloud (Gauss and Mamona)

Scenario C combines a private cloud built on top of the Gauss nodes to another private cloud in a different geographical location using the Mamona nodes. To test this setup the data source was configured in one of the Mamona nodes and the GEMM benchmark was executed. Moreover, a restriction was set in the experiment to force the result of the operation to be sent to a Gauss node. The main objective of this experiment is to verify what sort of scheduling decision MPSF adopts knowing that the amount of data is considerable, the network bandwidth between each cloud is comparatively slow (10 Mbps), but the results must go over the network eventually.

Table 33 - Results (run time in seconds) for GEMM, 40k matrices running the workflow only on Mamona nodes vs. also using the Gauss nodes.

Algorithm	A/Gauss	A/Mamona	C/+Gauss
Naïve/Core	128.93	7,098.21	5,443.31
MPSF no-resc.	122.35	6,172.35	3,601.54
MPSF	117.41	5,075.12	3,330.68

Table 33 shows the results of running GEMM with 40k matrices. The new run times are much longer as the results must be sent over the network from Mamona to Gauss. To simplify the tests, only the Naive/Core approach was considered in the comparisons. The Xeon E3-1230 provides slightly better performance than the X5687 and sending result fragments instead on input fragments halves the amount of data to be sent over the network. Executing the workflow using Mamona nodes only and using the Naive/Core approach leads to a total run time of around 2h – 1h20min is spent just to send the results over the network to Gauss. On the other hand, the results for MPSF indicate an improvement of 28.5% (speedup of 1.41x), as the framework handles the performance variations before and during the makespan.

For Scenario C the Naive/Core approach simply distributes the load across all containers (one container per core) without considering the I/O cost of sending the data over the network. The loads are distributed using a basic weighted average considering the processing power of each node and system. This leads to a total run time of around 1h30min, an improvement of 23.3% over the A/Mamona run time. However, using MPSF with the rescheduling capabilities leads to a run time of around 55min, 38.8% of improvement over the Naive/Core approach and 7.5% over MPSF without rescheduling. Moreover, the utilization of MPSF perfectly balances the amount of time spent on Mamona and Gauss sides – controlling this imbalance is essential to minimize the makespan.

4.6 Analysis

MPSF consistently delivered better performance over naive approaches that try to greedily explore the available resources. Internally, MPSF also provides rescheduling capabilities that are particularly useful for workflows with long duration (pipeline longer than one hour). Based on high-level workflow descriptions such as the APEX Workflows (APEX, 2016), this is usually the case for scientific applications. In fact, for the majority of the APEX workflows the wall time of a pipeline is longer than 50 hours. Figure 54 shows a summary of the experimental results showing the relative run time of MPSF compared to Naive/Core.

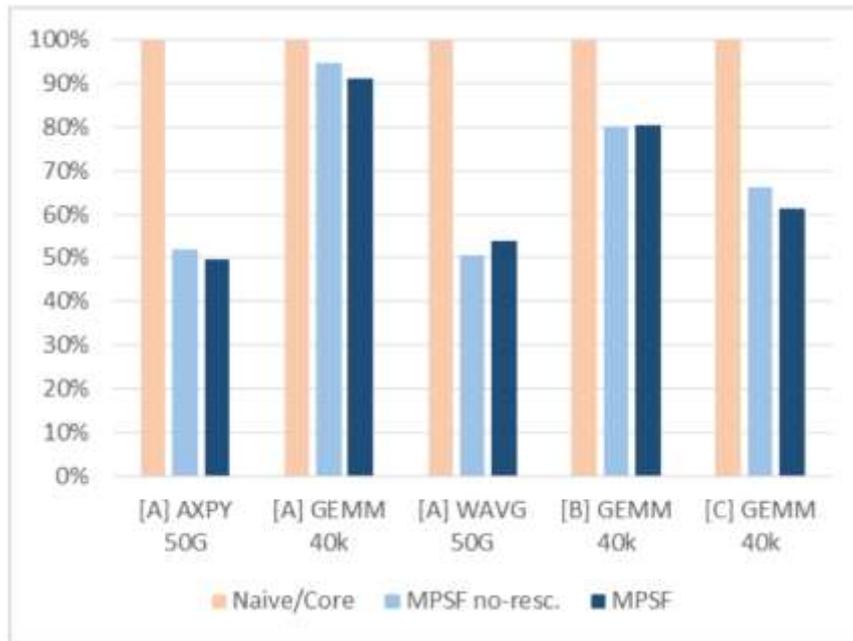


Figure 54 - Summary of experimental results showing relative run time of MPSF compared to the Naive/Core approach. Figure shows scenarios (A, B, C), benchmark (AXPY, GEMM, WAVG), and problem size.

MPSF performs well especially for workflows wherein the data distribution constitutes an important component in the total time. For AXPY and WAVG MPSF shows a relative run time of around 50% compared to Naive/Core. Moreover, the experiments for scenarios B and C show that a scheduling algorithm that considers the time to transfer data, whether input or results, leads to much better performance. Combined to the proactive rescheduling and continuous system monitoring, MPSF delivers consistent results to optimize resource utilization across cloud systems.

Regarding implementation, MPSF also provides mechanisms to support alternative implementations of the same workflow. Figure 55 presents an implementation of GEMM that only sends matrix C after merging the results from $A \times B$. This implementation can be particularly efficient for Scenarios B and C, as the cost of sending data over the network is too high. The selection of the best mechanism is automatically done by MPSF, as long as the

alternatives are provided to the framework and the inputs and expected outputs are similar. Moreover, the algorithm changes must be reflected in the workflow characterization – it is not enough just to change the algorithm, it is also necessary to transmit these changes to MPSF.

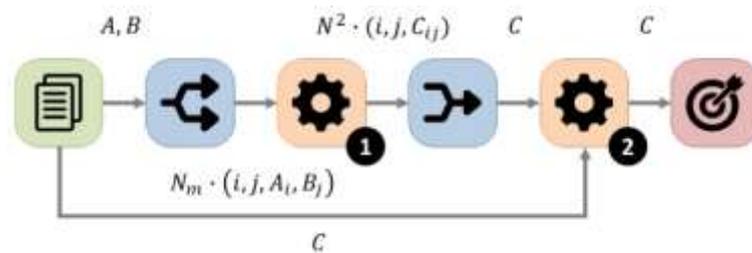


Figure 55 - Alternative workflow description for GEMM

Another example of alternative implementation is observed in Figure 56. This implementation of WAVG specifies two mergers that can be distributed across the network. For Scenario B, for instance, the first merger component could be executed on each cloud (one partial merger for Gauss and another for Vulcan). Then, the second merger receives the partial averages avg_p and sum of weights ws_p and computes the final average.

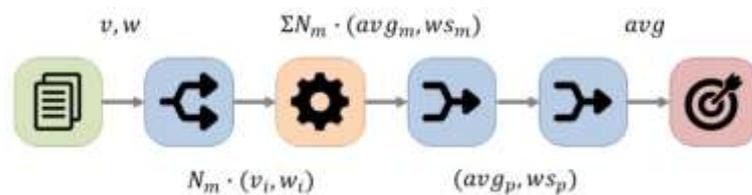


Figure 56 - Alternative workflow description for WAVG

4.7 Chapter Considerations

This chapter presented the experimental results obtained for the utilization of MPSF to manage resource and task execution for different cloud scenarios. Three scenarios were considered in this investigation: one comprising a private cloud (scenario A), one with a hybrid cloud (scenario B), and a multicloud setup (scenario C). These scenarios were used to execute three benchmarks selected due to their importance as kernels to implement more complex workflows. These benchmarks were scaled vector addition, general matrix to matrix multiplication, and weighted average.

Several methods were compared to MPSF in terms of scheduling and allocation, including greedy methods that tried to use all available resources with different granularity levels. The experiments showed that MPSF is able to deliver more consistent and better performing results than the other methods, especially for workflows with long-duration, executed in complex infrastructures, and with data-intensive phases and tasks. The fact that the makespan could be reduced to 80% and even to 60% of the original makespan considering the best greedy approach shows that simply distributing tasks and load to all available resources is not an optimal technique nor it leads to best makespan. Being able to efficiently distribute the load based on workflow and resource properties led to significant performance improvement. In contrast, the overhead imposed by the framework and by the rescheduling activities affects performance and leads to worse performance than greedy approaches for small data sets or for problems that require much more raw computation than data transfers. Matrix multiplication is an example that requires a lot of computation per byte transferred, leading to lower performance gains compared to the other problems (vector addition and weighted average) that show much higher performance gains.

5 FINAL CONSIDERATIONS

MPSF (Multi-Phase Proactive Scheduling Framework), the cloud resource management framework proposed in this work, defines a set of components to read and continuously monitor the resources of systems, and then distribute the workload based on a high-level description of the systems and the characterization of the workflow. The framework is able to predict this workload by calculating the optimal run time based on the distribution of the input data to the containers configured across the systems. The framework also provides components to reschedule a workflow based on the performance fluctuations of the system. Finally, the framework is able to consider the I/O time component, such as data transfers among containers and between different clouds, leading to a better distribution of this initial load and thus balancing the execution of the different fragments throughout the clouds. This final chapter provides an overview of the analyses and proposals presented throughout this work. Section 5.1 presents an overview and summary of the analysis of related work and identification of challenges and gaps to be addressed by cloud resource management solutions. Section 5.2 presents an overview of the proposed framework and summarizes the contributions and innovations provided by this proposal. Section 5.3 presents a recapitulation of the experimental results obtained by implementing and testing MPSF. Section 5.4 presents the limitations of MPSF, focusing both on aspects related to the design of the solution and also related to the experimental results. Finally, Section 5.5 presents future directions and work regarding MPSF and cloud resource management.

5.1 Related Work and Challenges

The study presented in Chapter 2 covered a broad range of works in the area of cloud resource management, also comprising other distributed network computing environments prior to the emerging of cloud computing. The study started with the analysis of existing definitions for resource management. In this sense, three works were selected for further analysis and to create a consolidated definition to be used throughout this work. The first work was (SINGH; CHANA, 2015b), that organizes resource management under three main aspects: provisioning, which is the selection of resources to execute a task; scheduling, which is the implementation of this selection by assigning resources to start the execution of the task; and monitoring, which is the continuous assessment of resource utilization and task execution. The second work analyzed was (JENNINGS; STADLER, 2015), which as further subdivides the resource management activities into profiling, pricing, and provisioning for the initial task of resource selection, and estimation and scheduling for the execution of the task. Finally, the last work considered in this analysis was (MANVI; SHYAM, 2014), which also further subdivides the resource management activities considering actions related to resource discovery, modeling, brokering, and mapping. This analysis identified that the common point among these definitions is the aspect of managing the life cycle of resources and their association to the execution of tasks. The ultimate purpose of resource management is to enable task execution and optimize infrastructural efficiency based on a set of specified objectives, such as makespan, cost, and energy consumption. The definition adopted for this work is based on (SINGH; CHANA, 2015b), which encompasses all aspects presented by the other authors (as analyzed in Table 3) while simplifying the overall model to a minimum of categories.

The next step of the analysis focused on the identification of related work and existing surveys in the field of resource management. This led to identification of over 110 references covering a wide range of scheduling objectives and approaches. Table 6 summarizes the findings by organizing the works using the taxonomy proposed in Section 2.2.3.5. The taxonomy focuses on the challenges to be addressed by MPSF, including data-intensive workflows, hybrid and multicloud scenarios, performance fluctuations and the effects of multitenancy (which are related to the ability of the resource management solution to dynamically cope with these fluctuations), and reliability. The proposed taxonomy also encompasses other aspects which are typically addressed by existing solutions, such as the time to complete the tasks (makespan), deadlines, cost, and energy. This study revealed that 82 works (73%) cover at least one of these aspects. Makespan is addressed by almost 40% of the works, and a similar proportion is observed for cost-driven solutions. On the other hand, data-intensive workflows and hybrid or multicloud scenarios are addressed by less than 10% of the works investigated. In parallel, almost 60% of the works cover some aspect of workflow management, but only 7% address both workflows and some form of dynamic resource management that would be able to cope with performance fluctuations in the system. In addition, only 2% (two works) cover both workflows and hybrid or multicloud scenarios. Finally, none of the works cover workflows, data-intensive loads, reliability aspects, and dynamic mechanisms in an integrated way, which is the main objective of MPSF.

These observations regarding the lack of existing solutions covering the aspects which represent the main challenges addressed by MPSF led to identification of gaps presented in Section 2.3 and to the problem definition presented in Section 2.4. The results of this analysis combined to the requirements identified for future workloads such as (APEX, 2016) led to the conclusion that modern solutions aiming at providing resource management for large-scale

deployments and to execute large-scale problems must provide mechanisms to address data movement in massive scale while adequately distributing resources to tasks, adjusting this distribution depending on the fluctuations observed in the system.

5.2 Proposed Framework and Contributions

Based on the challenges initially set to be addressed by MPSF and the results of the analysis of existing works, the solution requirements for MPSF and associated features were defined as presented in Section 3.1. The requirements were organized in six components: a workflow model to adequately describe the properties of workflows, including their tasks and connections; a resource model able to describe and organized resources from multiple cloud environments; a performance model able to capture the properties of data-intensive workflows and distribute the tasks accordingly; a reliability model based on measurable metrics to mitigate the effects of potential failures in the system; a framework with well-defined management phases associated to the life cycle of resources and tasks; and an architecture able to integrate all these components to enable the implementation of the solution.

Section 3.3 presented the proposed performance models to distribute resources to tasks for further execution. The section evolves from a simple model based on a weighted average considering transfer and processing capabilities of the nodes without considering I/O contention effects from multiple concurrent accesses to data. This led to poor performance and high disparity between the expected behavior and the observed results. Based on these results, the model evolves to an enhanced version the **gradually distributes and processes data**, leading to slightly **better results** in terms of performance and **much better accuracy** in terms of predicting the behavior of the execution of programs. Finally, the model evolves to an

augmented version with special mechanisms to distribute data according to the requirements of the workflow phases being executed.

Another contribution of the performance model is the distinction between **shared and chunk data**, allowing the distribution of data with much better performance than multiple direct transfers from data sources to worker nodes and containers (logarithmic variation versus linear from traditional solutions). To complement the functionalities of the final performance model, MPSF also provides a **probabilistic model** to cope with performance fluctuations based on learning techniques that gradually assess the system and determine the load level of nodes in order to calculate correction factors to adjust the load distribution. The learning technique is based on a **Bayesian system with multiple hypotheses and multiple evidences**. While there are solutions using machine learning and even Bayesian techniques (typically Bayesian networks) to augment resource management mechanisms, a thorough search of relevant literature yielded no prior work using a Bayesian system based on multiple hypotheses and multiple evidences to calculate correction factors for load distribution.

Regarding the appendices, Appendix B presented the proposal for workflow characterization. The proposal revolves around the definition of **five primitives** used to describe workflow phases and their interactions. While these primitives already are present in other works such as (BHARATHI et al., 2008) and (BPMN, 2016), the definition of mechanisms for branching, multiple execution paths, nesting, and workflow composition using this small set of primitives is innovative in the sense of **minimizing the overhead** of absorbing the concepts of a new programming paradigm to then adapt existing codes and applications. The **mapping** between the primitives and their actual implementation (i.e., programs and functions that implement and execute the functionalities represented by each primitive) is

transparent and does not require major modification in existing codes. The framework is able to **integrate to existing code**, including code written for distributed computing frameworks such as MPI, OpenMP, and CUDA. Another contribution of this proposal is the definition of **cost models** associated to each workflow phase and transition which can be defined using different strategies, such as fixed costs, costs based on complexity notation (i.e., varying according to the amount of data processed), and even an **automated process** to determine these costs. The attribution of costs to phases and transitions is not novel, but being able to automatically analyze and determine these costs is an important contribution of this work to facilitate the process of converting existing applications to be used in conjunction with the framework.

Appendix C presented the proposal for resource characterization based on a generic model for cloud infrastructures with the definition of a **hierarchical structure** composed by containers, nodes, and clouds (allowing the integration of multiple clouds). Two extra hierarchical levels were proposed: **aggregates**, allowing the combination of resources from multiple nodes, and the **execution environment**, which encompasses the resources of multiple clouds. These abstractions are required to implement comprehensive resource management mechanisms able to allocate and execute tasks considering all resources available provided by the cloud infrastructures managed by MPSF. Moreover, an additional abstraction named **execution space** was proposed to designate groups of containers (which can be implemented by any method of compartmentalization, such as virtual machines and Docker containers) that share the same set of goals and requirements, as well as workflows and tasks to execute. A process named **segmentation** is responsible for coping with the multiple hierarchical levels and assigning tasks of a workflow to these different levels, distributing split and merge operations across the nodes and containers.

Appendix D presented the proposed model for reliability, which is based on **measurable properties** of the system such as Failures In Time (FIT) and Mean Time Between Failures (MTBF). Based on the probability of failure of a node MPSF defines **five policy levels** to handle these potential failures. The first level is applied to nodes with little to no probability of failure. The second level is used to avoid scheduling mission critical and expensive tasks on specific nodes. The third level is applied to nodes for scheduling only cheap tasks (in terms of time) and auxiliary functionalities only. The fourth level is used to designate nodes that should only execute auxiliary tasks, such as ones related to recovery-related mechanisms and relaying information. Finally, the fifth level determines that the node must not be used, typically assigned to nodes with over 95% chance of failure.

Appendix E presented the scheduling framework focusing on the scheduling phases and mechanisms used to control the process of scheduling and distributing resources to tasks. **Four phases** are used to describe the resource allocation life cycle defined by MPSF: preallocation, which is related to the reservation of resources; allocation, which is related to the execution of tasks and workflows; reallocation, which is controlled by **triggers and thresholds**; and deallocation, which is the termination of a task. Triggers and thresholds define when a scheduling operation should be started and when it should be deployed. Triggers can be defined based on elapsed time, resource utilization, and workflow phases executed. In parallel, thresholds can be defined as absolute values in terms of cost-benefit of deploying a new resource distribution, or based on dynamic levels. These levels can be set via configuration or set to a default value and then automatically adjusted via a **positive-feedback** mechanism based on counters and correct assumptions.

Appendix F presented the solution architecture of MPSF, the last component of the solution and responsible for integrating all pieces. The architecture defines elements to interact with client applications, components to monitor the environment, to discover new resources, to characterize those resources, to define and manage execution spaces, and to characterize workflows. Moreover, the architecture also defines reallocation components related to triggers and thresholds. Finally, MPSF defines components for task handling and execution, as well as modules to interact with the performance and reliability models, as well as to assess the requirements in terms of workflows and security.

Table 34 presents a summary of the contributions of this work focusing on the components presented in Chapter 3 and complementary appendices.

Table 34 – Summary of Contributions

Topic	Contribution	Description
Workflow Characterization	Simplified set of primitives with comprehensive support to complex features	MPSF proposes a simple set of five primitives to describe workflows. This facilitates the understanding of these primitives by developers and still supports the implementation of complex features such as branching, nesting, and workflow composition.
Workflow Characterization	Transparent integration to existing applications	MPSF mapping features allow direct integration between the workflow characterization and the implementation of each phase as a singular executable component.
Workflow Characterization	Transparent integration to existing distributed computing frameworks	MPSF provides seamless integration to existing distributed frameworks by treating programs based on these frameworks as executable components and then applying the features of transparent integration of existing applications.
Workflow Characterization	Automated workflow analysis and cost description based on data	MPSF provides mechanisms to automate the process of defining costs for phases and transitions based on the amount of data to be transferred and processed.
Resource Characterization	Aggregates, execution environment, and execution spaces	MPSF defines a hierarchical structure to organize resources from multiple nodes and multiple clouds. Moreover, execution spaces are used to define a group of resources with shared goals and requirements, as well as tasks.

Table 34 – Summary of Contributions (cont. I)

Topic	Contribution	Description
Resource Characterization	Resource segmentation and task distribution based on resource hierarchy	MPSF defines a segmentation process used to leverage the hierarchical organization of resources to execute workflows by assigning tasks in order to distribute them according to a specific hierarchical layer.
Performance Model	Gradual data distribution with minimized contention effects and higher performance prediction accuracy	Most if not all existing resource management solutions adopt data distribution based on multiple concurrent accesses to data. This leads to severe data contention effects when applied to practical scenarios. MPSF is based on gradual data distribution that yields better performance results and leads to much better accuracy to predict the duration of workflow phases and transitions.
Performance Model	Shared and chunk data for optimized data distribution	MPSF defines the concept of shared and chunk data to optimize data distribution. Shared data is distributed similar to the spread of a virus using selective neighbor pattern that leads to a logarithmic cost versus the typical linear cost of existing solutions.
Performance Model	Pipelining	MPSF uses a mechanism based on pipelining to improve resource utilization regarding idle time due to the gradual distribution of data.
Performance Model	Probabilistic model based on a Bayesian system with multiple hypotheses and multiple evidences	MPSF defines a Bayesian system to calculate correction factors to improve data distribution and mitigate the effects of performance fluctuations and imperfections in the predictions.
Reliability Model	Probabilistic model based on measurable properties	MPSF defines a probabilistic model based on properties such as FIT and MTBF
Reliability Model	Policy levels for handling failure probabilities	MPSF uses the probability levels of failure to assign policies to nodes and therefore determine the types of tasks the node should or should not execute.
Scheduling Framework	Triggers and thresholds and positive-feedback mechanism	Scheduling and allocation operations are first executed when a task must be deployed and then re-executed during the reallocation phase. For the reallocation phase these operations are controlled by triggers and thresholds that determine when to calculate a new distribution and when to deploy this new distribution. These levels can be set or left for MPSF to dynamically adjust them via a positive-feedback mechanism based on counters.

MPSF contributions focus on how to describe workflows, how to describe resources and cloud infrastructures, and how to deliver these resources for workflow execution while

improving performance and minimizing the effects of I/O contention, performance fluctuations, and potential failures.

Papers published include:

- GONZALEZ, N. M.; GOYA, W. A.; PEREIRA, R. F.; LANGONA, K.; SILVA, E. A.; CARVALHO, T. C. M. B.; MIERS, C. C.; MANGS, J-E.; SEFIDCON, A. **Fog computing: Data Analytics and Cloud Distributed Processing on Network Edges**. *35th International Conference of the Chilean Computer Science Society (SCCC '16)*. Valparaiso, Chile. 2016.
- GONZALEZ, N. M.; CARVALHO, T. C. M. B.; MIERS, C. C. **Multi-Phase Proactive Cloud Scheduling Framework Based on High Level Workflow and Resource Characterization**. *IEEE 15th International Symposium on Network Computing and Applications (NCA)*. 2016.

Paper submitted:

- GONZALEZ, N. M.; CARVALHO, T. C. M. B.; MIERS, C. C. **Cloud Resource Management: Towards Efficient Execution of Large-Scale Scientific Applications and Workflows on Complex Infrastructures**. *Journal of Cloud Computing: Advances, Systems, and Applications*. 2016-2017.

5.3 Experimental Results

MPSF was compared to greedy approaches that represent a wide class of solutions that focus on the time to complete the tasks (makespan) and deadline. Experimental results indicate run time improvement of over 2x compared to these greedy approaches, especially for larger

input data sets and longer computations. Scenarios B and C show that MPSF is also able to adequately distribute the load across containers from different clouds and different geographical locations, taking as input the time to send this information across the network. Moreover, the high-level description of workflows and systems is not restricted to just specifying the compute and transfer cost of each workflow phase. For instance, this characterization allows MPSF to determine the memory footprint of executing the workflow for a specific input.

5.4 Limitations

Regarding limitations, the automatic detection and analysis of different algorithm implementations (or binaries) and the mechanism to choose the best alternative require these alternatives to be represented in the workflow description. In addition, the rescheduling capabilities are extremely useful for long duration workflows, but for short computations the inherent overhead might degrade performance, although by a small factor (less than 5%).

Another clear limitation of MPSF is the need to express existing applications using the notation defined by the five primitives used by the framework. Minimizing the number of primitives and simplifying their integration to existing code represent two steps to simplify this process, but it still requires some degree of porting.

In addition, the experimental results also reveal that using MPSF for workflows wherein computation is much more expressive than data movement (e.g., matrix multiplication) does not yield such a drastic performance improvement as the one observed for the other cases (i.e., vector addition and weighted average). On the other hand, it is expected that the amount of data to be handled for current and future workflows is bound to increase substantially (APEX, 2016),

which renders MPSF as an important tool to manage resources for execution of large-scale applications.

MPSF provides a limited degree of coverage of security issues. There is no support for secure execution of workflows and data movement, apart from the definition of execution spaces with requirements related to security (e.g., only execute a certain piece of code in a specific execution space).

Regarding the performance models, MPSF still lacks further pipelining mechanisms to better support the utilization of resources considering the gradual distribution of data. Finally, MPSF focuses on the optimization of workflow execution for a particular set of resources (i.e., execution space). The scheduling of multiple workflows on the cloud infrastructure while optimizing resource utilization is directly not covered – it is assumed that a subset of resources is allocated to execute a workflow, and based on this selection the framework is used to distribute the load. This is the typical case for allocations of large-scale scientific clusters, wherein a chunk of resources (and time) is associated to a particular workflow. Nevertheless, the ability to automatically define this allocation is highly desirable.

5.5 Future Directions and Work

The first and main future direction based on MPSF or any other resource management solution is to define resource management mechanisms that are able to distribute tasks not only from a single workflow, but from multiple workflows onto the same infrastructure of nodes and resources. As mentioned previously, MPSF assumes that a subset of nodes is allocated to execute a workflow, and then the framework is used to distribute resources to execute the tasks of this workflow in order to minimize makespan. A fundamental future work in this sense is to expand MPSF to be able to allocate resources for multiple workflows without having a pre-definition of execution spaces for each workflow. An initial approach would comprise the adaptation of existing load balancing solutions to define the execution spaces and then use MPSF to optimize the execution of each workflow, internally. A further approach would be to use the same mechanisms of gradual distribution to schedule multiple workflows, abstracting these workflows as tasks of a greater workflow (the set of workflows to be executed on the infrastructure).

Another future work or research direction is to integrate the performance and reliability models proposed in this work with other machine learning based solutions to augment the resource distribution. MPSF is based on a Bayesian system to calculate correction factors to cope with performance fluctuations, but other machine learning strategies could be used to improve this distribution.

Finally, MPSF does not address security aspects beyond defining safe zones based on security requirements assigned to execution spaces. Future work in this sense would comprise the definition of actual security mechanisms to protect data in transit and data being processed in the nodes, using some form of encryption while maintaining acceptable performance levels.

REFERENCES

- ABRISHAMI, S., NAGHIBZADEH, M., EPEMA, D. H. **Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds**. *Future Generation Computer Systems*, 29(1), 158-169. 2013.
- ANAND, L., GHOSE, D., MANI, V. **ELISA: an estimated load information scheduling algorithm for distributed computing systems**. *Computers & Mathematics with Applications*, 37(8), 57-85. 1999.
- ANDRADE, N., CIRNE, W., BRASILEIRO, F., ROISENBERG, P. **OurGrid: An approach to easily assemble grids with equitable resource sharing**. In *Workshop on Job Scheduling Strategies for Parallel Processing* (pp. 61-86). Springer Berlin Heidelberg. 2003.
- APEX Workflows**. LANL, NERSC, SNL. SAND2016-2371 O; LA-UR-15-29113. March, 2016. Available at: <http://www.nersc.gov/assets/apex-workflows-v2.pdf>. Last access: September 11, 2016.
- ARABNEJAD, H., BARBOSA, J. G. **List scheduling algorithm for heterogeneous systems by an optimistic cost table**. *IEEE Transactions on Parallel and Distributed Systems*, 25(3), 682-694. 2014.
- ARABNEJAD, H., & BARBOSA, J. G. **A budget constrained scheduling algorithm for workflow applications**. *Journal of Grid Computing*, 12(4), 665-679. 2014a.
- ARABNEJAD, V.; BUBENDORFER, K. **Cost Effective and Deadline Constrained Scientific Workflow Scheduling for Commercial Clouds**. In *Network Computing and Applications (NCA)*, 2015 IEEE 14th International Symposium on (pp. 106-113). IEEE. 2015.
- ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I.; ZAHARIA, M. **Above the clouds: A Berkeley view of cloud computing**. *Electrical Engineering and Computer Sciences, University of California, Berkeley, Technical Report No. UCB/EECS-2009-28*, 2009.
- ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I.; ZAHARIA, M. **A view of cloud computing**. *Communications of the ACM*, 53(4), 50-58, 2010.

BACKBLAZE. **Hard Drive Stats for Q2 2016.**

Available at: <https://www.backblaze.com/blog/hard-drive-failure-rates-q2-2016/>. Last access: October, 2016.

BALA, A.; CHANA, I. **A survey of various workflow scheduling algorithms in cloud environment.** *In 2nd National Conference on Information and Communication Technology (NCICT)* (pp. 26-30). 2011.

BELLAVISTA, P., CORRADI, A., KOTOULAS, S., REALE, A. **Adaptive Fault-Tolerance for Dynamic Resource Provisioning in Distributed Stream Processing Systems.** *In EDBT* (pp. 85-96). 2014.

BERL, A., GELENBE, E., DI GIROLAMO, M., GIULIANI, G., DE MEER, H., DANG, M. Q., PENTIKOUSIS, K. **Energy-efficient cloud computing.** *The computer journal*, 53(7), 1045-1051. 2010.

BESSAI, K., YUCEF, S., OULAMARA, A., GODART, C., & NURCAN, S. **Bi-criteria workflow tasks allocation and scheduling in Cloud computing environments.** *In Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on* (pp. 638-645). IEEE. 2012.

BHARATHI, S., CHERVENAK, A. **Data staging strategies and their impact on the execution of scientific workflows.** *In Proceedings of the second international workshop on Data-aware distributed computing* (p. 5). ACM. 2009.

BHARATHI, S.; CHERVENAK, A.; DEELMAN, E.; MEHTA, G.; SU, M. H.; VAHI, K. **Characterization of scientific workflows.** *In 2008 Third Workshop on Workflows in Support of Large-Scale Science* (pp. 1-10). IEEE, 2008.

BILGAIYAN, S., SAGNIKA, S., DAS, M. **Workflow scheduling in cloud computing environment using Cat Swarm Optimization.** *In Advance Computing Conference (IACC), 2014 IEEE International* (pp. 680-685). IEEE. 2014.

BITTENCOURT, L. F., MADEIRA, E. R. M. **HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds.** *Journal of Internet Services and Applications*, 2(3), 207-227. 2011.

BITTENCOURT, L. F.; MADEIRA, E. R.; DA FONSECA, N. L. **Scheduling in hybrid clouds.** *IEEE Communications Magazine*, 50(9), 42-47. 2012.

BPMN. Available at: <http://www.bpmn.org/>. Last access: September 2016.

BUTT, A. R., ZHANG, R., HU, Y. C. **A self-organizing flock of condors.** *In Proceedings of the 2003 ACM/IEEE conference on Supercomputing* (p. 42). ACM. 2003.

BUY YA, R.; YEO, C. S.; VENUGOPAL, S. **Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities.** *In High Performance Computing and Communications, 2008. HPCCC'08. 10th IEEE International Conference on* (pp. 5-13). IEEE, 2008.

BUY YA, R.; YEO, C. S.; VENUGOPAL, S.; BROBERG, J.; BRANDIC, I. **Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility.** *Future Generation computer systems*, 25(6), 599-616, 2009.

BYUN, E. K., KEE, Y. S., KIM, J. S., MAENG, S. **Cost optimized provisioning of elastic resources for application workflows.** *Future Generation Computer Systems*, 27(8), 1011-1026. 2011.

CALHEIROS, R. N., RANJAN, R., BELOGLAZOV, A., DE ROSE, C. A., BUY YA, R. **CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms.** *Software: Practice and Experience*, 41(1), 23-50. 2011.

CALHEIROS, R. N., VECCHIOLA, C., KARUNAMOORTHY, D., BUY YA, R. **The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds.** *Future Generation Computer Systems*, 28(6), 861-870. 2012.

CANON, L. C., JEANNOT, E., SAKELLARIOU, R., ZHENG, W. **Comparative evaluation of the robustness of dag scheduling heuristics.** *In Grid Computing* (pp. 73-84). Springer US. 2008.

CHANA, I.; SINGH, S. **Quality of service and service level agreements for cloud environments: Issues and challenges.** *In Cloud Computing* (pp. 51-72). Springer International Publishing. 2014.

CHARD, R.; CHARD, K.; BUBENDORFER, K.; LACINSKI, L.; MADDURI, R.; FOSTER, I. **Cost-aware elastic cloud provisioning for scientific workloads.** *In 2015 IEEE 8th International Conference on Cloud Computing* (pp. 971-974). IEEE. 2015.

CHEN, W. N., ZHANG, J. **An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements.** *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 39(1), 29-43. 2009.

CHIEU, T. C.; MOHINDRA, A.; KARVE, A. A.; SEGAL, A. **Dynamic scaling of web applications in a virtualized cloud computing environment.** *In e-Business Engineering, 2009. ICEBE'09. IEEE International Conference on* (pp. 281-286). IEEE, 2009.

CORDASCO, G., MALEWICZ, G., ROSENBERG, A. L. **Extending IC-scheduling via the Sweep algorithm.** *Journal of Parallel and Distributed Computing*, 70(3), 201-211. 2010.

DASTJERDI, A. V., BUYYA, R. **An autonomous reliability-aware negotiation strategy for cloud computing environments.** *In Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on* (pp. 284-291). IEEE. 2012.

DAVAL-FREROT, C., LACROIX, M., GUYENNET, H. **Federation of resource traders in objects-oriented distributed systems.** *In Proceedings of the International Conference on Parallel Computing in Electrical Engineering* (p. 84). IEEE Computer Society. 2000.

DEELMAN, E. **Grids and clouds: making workflow applications work in heterogeneous distributed environments.** *International Journal of High Performance Computing Applications*, 24(3), 284-298, 2010.

DEELMAN, E.; SINGH, G.; LIVNY, M.; BERRIMAN, B.; GOOD, J. **The cost of doing science on the cloud: the Montage example.** *In Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (p. 50). IEEE Press. 2008.

DEMCHENKO, Y.; BLANCHET, C.; LOOMIS, C.; BRANCHAT, R.; SLAWIK, M.; ZILCI, I.; BEDRI, M.; GIBRAT, J-F.; LODYGENSKY, O.; ZIVKOVIC, M.; DE LAAT, C. **CYCLONE: A Platform for Data Intensive Scientific Applications in Heterogeneous Multi-cloud/Multi-provider Environment.** *In 2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)* (pp. 154-159). IEEE, 2016.

DIAS, M., BUYYA, R., VENUGOPAL, S. **InterGrid: A case for internetworking islands of Grids.** *Concurrency and Computation: Practice and Experience*, 20(8), 997-1024. 2008.

DILLON, T.; WU, C.; & CHANG, E. **Cloud computing: issues and challenges.** *In 2010 24th IEEE international conference on advanced information networking and applications* (pp. 27-33). IEEE, 2010.

DOGAN, A., OZGUNER, F. **Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing.** *IEEE Transactions on Parallel and Distributed Systems*, 13(3), 308-323. 2002.

DOGAN, A., OZGUNER, F. **Biobjective scheduling algorithms for execution time–reliability trade-off in heterogeneous computing systems.** *The Computer Journal*, 48(3), 300-314. 2005.

DONGARRA, J. J., JEANNOT, E., SAULE, E., SHI, Z. **Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems.** *In Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures* (pp. 280-288). ACM. 2007.

EL-REWINI, H., LEWIS, T. G. **Scheduling parallel program tasks onto arbitrary target machines.** *Journal of parallel and Distributed Computing*, 9(2), 138-153. 1990.

ELLERMAN, P. **Calculating Reliability using FIT & MTTF: Arrhenius HTOL Model.** Microsemi Corp., 2012.

FARD, H. M., PRODAN, R., BARRIONUEVO, J. J. D., FAHRINGER, T. **A multi-objective approach for workflow scheduling in heterogeneous environments.** *In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2012)* (pp. 300-309). IEEE Computer Society. 2012.

FARD, H. M.; PRODAN, R.; FAHRINGER, T. **A truthful dynamic workflow scheduling mechanism for commercial multicloud environments.** *IEEE Transactions on Parallel and Distributed systems*, 24(6), 1203-1212. 2013.

FELTER, W.; FERREIRA, A.; RAJAMONY, R.; RUBIO, J. **An updated performance comparison of virtual machines and Linux containers.** *In Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On* (pp. 171-172). IEEE, 2015.

FÖLLING, A., GRIMME, C., LEPPING, J., PAPASPYROU, A. **Decentralized grid scheduling with evolutionary fuzzy systems.** *In Workshop on Job Scheduling Strategies for Parallel Processing* (pp. 16-36). Springer Berlin Heidelberg. 2009.

FOSTER, I.; ZHAO, Y.; RAICU, I.; & LU, S. **Cloud computing and grid computing 360-degree compared.** *In 2008 Grid Computing Environments Workshop* (pp. 1-10). IEEE, 2008.

FRINCU, M. E., CRACIUN, C. **Multi-objective meta-heuristics for scheduling applications with high availability requirements and cost constraints in multi-cloud environments.** *In Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on* (pp. 267-274). IEEE. 2011.

GAO, Y., WANG, Y., GUPTA, S. K., PEDRAM, M. **An energy and deadline aware resource provisioning, scheduling and optimization framework for cloud systems.** *In Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis* (p. 31). IEEE Press. 2013.

GARG, S. K., BUYYA, R., SIEGEL, H. J. **Time and cost trade-off management for scheduling parallel applications on utility grids.** *Future Generation Computer Systems*, 26(8), 1344-1355. 2010.

GARG, S. K., GOPALAIYENGAR, S. K., BUYYA, R. **SLA-based resource provisioning for heterogeneous workloads in a virtualized cloud datacenter.** *In international conference on Algorithms and architectures for parallel processing* (pp. 371-384). Springer Berlin Heidelberg. 2011.

GONZALEZ, N. M.; MIERS, C. C.; CARVALHO, T. C. M. B. **Multi-Phase Proactive Cloud Scheduling Framework Based on High Level Workflow and Resource Characterization.** *In 15th IEEE International Symposium on Network Computing and Applications (NCA 2016)*. IEEE, 2016.

GREKIOTI, A., SHAKHLEVICH, N. V. **Scheduling bag-of-tasks applications to optimize computation time and cost.** *In International Conference on Parallel Processing and Applied Mathematics* (pp. 3-12). Springer Berlin Heidelberg. 2013.

GREWAL, R. K., PATERIYA, P. K. A rule-based approach for effective resource provisioning in hybrid cloud environment. *In New Paradigms in Internet Computing* (pp. 41-57). Springer Berlin Heidelberg. 2013.

GROZEV, N., BUYYA, R. **Inter- Cloud architectures and application brokering: taxonomy and survey.** *Software: Practice and Experience*, 44(3), 369-390. 2014.

HAKEM, M., BUTELLE, F. **Reliability and scheduling on systems subject to failures.** *In 2007 International Conference on Parallel Processing (ICPP 2007)* (pp. 38-38). IEEE. 2007.

HAYES, B. **Cloud computing.** *Communications of the ACM*, 51(7), 9-11, 2008.

HE, S.; GUO, L.; GUO, Y.; WU, C.; GHANEM, M.; HAN, R. **Elastic application container: A lightweight approach for cloud resource provisioning.** *In 2012 IEEE 26th International Conference on Advanced Information Networking and Applications* (pp. 15-22). IEEE, 2012.

HOFMANN, P.; WOODS, D. **Cloud computing: the limits of public clouds for business applications.** *IEEE Internet Computing*, 14(6), 90-93. 2010.

- HÖNIG, U., SCHIFFMANN, W. **A meta-algorithm for scheduling multiple DAGs in homogeneous system environments.** *In Proceedings of the eighteenth IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'06)*. 2006.
- HUANG, Y., BESSIS, N., NORRINGTON, P., KUONEN, P., HIRSBRUNNER, B. **Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm.** *Future Generation Computer Systems*, 29(1), 402-415. 2013.
- HUSSAIN, H., MALIK, S. U. R., HAMEED, A., KHAN, S. U., BICKLER, G., Min-ALLAH, N., KOLODZIEJ, J. A survey on resource allocation in high performance distributed computing systems. *Parallel Computing*, 39(11), 709-736. 2013.
- HWANG, E., KIM, K. H. Minimizing cost of virtual machines for deadline-constrained MapReduce applications in the cloud. *In 2012 ACM/IEEE 13th International Conference on Grid Computing* (pp. 130-138). IEEE. 2012.
- HWANG, J. J., CHOW, Y. C., ANGER, F. D., LEE, C. Y. **Scheduling precedence graphs in systems with interprocessor communication times.** *SIAM journal on computing*, 18(2), 244-257. 1989.
- HWANG, S., KESSELMAN, C. **Grid workflow: a flexible failure handling framework for the grid.** *In High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on* (pp. 126-137). IEEE. 2003.
- IOSUP, A.; EPEMA, D. **Grid computing workloads.** *IEEE Internet Computing*, 15(2), 19-26, 2011.
- IOSUP, A., TANNENBAUM, T., FARRELLEE, M., EPEMA, D., LIVNY, M. **Inter-operating grids through delegated matchmaking.** *Scientific Programming*, 16(2-3), 233-253. 2008.
- IQBAL, W., DAILEY, M. N., CARRERA, D., JANECEK, P. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Computer Systems*, 27(6), 871-879. 2011.
- IVERSON, M. A. **Hierarchical, competitive scheduling of multiple DAGs in a dynamic heterogeneous environment.** *Distributed Systems Engineering*, 6(3), 112. 1999.
- JENNINGS, B.; STADLER, R. **Resource management in clouds: Survey and research challenges.** *Journal of Network and Systems Management*, 23(3), 567-619. 2015.

- JRAD, F.; TAO, J.; STREIT, A. **A broker-based framework for multi-cloud workflows.** *In Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds* (pp. 61-68). ACM. 2013.
- JUVE, G.; CHERVENAK, A.; DEELMAN, E.; BHARATHI, S.; MEHTA, G.; VAHI, K. **Characterizing and profiling scientific workflows.** *Future Generation Computer Systems*, 29(3), 682-692, 2013.
- KACAMARGA, M. F.; PARDAMEAN, B.; & WIJAYA, H. **Lightweight virtualization in cloud computing for research.** *In International Conference on Soft Computing, Intelligence Systems, and Information Technology* (pp. 439-445). Springer Berlin Heidelberg, 2015.
- KERTESZ, A., KECSKEMETI, G., BRANDIC, I. **Autonomic SLA-aware service virtualization for distributed systems.** *In 2011 19th International EuroMicro Conference on Parallel, Distributed and Network-Based Processing* (pp. 503-510). IEEE. 2011.
- KLUTKE, G. A., KIESSLER, P. C., WORTMAN, M. A. **A critical look at the bathtub curve.** *IEEE Transactions on Reliability*, 52(1), 125-129. 2003.
- KIM, K. H., BUYYA, R., & KIM, J. **Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-enabled Clusters.** *In CCGRID* (Vol. 7, pp. 541-548). 2007.
- KIM, S. J., BROWNE, J. C. A general approach to mapping of parallel computation upon multiprocessor architectures. *In International conference on parallel processing* (Vol. 3, No. 1, p. 8). 1988.
- KOUSIOURIS, G., MENYCHTAS, A., KYRIAZIS, D., GOGOUVITIS, S., VARVARIGOU, T. **Dynamic, behavioral-based estimation of resource provisioning based on high-level application terms in Cloud platforms.** *Future Generation Computer Systems*, 32, 27-40. 2014.
- KRUATRACHUE, B., LEWIS, T. **Grain size determination for parallel processing.** *IEEE software*, 5(1), 23-32. 1988.
- KWOK, Y. K., AHMAD, I. **Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors.** *IEEE transactions on parallel and distributed systems*, 7(5), 506-521. 1996.
- LAI, K., RASMUSSEN, L., ADAR, E., ZHANG, L., HUBERMAN, B. A. **Tycoon: An implementation of a distributed, market-based resource allocation system.** *Multiagent and Grid Systems*, 1(3), 169-182. 2005.

- LEAL, K., HUEDO, E., LLORENTE, I. M. A decentralized model for scheduling independent tasks in federated grids. *Future Generation Computer Systems*, 25(8), 840-852. 2009.
- LEE, Y. C.; SUBRATA, R.; ZOMAYA, A. Y. **On the performance of a dual-objective optimization model for workflow applications on grid platforms.** *IEEE Transactions on Parallel and Distributed Systems*, 20(9), 1273-1284. 2009.
- LEE, Y. C., ZOMAYA, A. Y. **Energy conscious scheduling for distributed computing systems under different operating conditions.** *IEEE Transactions on Parallel and Distributed Systems*, 22(8), 1374-1381. 2011.
- LENK, A.; KLEMS, M.; NIMIS, J.; TAI, S.; SANDHOLM, T. **What's inside the Cloud? An architectural map of the Cloud landscape.** *In Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing* (pp. 23-31). IEEE Computer Society. 2009.
- LI, J., SU, S., CHENG, X., HUANG, Q., ZHANG, Z. **Cost-conscious scheduling for large graph processing in the cloud.** *In High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on* (pp. 808-813). IEEE. 2011.
- LI, X. Y.; ZHOU, L. T.; SHI, Y.; GUO, Y. **A trusted computing environment model in cloud architecture.** *In 2010 International Conference on Machine Learning and Cybernetics* (Vol. 6, pp. 2843-2848). IEEE, 2010.
- LIN, C., LU, S. **Scheduling scientific workflows elastically for cloud computing.** *In Cloud Computing (CLOUD), 2011 IEEE International Conference on* (pp. 746-747). IEEE. 2011.
- LIN, X., WU, C. Q. **On scientific workflow scheduling in clouds under budget constraint.** *In 2013 42nd International Conference on Parallel Processing* (pp. 90-99). IEEE. 2013.
- LITZKOW, M. J., LIVNY, M., MUTKA, M. W. **Condor-a hunter of idle workstations.** *In Distributed Computing Systems, 1988., 8th International Conference on* (pp. 104-111). IEEE. 1988.
- LIU, D.; ZHAO, L. **The research and implementation of cloud computing platform based on Docker.** *In Wavelet Active Media Technology and Information Processing (ICCWAMTIP), 2014 11th International Computer Conference on* (pp. 475-478). IEEE, 2014.
- LIU, K., JIN, H., CHEN, J., LIU, X., YUAN, D., YANG, Y. **A compromised-time-cost scheduling algorithm in SwinDeW-C for instance-intensive cost-constrained workflows on cloud computing platform.** *International Journal of High Performance Computing Applications*. 2010.

MAHESHWARI, K., JUNG, E. S., MENG, J., MOROZOV, V., VISHWANATH, V., KETTIMUTHU, R. **Workflow performance improvement using model-based scheduling over multiple clusters and clouds.** *Future Generation Computer Systems*, 54, 206-218. 2016.

MAJUMDAR, S. **Resource management on clouds and grids: challenges and answers.** *In Proceedings of the 14th Communications and Networking Symposium* (pp. 151-152). Society for Computer Simulation International. 2011.

MALAWSKI, M., JUVE, G., DEELMAN, E., NABRZYSKI, J. **Cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds.** *In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (p. 22). IEEE Computer Society Press. 2012.

MALEWICZ, G., FOSTER, I., ROSENBERG, A. L., WILDE, M. **A tool for prioritizing DAGMan jobs and its evaluation.** *Journal of Grid Computing*, 5(2), 197-212. 2007.

MANVI, S. S.; SHYAM, G. K. **Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey.** *Journal of Network and Computer Applications*, 41, 424-440, 2014.

MAO, M., HUMPHREY, M. **Auto-scaling to minimize cost and meet application deadlines in cloud workflows.** *In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (p. 49). ACM. 2011.

MAO, M., LI, J., HUMPHREY, M. **Cloud auto-scaling with deadline and budget constraints.** *In 2010 11th IEEE/ACM International Conference on Grid Computing* (pp. 41-48). IEEE.

MARINESCU, D. C. **Cloud computing: theory and practice.** Newnes. 2013.

MELL, P.; GRANCE, T. **The NIST definition of cloud computing,** *National Institute of Standards and Technology Special Publication 800-145*, U.S. Department of Commerce, 2011.

MERKEL, D. **Docker: lightweight Linux containers for consistent development and deployment.** *Linux Journal*, 2014(239), 2. 2014.

MEZMAZ, M., MELAB, N., KESSACI, Y., LEE, Y. C., TALBI, E. G., ZOMAYA, A. Y., TUYTTENS, D. **A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems.** *Journal of Parallel and Distributed Computing*, 71(11), 1497-1508. 2011.

- MISHRA, R., RASTOGI, N., ZHU, D., MOSSÉ, D., MELHEM, R. **Energy aware scheduling for distributed real-time systems.** In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International* (pp. 9-pp). IEEE. 2003.
- NURMI, D; WOLSKI, R; GRZEGORCZYK, C; OBERTELLI, G; SOMAN, S; YOUSEFF, L; ZAGORODNOV, D. **The eucalyptus open-source cloud-computing system.** In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on* (pp. 124-131). IEEE, 2009.
- PANDEY, S., WU, L., GURU, S. M., BUYYA, R. **A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments.** In *2010 24th IEEE international conference on advanced information networking and applications* (pp. 400-407). IEEE. 2010.
- PARK, S. M., HUMPHREY, M. **Data throttling for data-intensive workflows.** In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on* (pp. 1-11). IEEE. 2008.
- PARSA, S., ENTEZARI-MALEKI, R. **RASA: A new task scheduling algorithm in grid environment.** *World Applied sciences journal*, 7, 152-160. 2009.
- PHAPHOOM, N.; WANG, X.; ABRAHAMSSON, P. **Foundations and technological landscape of cloud computing.** *ISRN Software Engineering*, 2013.
- POOLA, D., GARG, S. K., BUYYA, R., YANG, Y., RAMAMOHANARAO, K. **Robust scheduling of scientific workflows with deadline and budget constraints in clouds.** In *2014 IEEE 28th International Conference on Advanced Information Networking and Applications* (pp. 858-865). IEEE. 2014.
- POOLA, D., RAMAMOHANARAO, K., BUYYA, R. **Fault-tolerant workflow scheduling using spot instances on clouds.** *Procedia Computer Science*, 29, 523-533. 2014a.
- PRODAN, R., WIECZOREK, M. **Bi-criteria scheduling of scientific grid workflows.** *IEEE Transactions on Automation Science and Engineering*, 7(2), 364-376. 2010.
- PRUHS, K., van STEE, R., & UTHAISOMBUT, P. Speed scaling of tasks with precedence constraints. *Theory of Computing Systems*, 43(1), 67-80. 2008.
- RAMAKRISHNAN, A., SINGH, G., ZHAO, H., DEELMAN, E., SAKELLARIOU, R., VAHI, K., SAMIDI, M. **Scheduling data-intensive workflows onto storage-constrained distributed resources.** In *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)* (pp. 401-409). IEEE. 2007.

REN, K.; WANG, C.; WANG, Q. **Security challenges for the public cloud.** *IEEE Internet Computing*, 16(1), 69. 2012.

RITTINGHOUSE, J. W.; RANSOME, J. F. **Cloud computing: implementation, management, and security.** CRC press. 2016.

RODERO, I., GUIM, F., CORBALAN, J. **Evaluation of coordinated Grid scheduling strategies.** *In High Performance Computing and Communications, 2009. HPCC'09. 11th IEEE International Conference on* (pp. 1-10). IEEE. 2009.

RODERO, I., GUIM, F., CORBALAN, J., FONG, L., SADJADI, S. M. **Grid broker selection strategies using aggregated resource information.** *Future Generation Computer Systems*, 26(1), 72-86. 2010.

RODRIGUEZ, M. A., BUYYA, R. **Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds.** *IEEE Transactions on Cloud Computing*, 2(2), 222-235. 2014.

ROSENBERG, F., CELIKOVIC, P., MICHLMAYR, A., LEITNER, P., DUSTDAR, S. **An end-to-end approach for QoS-aware service composition.** *In Enterprise Distributed Object Computing Conference, 2009. EDOC'09. IEEE International* (pp. 151-160). IEEE. 2009.

SAKELLARIOU, R., ZHAO, H. **A low-cost rescheduling policy for efficient mapping of workflows on grid systems.** *Scientific Programming*, 12(4), 253-262. 2004.

SAKELLARIOU, R., ZHAO, H., TSIAKKOURI, E., DIKAIAKOS, M. D. **Scheduling workflows with budget constraints.** *In Integrated research in GRID computing* (pp. 189-202). Springer US. 2007.

SCHWIEGELSHOHN, U., YAHYAPOUR, R. **Resource allocation and scheduling in metasystems.** *In International Conference on High-Performance Computing and Networking* (pp. 851-860). Springer Berlin Heidelberg. 1999.

SELVARANI, S.; SADHASIVAM, G. S. **Improved cost-based algorithm for task scheduling in cloud computing.** *In Computational intelligence and computing research (ICCIC), 2010 IEEE international conference on* (pp. 1-5). IEEE. 2010.

SENTURK, I. F., BALAKRISHNAN, P., ABU-DOLEH, A., KAYA, K., MALLUHI, Q., ÇATALYÜREK, Ü. V. **A resource provisioning framework for bioinformatics applications in multi-cloud environments.** *Future Generation Computer Systems*. 2016.

SHAH, R., VEERAVALLI, B., MISRA, M. **On the design of adaptive and decentralized load balancing algorithms with load estimation for computational grid environments.** *IEEE Transactions on parallel and distributed systems*, 18(12), 1675-1686. 2007.

SHARIFI, M., SHAHRIVARI, S., SALIMI, H. **PASTA: a power-aware solution to scheduling of precedence-constrained tasks on heterogeneous computing resources.** *Computing*, 95(1), 67-88. 2013.

SHI, Z., JEANNOT, E., DONGARRA, J. J. **Robust task scheduling in non-deterministic heterogeneous computing systems.** *In 2006 IEEE International Conference on Cluster Computing* (pp. 1-10). IEEE. 2006.

SIH, G. C., LEE, E. A. **A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures.** *IEEE transactions on Parallel and Distributed systems*, 4(2), 175-187. 1993.

SIMAO, J., VEIGA, L. **Flexible SLAs in the cloud with a partial utility-driven scheduling architecture.** *In Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on* (Vol. 1, pp. 274-281). IEEE. 2013.

SIMARRO, J. L. L., MORENO-VOZMEDIANO, R., MONTERO, R. S., LLORENTE, I. M. **Dynamic placement of virtual machines for cost optimization in multi-cloud environments.** *In High Performance Computing and Simulation (HPCS), 2011 International Conference on* (pp. 1-7). IEEE. 2011.

SINGH, S.; CHANA, I. **Q-aware: Quality of service based cloud resource provisioning.** *Computers & Electrical Engineering*, 47, 138-160, 2015a.

SINGH, S.; CHANA, I. **Cloud resource provisioning: survey, status and future research directions.** *Knowledge and Information Systems*, 1-65, 2015b.

SINGH, S.; CHANA, I. **A survey on resource scheduling in cloud computing: Issues and challenges.** *Journal of Grid Computing*, 14(2), 217-264, 2016.

SLOMINSKI, A.; MUTHUSAMY, V.; KHALAF, R. **Building a multi-tenant cloud service from legacy code with Docker containers.** *In Cloud Engineering (IC2E), 2015 IEEE International Conference on* (pp. 394-396). IEEE, 2015.

SMANCHAT, S.; VIRIYAPANT, K. **Taxonomies of workflow scheduling problem and techniques in the cloud.** *Future Generation Computer Systems*, 52, 1-12. 2015.

SOTIRIADIS, S., BESSIS, N., ANTONOPOULOS, N. **Towards inter-cloud schedulers: A survey of meta-scheduling approaches.** *In P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2011 International Conference on* (pp. 59-66). IEEE. 2011.

STAVRINIDES, G. L., KARATZA, H. D. **Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques.** *Simulation Modelling Practice and Theory*, 19(1), 540-552. 2011.

SUBRAMANI, V., KETTIMUTHU, R., SRINIVASAN, S., SADAYAPPAN, S. **Distributed job scheduling on computational grids using multiple simultaneous requests.** *In High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on* (pp. 359-366). IEEE. 2002.

SZEPIENIEC, T., BUBAK, M. **Investigation of the DAG eligible jobs maximization algorithm in a grid.** *In Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing* (pp. 340-345). IEEE Computer Society. 2008.

TALUKDER, A. K. M., KIRLEY, M., BUYYA, R. **Multiobjective differential evolution for scheduling workflow applications on global Grids.** *Concurrency and Computation: Practice and Experience*, 21(13), 1742-1756. 2009.

TAYLOR, I. J.; DEELMAN, E.; GANNON, D. B.; SHIELDS, M. **Workflows for e-Science: scientific workflows for grids.** Springer Publishing Company, Incorporated. 2014.

TIAN, F., CHEN, K. **Towards optimal resource provisioning for running MapReduce programs in public clouds.** *In Cloud Computing (CLOUD), 2011 IEEE International Conference on* (pp. 155-162). IEEE. 2011.

TOPCUOGLU, H., HARIRI, S., WU, M. Y. **Performance-effective and low-complexity task scheduling for heterogeneous computing.** *IEEE transactions on parallel and distributed systems*, 13(3), 260-274. 2002.

TSAI, Y. L., HUANG, K. C., CHANG, H. Y., KO, J., WANG, E. T., HSU, C. H. **Scheduling multiple scientific and engineering workflows through task clustering and best-fit allocation.** *In 2012 IEEE Eighth World Congress on Services* (pp. 1-8). IEEE. 2012.

VARALAKSHMI, P., RAMASWAMY, A., BALASUBRAMANIAN, A., & VIJAYKUMAR, P. **An optimal workflow based scheduling and resource allocation in cloud.** *In International Conference on Advances in Computing and Communications* (pp. 411-420). Springer Berlin Heidelberg. 2011.

VECCHIOLA, C., CALHEIROS, R. N., KARUNAMOORTHY, D., BUYYA, R. Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka. *Future Generation Computer Systems*, 28(1), 58-65. 2012.

VENKATACHALAM, V., FRANZ, M. **Power reduction techniques for microprocessor systems**. *ACM Computing Surveys (CSUR)*, 37(3), 195-237. 2005.

VIGRASS, W. J. **Calculation of Semiconductor Failure Rates**. Intersil, 2001. Available at: http://www.intersil.com/content/dam/Intersil/quality/rel/calculation_of_semiconductor_failure_rates.pdf. Last accessed: October 2016.

WANG, C. M., CHEN, H. M., HSU, C. C., LEE, J. Dynamic resource selection heuristics for a non-reserved bidding-based Grid environment. *Future Generation Computer Systems*, 26(2), 183-197. 2010.

WANG, L.; ZHAN, J.; SHI, W.; & LIANG, Y. **In cloud, can scientific communities benefit from the economies of scale?**. *IEEE Transactions on Parallel and Distributed Systems*, 23(2), 296-303. 2012.

WANG, M., CHEN, J. **Dependency-based risk evaluation for robust workflow scheduling**. *In Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International* (pp. 2328-2335). IEEE. 2012.

WAZLAWICK, R. *Metodologia de Pesquisa para Ciência da Computação, 2ª Edição (Vol. 2)*. Elsevier Brasil. 2014.

WEINGÄRTNER, R.; BRÄSCHER, G. B.; WESTPHALL, C. B. **Cloud resource management: A survey on forecasting and profiling models**. *Journal of Network and Computer Applications*, 47, 99-106, 2015.

WEISSMAN, J. B., GRIMSHAW, A. S. **A federated model for scheduling in wide-area systems**. *In High Performance Distributed Computing*, 1996., Proceedings of 5th IEEE International Symposium on (pp. 542-550). IEEE. 1996.

Workflow Patterns Initiative. Available at: <http://www.workflowpatterns.com/>. Last access: October, 2016.

WU, M. Y., GAJSKI, D. D. **Hypertool: A programming aid for message-passing systems**. *IEEE transactions on parallel and distributed systems*, 1(3), 330-343. 1990.

WU, Z., LIU, X., NI, Z., YUAN, D., YANG, Y. **A market-oriented hierarchical scheduling strategy in cloud workflow systems**. *The Journal of Supercomputing*, 63(1), 256-293. 2013.

WU, Z., NI, Z., GU, L., LIU, X. A revised discrete particle swarm optimization for cloud workflow scheduling. In Computational Intelligence and Security (CIS), 2010 International Conference on (pp. 184-188). IEEE. 2010.

WU, F., WU, Q., TAN, Y. **Workflow scheduling in cloud: a survey.** *The Journal of Supercomputing*, 71(9), 3373-3418. 2015.

XIAO, P., HU, Z. G., ZHANG, Y. P. **An energy-aware heuristic scheduling for data-intensive workflows in virtualized datacenters.** *Journal of computer science and technology*, 28(6), 948-961. 2013.

XIAO, Y., LIN, C., JIANG, Y., CHU, X., SHEN, X. Reputation-based QoS provisioning in cloud computing via Dirichlet multinomial model. In Communications (ICC), 2010 IEEE International Conference on (pp. 1-5). IEEE. 2010.

XU, B., ZHAO, C., HU, E., HU, B. **Job scheduling algorithm based on Berger model in cloud environment.** *Advances in Engineering Software*, 42(7), 419-425. 2011.

XU, M., CUI, L., WANG, H., BI, Y. **A multiple QoS constrained scheduling strategy of multiple workflows for cloud computing.** In *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications* (pp. 629-634). IEEE. 2009.

YANG, T., GERASOULIS, A. **A fast static scheduling algorithm for DAGs on an unbounded number of processors.** In *Proceedings of the 1991 ACM/IEEE conference on Supercomputing* (pp. 633-642). ACM. 1991.

YASSA, S., CHELOUAH, R., KADIMA, H., GRANADO, B. Multi-objective approach for energy-aware workflow scheduling in cloud computing environments. *The Scientific World Journal*, 2013. 2013.

YI, S.; ANDRZEJAK, A.; KONDO, D. **Monetary cost-aware checkpointing and migration on Amazon cloud spot instances.** *IEEE Transactions on Services Computing*, 5(4), 512-524. 2012.

YOO, S., KIM, S. **SLA-aware adaptive provisioning method for hybrid workload application on cloud computing platform.** In *Proceedings of the international multi-conference of engineers and computer scientists* (Vol. 1). 2013.

YU, J., BUYYA, R. **Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms.** *Scientific Programming*, 14(3-4), 217-230. 2006.

- YU, J., BUYYA, R., THAM, C. K. **Cost-based scheduling of scientific workflow applications on utility grids.** *In First International Conference on e-Science and Grid Computing (e-Science'05)* (pp. 8-pp). IEEE. 2005.
- YU, J., KIRLEY, M., BUYYA, R. **Multi-objective planning for workflow execution on grids.** *In Proceedings of the 8th IEEE/ACM International conference on Grid Computing* (pp. 10-17). IEEE Computer Society. 2007.
- YU, J., RAMAMOCHANARAO, K., BUYYA, R. **Deadline/budget-based scheduling of workflows on utility grids.** *Market-Oriented Grid and Utility Computing*, 200(9), 427-450. 2009.
- YU, Z., & SHI, W. **A planner-guided scheduling strategy for multiple workflow applications.** *In 2008 International Conference on Parallel Processing-Workshops* (pp. 1-8). IEEE. 2008.
- YUAN, Y., LI, X., WANG, Q., ZHANG, Y. **Bottom level based heuristic for workflow scheduling in grids.** *CHINESE JOURNAL OF COMPUTERS-CHINESE EDITION-*, 31(2), 282. 2008.
- YUAN, Y., LI, X., WANG, Q., ZHU, X. **Deadline division-based heuristic for cost optimization in workflow scheduling.** *Information Sciences*, 179(15), 2562-2575. 2009.
- ZAMAN, S., GROSU, D. **Combinatorial auction-based dynamic VM provisioning and allocation in clouds.** *In Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on* (pp. 107-114). IEEE. 2011.
- ZENG, L., VEERAVALLI, B., LI, X. **Scalestar: Budget conscious scheduling precedence-constrained many-task workflow applications in cloud.** *In 2012 IEEE 26th International Conference on Advanced Information Networking and Applications* (pp. 534-541). IEEE. 2012.
- ZHANG, J., YOUSIF, M., CARPENTER, R., FIGUEIREDO, R. J. **Application resource demand phase analysis and prediction in support of dynamic resource provisioning.** *In Fourth International Conference on Autonomic Computing (ICAC'07)* (pp. 12-12). IEEE. 2007.
- ZHANG, J., KIM, J., YOUSIF, M., CARPENTER, R. **System-level performance phase characterization for on-demand resource provisioning.** *In 2007 IEEE International Conference on Cluster Computing* (pp. 434-439). IEEE. 2007a.
- ZHANG, Q; CHENG, L; BOUTABA, R. **Cloud computing: state-of-the-art and research challenges.** *Journal of internet services and applications*, 1(1), 7-18, 2010.

ZHANG, Q., ZHANI, M. F., ZHANG, S., ZHU, Q., BOUTABA, R., HELLERSTEIN, J. L. **Dynamic energy-aware capacity provisioning for cloud computing environments.** *In Proceedings of the 9th international conference on Autonomic computing* (pp. 145-154). ACM. 2012.

ZHAO, H., & SAKELLARIOU, R. **Scheduling multiple DAGs onto heterogeneous systems.** *In Proceedings 20th IEEE International Parallel & Distributed Processing Symposium* (pp. 14-pp). IEEE. 2006.

ZHAO, Y.; LI, Y.; RAICU, I.; LU, S.; TIAN, W.; LIU, H. **Enabling scalable scientific workflow management in the Cloud.** *Future Generation Computer Systems*, 46, 3-16. 2015

ZHENG, W., SAKELLARIOU, R. **Budget-deadline constrained workflow planning for admission control in market-oriented environments.** *In International Workshop on Grid Economics and Business Models* (pp. 105-119). Springer Berlin Heidelberg. 2011.

ZHENG, W., SAKELLARIOU, R. **Stochastic DAG scheduling using a Monte Carlo approach.** *Journal of Parallel and Distributed Computing*, 73(12), 1673-1689. 2013.

ZHONG, H.; TAO, K.; ZHANG, X. **An approach to optimized resource scheduling algorithm for open-source cloud systems.** *In 2010 Fifth Annual China Grid Conference* (pp. 124-129). IEEE. 2010.

ZHOU, A. C., HE, B., LIU, C. **Monetary cost optimizations for hosting workflow-as-a-service in IaaS clouds.** *IEEE Transactions on Cloud Computing*, 4(1), 34-48. 2016.

GLOSSARY

- **Branching:** In the context of workflow characterization, branching corresponds to the definition of multiple paths either as input or output of a primitive (see below).
- **Characterization:** Description of an element in terms of a set of characteristics and properties that are commonly observed for other similar elements. In the context of workflows, the workflow characterization is the description of the workflow using a set of characteristics that can be observed (and ideally measured) for other workflows as well.
- **Construct:** Sequence or set of primitives (see below) arranged to implement a specific logic. For example, the set of primitives used to implement a control loop is a control loop construct.
- **Nesting:** Arrangement of multiple workflows so that one workflow becomes the Compute primitive (see below) of another workflow. For instance, a workflow that implements matrix factorization (lower upper decomposition) can be used inside another workflow that solves systems of linear equations.
- **Primitive:** Essential element in the workflow characterization structure. Each primitive implements its own set of internal mechanisms that are interpreted and executed by the framework. Moreover, each primitive has its own rules in terms of branching (see above) and nesting (see above).

APPENDIX A – FURTHER ANALYSIS OF RELATED WORK

Based on the consolidated taxonomy and on the surveys and general literature analyzed, Table 6 provides a comprehensive analysis of several publications related to resource management for cloud computing and also some publications relative to other distribute systems. Several of these works were selected for further analysis. This section presents the method used for selecting these works and an analysis of them.

A.1 Selection Method

All publications marked with an asterisk in Table 6 were selected for further analysis. The selection method adopted is related to the interests of this work, mainly focusing on resource management for workflows, in particular for heterogeneous environments such as hybrid clouds and multcloud scenarios. Moreover, some works comprising multi-objective scheduling were selected (e.g., makespan and reliability).

A.2 Analysis

A.2.1 A Particle Swarm Optimization-based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments

The first selected work is (PANDEY et al., 2010): “A Particle Swarm Optimization-based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments”. It was selected because of its dynamic scheduling properties combined to workflow management

and data-intensive characteristics. The authors propose a heuristic based on PSO that considers both computation and data transmission costs. The workflow is modeled as a DAG. Transfer cost is calculated according to the bandwidth between sites. Average cost of communication between two resources is considered to be applicable only when two tasks have file dependency between them. According to the authors, for two or more tasks executing on the same resource the communication cost is zero. This implies no cost relative to sequential accesses to a file (e.g., the input file), but a rather uniform distribution of content among nodes. On the other hand, for a data-intensive workflow with large inputs and several I/O-heavy intermediary phases, even the cost of accessing resources on the same node cannot be overlooked.

In terms of dynamic scheduling the authors claim that when it is not possible to assign tasks to resources due to resource unavailability, the recomputation phase of PSO dynamically balances other tasks' mappings. This step is usually executed after a period defined by a *polling time* variable. Communication costs are updated according to current network load, which is when the PSO mappings can be dynamically modified. In summary, the main aspect addressed by the dynamic (re)scheduling is related to updating the communication costs in every scheduling loop. There is no explicit mention to dynamically (re)scheduling based on other aspects, such as performance fluctuations and reliability issues.

Workflow support is limited to the usual DAG-based description wherein computation costs of a task on a compute host is a known information and edges represent the communication among phases. This representation provides a limited amount of information regarding the workflow, such as performance fluctuation due to branches and other logic, requirements related to memory and local storage, and the actual performance observed when executing one of the phases on a node.

A.2.2 Scheduling Scientific Workflows Elastically for Cloud Computing

The next work analyzed is (LIN; LU, 2011): “Scheduling Scientific Workflows Elastically for Cloud Computing”. The authors propose an algorithm named SHEFT, Scalable HEFT (Heterogeneous Earliest Finish Time). Initially the authors claim that resources within one cluster usually share the same network communication, so they have the same data transfer rate with each other. While there might be network utilization fluctuations during the execution of a workflow (and even in idle state) that invalidate this affirmation, the fact is that even locally (in the same node) there is data access imbalance due to contention – concurrency to access the same resources, in this case I/O. For example, if two containers (or virtual machines) located in the same node attempt to access a file or a network stream, they will naturally compete for resources.

There is not clear support to dynamic scheduling to address reliability-related issues or performance fluctuations. The solution supports workflows but there are no details on how workflows are modeled or mapped into execution space.

A.2.3 A Multiple QoS Constrained Scheduling Strategy of Multiple Workflows for Cloud Computing

The next work analyzed is (XU et al., 2009): “A Multiple QoS Constrained Scheduling Strategy of Multiple Workflows for Cloud Computing”. The authors propose MQMW, a Multiple QoS constrained scheduling strategy of Multi-Workflows. According to the authors the solution is able to schedule multiple workflows started at any time. Four factors that affect makespan and cost are selected: available service number, time and cost covariance, time quota, and cost quota. Workflows are modeled as DAGs but no specific information about the

modeling is provided. The approach adopted by the authors to support multiple workflows is based on the creation of composite DAGs representing multiple workflows. DAG nodes with no predecessors (e.g., input nodes) are connected to a common entry node shared by multiple workflows. In this sense, new workflows to be executed are joined via a single merging point. Finally, there is no explicit support to dynamic scheduling or heterogeneous environments.

A.2.4 A Federated Model for Scheduling in Wide-Area Systems

The next work is (WEISSMAN; GRIMSHAW, 1996): “A Federated Model for Scheduling in Wide-Area Systems”. This work was selected as it proposes a scheduling solution for heterogeneous environments (wide-area systems) that encompasses data-intensive and dynamic scheduling properties. The solution also maintains local autonomy for scheduling decisions – remote resources are explored only when appropriate. Moreover, according to the authors the unpredictability of resource sharing in large distributed areas requires scheduling to be deferred until runtime.

For data-intensive properties, it is assumed that the system infrastructure is able to access data and files independent of location. If data needs to be transported (e.g., jobs scheduled in a site that does not have direct access to needed data), the scheduling system assumes that data transport cost can be amortized over the course of job execution. If this is not the case the job must be moved to the data – this is handled as a scheduling constraint. In other words, the method used to handle data-intensive workflows and irregularities in data distribution is establishing a constraint wherein worker nodes must be assigning close to the required data. This is not always possible as even local transfers can be expensive, especially if multiple local workers shared the same resources – a common scenario for cloud environments, with a high density of worker elements per physical node.

The authors also mention the necessity of having fault-tolerant mechanisms in place to handle failures across the distributed computing area. However, how these mechanisms are used is considered to be out of scope of the paper. In terms of dynamic scheduling, the wide-area (global) scheduler makes a runtime decision on how to assign resources. In contrast, rebalancing jobs is a local decision. Nevertheless, if the local scheduler decides that there are not enough resources locally to execute a job, this decision is relayed to the global scheduler. The details of these mechanisms unfortunately are also considered to be out of scope by the authors.

A.2.5 An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem With Various QoS Requirements

The next selected work is (CHEN; ZHANG, 2009): “An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem With Various QoS Requirements”. The Ant Colony Optimization (ACO) is a metaheuristic that simulates the pheromone depositing and following behavior of ants and it is applied to numerous intractable combinatorial optimization problems. QoS parameters are based on reliability, makespan, and cost. Reliability is defined as the minimum reliability of all selected service instances in the workflow. The actual reliability aspects used in the calculations, however, are not disclosed. The same applies to cost. Data communication and transfers are not explicitly addressed in the paper.

A.2.6 Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds

The next selected work is (RODRIGUEZ; BUYYA, 2014): “Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds”. The authors

propose a resource provisioning and scheduling solution for execution of scientific workflows on cloud infrastructures. The solution is based on particle swarm optimization aiming at minimizing execution cost while meeting deadline constraints. CloudSim (CALHEIROS et al., 2011), a framework typically used to simulate cloud environments, is used to evaluate the solution.

The general approach adopted by the authors is similar to the one from (PANDEY et al., 2010) – one of the authors participated in both papers. (RODRIGUEZ; BUYYA, 2014) compare their work to (PANDEY et al., 2010) by stating that while cost minimization is highly desirable also for cloud environments (approach adopted by (PANDEY et al., 2010)), according to (RODRIGUEZ; BUYYA, 2014) this aspect of load balancing makes more sense for non-elastic environments such as clusters and grids. They state that workflow execution time is not considered by (PANDEY et al., 2010), as well as the elasticity properties of the cloud by assuming a fixed set of available VMs. However, even in their model (RODRIGUEZ; BUYYA, 2014) make several assumptions that rigidify the cloud environment. For instance, virtual machines are assumed to have a fixed compute capacity (measured in FLOPS), although some degree of performance variation due to degradation is considered in their model. The authors also assume that virtual machines have enough memory to execute the workflow tasks, when in fact several phases of a workflow can be extremely memory hungry (APEX, 2016). In addition, the authors assume that workflows are executed on a single data center or region, and as a consequence the bandwidth between each virtual machine should be roughly the same. However, this might not be true even for a set of nodes connected to the same switch, especially during phases wherein several demanding data transfers are executed among nodes – for example, when inputs are distributed to all worker nodes. Finally, the transfer cost between two tasks being executed on the same virtual machine is assumed to be zero, while the actual

communication can be much more expensive than that, especially if it is via file I/O. Rodriguez et al. implicitly justify this approach by stating that cloud providers usually do not charge for transfers within the same data center, thus this aspect does not have to be considered in the cost calculation. However, all these aspects will affect the performance and the makespan, so it is important to consider them in the performance modeling in order to correctly estimate the total duration of the workflow phases, particularly for data-intensive workflows that are both memory and I/O hungry.

The workflow modeling is based on a DAG with fixed transfer costs (edges). Task costs are calculated based on the size of the task measured in FLOPS. The cost of a task, consequently, depends on the computational complexity of this task instead of the input data. Of course the number of FLOPS can be calculated based on the size of the input data, but no remarks are made in that sense. No other properties are defined, such as performance variation due to branching and input sizes.

The experimental evaluation uses four workflows from different scientific areas: Montage, LIGO, SIPHT, and CyberShake (JUVE et al. 2013). The authors state that Montage and CyberShake are I/O intensive while LIGO and SIPHT are CPU intensive. Also, the evaluation is based on four different deadline levels based on slowest and fastest runtimes. These calculations do not consider transfer times, an intriguing fact considering the evaluation of I/O intensive workflows such as Montage and CyberShake. Nevertheless, these difference between these runtimes is divided by five to get an interval size. Then, the n th deadline interval is calculated by adding n interval sizes to the fastest runtime. Consequently, smaller deadline interval values represent a harder deadline to be met. The experimental results reflect the assumptions related to data transfers, particularly for Montage. The authors compare their

approach (PSO) to SCS (MAO; HUMPHREY, 2011), which is a dynamic approach for scheduling multiple workflows. Results are shown in Figure 57.

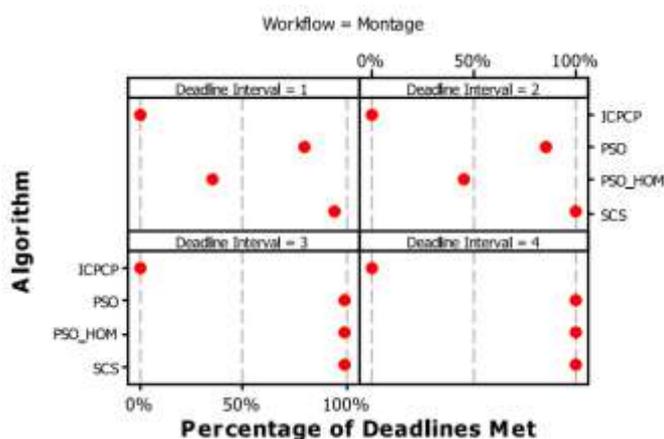


Figure 57 – Experimental results obtained by Rodriguez et al. (figure extracted from (RODRIGUEZ; BUYYA, 2014))

Each of the four smaller boxes shows a deadline interval. Deadline interval = 1 is the most demanding one and deadline interval = 4 is the most relaxed one. X-axis represents the percentage of deadlines met (the higher the better) and Y-axis represents the scheduling algorithms considered. SCS is slightly closer to 100% than PSO especially for lower interval values, then PSO reaches the 100% mark for more relaxed deadlines. In terms of makespan the authors show that their results are comparable to SCS for Montage. The main strength of PSO is related to cost, as the intrinsic properties of cost modeling are not ingrained in SCS. Nevertheless, this series of results shows that it is important to adequately model how resources are leased from cloud infrastructures (which the authors did) but it is equally important to adequately consider transfer times to optimize total run time, especially for data-intensive workflows.

A.2.7 A Multi-Objective Approach for Workflow Scheduling in Heterogeneous Environments

The next selected work is (FARD et al., 2012): “A Multi-Objective Approach for Workflow Scheduling in Heterogeneous Environments”. The authors propose a multi-objective scheduling solution and present a case study comprising makespan, cost, energy, and reliability.

The workflow is modeled as a very simple DAG with fixed size data dependencies among tasks. Nodes are modeled as a mesh network wherein each point-to-point connection has a different bandwidth. Cost is modeled as a sum of computation, storage, and transfer costs. Energy consumption is modeled only after the compute phases of the workflow. The authors state that their focus is on computational-intensive applications, thus only the computation part of the activities are considered in the energy consumption calculation, while “data transfers and storage time are ignored”. Finally, reliability is modeled using an exponential distribution representing the probability of successful completion of a task.

In terms of transfers the makespan is calculated based on the longest input transfer time plus the computation time. This implicitly assumes that input distribution occurs in parallel, or at least it does not consider the contention between I/O activities.

A.2.8 Cost- and Deadline-Constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds

The next selected work is (MALAWSKI et al., 2012): “Cost- and Deadline-Constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds”. The authors investigate the

management of workflow ensembles under budget and deadline constraints on clouds. The resource model is assumed to be similar to the provisioning offered by Amazon Elastic Compute Cloud (EC2) wherein resources are characterized by number of CPUs, amount of memory, and disk space. The application model is related to workflow ensembles modeled as DAGs. Run time estimates are assumed to be known but a variation percentage is adopted to model the performance variation onto a uniform distribution. Moreover, the authors state that although workflows are often data-intensive, the algorithms described do not consider the size of input and output data when scheduling tasks”. In other words, the scheduling cost is uniquely based on computation time. The authors complement by stating that data is stored in a shared cloud storage system and that intermediate data transfer times are included in task run times – transfer time is modeled as part of computation time. It is also assumed that data transfer times between the shared storage and the VMs are equal for different VMs so that task placement decisions do not impact the runtime of the tasks. It is clear, then, that any issues related to contention, performance variation due to network and I/O bandwidth utilization shared among several worker nodes and virtual machines, and the impact of sequentially distributing input among workers are partially or entirely overlooked depending on the case.

In terms of dynamic allocation the authors propose an algorithm named DPDS – Dynamic Provisioning Dynamic Scheduling. Both provisioning and scheduling are handled during run time. DPDS provisions VMs at the start of the ensemble execution and then periodically computes resource utilization based on idle VMs. Based on the result the algorithm adjusts the allocation to minimize idle time. The authors also propose a modified version of DPDS named WA-DPDS (Workflow-Aware DPDS), in which workflow structure is considered in the scheduling process – essentially to determine the order the tasks should be executed.

A.2.9 A low-cost rescheduling policy for efficient mapping of workflows on grid systems

The next selected work is (SAKELLARIOU; ZHAO, 2004): “A low-cost rescheduling policy for efficient mapping of workflows on grid systems”. The authors propose a scheduling mechanisms that considers executing carefully selected rescheduling operations to achieve better performance without imposing a large overhead compared to solutions that dynamically attempt to reschedule before the execution of every task. While the proposal is designed for grid computing, the ideas related to the selection of points of interest to execute the rescheduling operation is relevant also for cloud environments.

The authors state that workflows are usually modeled after directed acyclic graphs, a model that provides an easy way of addressing the resource mapping problem. While several scheduling heuristics exist to address this problem, typically these heuristics assume that a precise prediction is available for the computation and the communication costs. However, as correctly pointed by the authors, for real applications and real environments this statement is not accurate due to the dynamic characteristics of both resources and tasks to be executed, thus leading to an inaccurate initial schedule. The authors argue that two main approaches are adopted to address this issue: 1) schedule all tasks at run time as they become available; or 2) to plan in advance, build an initial static schedule, and then correct it later by rescheduling tasks at run time.

At the time most grid management projects that had some level of rescheduling, but typically addressing set of independent tasks only. Rescheduling adds an overhead to the scheduling and execution process due to the cost of reevaluating the schedule and the cost of transferring tasks accordingly. In contrast, these costs might be offset by the performance gains after rescheduling. In this sense, the authors propose an algorithm that strikes a balance between

the costs and the gains of rescheduling by only considering specific tasks in this process. Experimental results show that their algorithm considers less than 30% of the DAG tasks for rescheduling and they obtain equally good performance compared to policies that reschedule every task.

From an implementation viewpoint their algorithm essentially attempts to optimize makespan while minimizing the frequency of rescheduling attempts. The rescheduling operation is executed when the delay between the real and the expected time is greater than the value of the *slack*, which is the maximum value that can be added to the execution time of a task without affecting overall makespan. If the execution of a task starts at a time greater than the statically estimated starting time plus the *slack*, the overall makespan changes. The results show significant improvement over approaches that attempt to reschedule at the beginning of every task. Depending on the static algorithm used for the initial scheduling, the dynamic mechanism is able to improve run time by up to 10x and by at least 2x. The load redistribution plays a fundamental role to improve final performance. Also, the number of rescheduling operations is also drastically decreased, which also should play another important role to decrease final makespan.

The resource and workflow models adopted by the authors imply a fundamental simplification of how computation and transfer costs are calculated. Each task has a different cost for each machine, expressed as time per data unit. Although this attempts to model performance differences between nodes, this implies that the computation cost of each task linearly varies with the amount of input data. In contrast, if the assumption is that the costs are expressed as a fixed amount, then they are simply fixed to a value assuming a certain amount of input. Both cases do not consider a more sophisticated workflow model in which

computation and communication costs vary according to the size of input data not linearly, but expressed as a general function that can be either predefined or dynamically obtained. This modeling affects both the initial static schedule and also subsequent rescheduling operations.

A.2.10 Dependency-based Risk Evaluation for Robust Workflow Scheduling

The next work is (WANG; CHEN, 2012): “Dependency-based Risk Evaluation for Robust Workflow Scheduling”. The authors propose a cost function that considers the robustness of a schedule regarding the probability of successful execution. Based on the paper, failure is considered to be any event that leads to abnormal termination of a task, and consequent loss of all workflow progress thus far. Afterwards the cost function is used in conjunction with a genetic algorithm to find an optimized schedule that maximizes its robustness. However, in the definition of the cost of failure function the authors assume that the potential loss in the execution cost of each task is independent of the other workflow tasks. In other words, a failure always has a local scope, without possibility of chaining impact outside the workflow. Moreover, there is no workflow characterization in terms of data transfers and task costs. Robustness or failure rates are not specified or tied to a specific property such as MTBF.

A.2.11 Fault-Tolerant Workflow Scheduling Using Spot Instances on Clouds

The next work is (POOLA et al., 2014a): “Fault-Tolerant Workflow Scheduling Using Spot Instances on Clouds”. The authors propose a fault-tolerant workflow scheduling using spot and on-demand cloud instances to reduce execution cost and meet workflow deadlines. The authors claim that their solution is robust against both performance variations and premature termination of spot instances.

Workflow model is based on a DAG. Data transfer times are accounted for with a model based on the data size and the cloud data center internal bandwidth (assumed to be fixed for all nodes). Task execution time is estimated based on the information of number of instructions of the task. For fault-tolerance the authors adopt checkpointing, which consists of creating snapshots of the data being manipulated by the workflow and run time structures, if necessary. The core idea is to store enough information to restart computation in case of an error.

One of the issues with the approach adopted is how checkpointing is considered in the model. Checkpointing worst-case scenario requires a full memory dump, meaning that 100% of the memory contents have to be written to a persistent storage (e.g., spinning disks). Depending on the memory footprint of the workflow phase this amount surpasses the order of gigabytes – in fact, for modern and future workflows such as the APEX Workflows (APEX, 2016) these amounts might surpass the order of terabytes per hour. However, in the model proposed in the paper the checkpointing cost is not considered “as the price of storage service is negligible compared to the cost of VMs”. Moreover, while checkpointing time was considered in their model, the actual checkpointing time on spinning disks, especially for cloud systems that are not specialized for parallel I/O, can represent much more than 10% of overhead, which is the value expected for very large-scale machines such as APEX and EXASCALE. Thus, either the checkpointing size adopted is much smaller than what is observed for real scientific workflow or the checkpointing mechanism is creating partial checkpoints. Nevertheless, the results obtained by the authors show that having checkpoints actually reduces the final cost. Yet, the fault-tolerance provided by the method only covers the repair part, not the fault avoidance part. There is no (explicit) logic to predict the probability of occurrence of failures due to some hardware or software property, for instance.

A.2.12 HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds

The next work is (BITTENCOURT; MADEIRA, 2011): “HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds”. The authors propose HCOC, the Hybrid Cloud Optimized Cost, a scheduling algorithm that selects the resources to be leased from a public cloud to complement the resources from a private cloud.

The workflow is modeled as a DAG. The objective of HCOC is to reduce makespan to fit a desired execution time or deadline while maintaining a reasonable cost. This cost constraint is introduced to limit the amount of resources leased from the public cloud, otherwise the public cloud would always be overutilized to address the time constraints. Intra-node communication is considered to be limitless, in the sense that the costs of local communication are ignored. Communication cost is calculated by dividing the amount of data by the link bandwidth, which is modeled as a constant value. Computation cost is based on the number of instructions and the processing capacity of a node, which is measured as instructions per time. There are several implicit assumptions in this model, such as fixed capacity for transferring and computing. There is not a function that varies the amount of computation based on the size of the input.

The HCOC algorithms starts by scheduling the execution using the private cloud. Then, if the projected makespan is larger than the deadline the algorithm selects the tasks to reschedule (i.e., to schedule considering additional resources from the public cloud), select the resources from the public cloud, and then recalculates the distribution. This process is repeated until the makespan is larger than the deadline. It is worth noticing that the rescheduling process adopted by the authors simply aims at correcting an initial schedule that does not address the deadline imposed by the user. In other words, it is simply a two-step scheduling rather than an actual dynamic process of rescheduling according to workflow and system conditions, for instance.

A.2.13 Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka

The next work is (VECCHIOLA et al., 2012): “Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka”. The authors claim that scientific applications require a large computing power that typically exceeds the resources of a single institution. In this sense, their solution aims at providing a deadline-based provisioning mechanisms for hybrid clouds, allowing the combination of local resources to the ones obtained from a public cloud service. The authors highlight the initiative named Science Cloud, deployed by the joint efforts of the University of Chicago, University of Illinois, Purdue University, and Masaryk University. The programming models supported by their solution include bag of tasks (i.e., describe a distributed application as a collection of independent tasks that can be executed in any order), distributed threads, MapReduce, and workflows. There are no specific details on how workflows are internally handled by their solution, nor how resources are mapped to workflow phases or how costs are calculated. Moreover, their solution (named Aneka) focuses on meeting a specific deadline, thus not addressing issues related to total execution time (makespan) or reliability.

A.3 Summary of Selected Related Work

Table 35 presents a summary of the analyses of the selected related work.

Table 35 – Summary of analyses of related work

<i>Work</i>	<i>Data transfers and imbalance</i>	<i>Dynamic scheduling</i>	<i>Hybrid and Multicloud</i>	<i>Workflow support</i>
PANDEY et al., 2010	Transfers are evaluated via workflow DAG and resource allocation; transfer imbalance is not addressed.	Only addresses fluctuations in the transfer costs. Other aspects such as performance fluctuations and reliability are not mentioned.	No explicit support or experiments.	Modeled as DAGs; richer characterizations are not supported.
LIN; LU, 2011	Transfer capacity of nodes in the same network are assumed to be uniform. Transfer imbalance is discarded.	Not addressed.	No explicit support or experiments	Supported; no details included.
XU et al., 2009	Transfers and data properties are not explicitly addressed.	Not addressed.	No explicit support or experiments.	Multiple workflows supported via common merging point; simple DAG modeling.
WEISSMAN; GRIMSHAW, 1996	Data locality is a scheduling constraint; worker must be assigned closer to data.	Two levels: local and global. Rescheduling is first handled on local level. Details are not provided.	Design for wide-area systems (pre-dates cloud computing).	No explicit support.
CHEN; ZHANG, 2009	Data communication and transfers are not explicitly addressed.	Not addressed.	No explicit support or experiments.	Simplified DAG model without edge costs.

Table 35 – Summary of analyses of related work (cont. I)

<i>Work</i>	<i>Data transfers and imbalance</i>	<i>Dynamic scheduling</i>	<i>Hybrid and Multicloud</i>	<i>Workflow support</i>
RODRIGUEZ; BUYYA, 2014	Rigidly modeled; fixed costs for transfers and no cost for local I/O.	Not addressed.	Not addressed.	DAG with fixed transfer costs and computation costs based on FLOPS.
FARD et al., 2012	Transfers considered but no contention. Energy calculations ignore transfer times.	Not addressed.	No explicit support or experiments.	DAG with fixed transfer costs; not details on task costs.
MALAWSKI et al., 2012	Algorithm does not consider the size of input data; transfer time is part of computation.	Initial scheduling plus periodic adjusting depending on amount of idle resources.	No explicit support or experiments.	DAG with fixed transfer costs and computation costs with slight variability.
SAKELLARIOU; ZHAO, 2004	Linear variation to amount of input data size.	Immediately before execution of tasks and bound to a condition to minimize number of reschedules.	Not addressed – solution originally designed for grids.	DAG with computation and transfer costs modeled with linear variation w.r.t. amount of input.
WANG; CHEN, 2012	Not addressed. DAG does not specify transfer costs.	Not addressed.	No explicit support.	DAG with tasks and implicit costs. No transfer costs.
POOLA et al., 2014a	Based on data size and one value for network bandwidth.	Not addressed.	No explicit support.	DAG with task cost based on number of instructions.
BITTENCOURT; MADEIRA, 2011	Based on data size and fixed network bandwidth values among nodes.	Two-step scheduling: static, then including public cloud to address deadline.	Public resources are used if necessary.	DAG with compute cost based on number of instructions.
VECCHIOLA et al., 2012	Not specified.	Not addressed.	Public resources used if necessary.	Supported, but no details provided.

A brief analysis of these works reveals that none of them addresses all four points of interest of this work, namely the support for data-intensive workflows, for hybrid and multicloud scenarios, for handling performance fluctuations, and for addressing reliability issues.

APPENDIX B – WORKFLOW CHARACTERIZATION

The workflow characterization is used to describe the workflow and its phases. The transfer and computation costs for each phase are described as a function of the input data to be manipulated. These values are used to calculate an estimate of the total run time for the workflow on a particular cloud infrastructure, as well as to determine the optimal input distribution to minimize makespan.

B.1 Characterization Primitives

MPSF uses a workflow characterization based on simple primitives that describe how to handle the input in each phase of the workflow. These primitives are presented in Figure 58.



Figure 58 – Primitives considered in the workflow characterization model

B.1.1 Input Primitive

The Input primitive is used to indicate the sources of input data, as well as to determine the beginning or reorganization point in the workflow. For workflows composed by multiple iterative pipelines, the Input primitive is also used to indicate the restart point for each iteration, or the restart point in case of a failure. These use cases are illustrated by Figure 59.

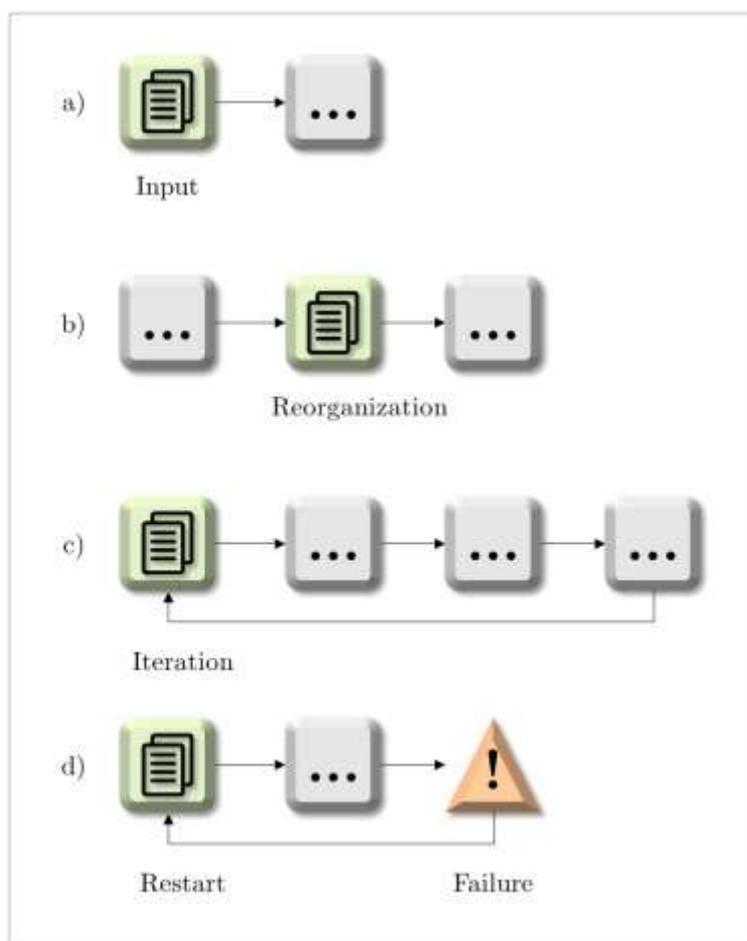


Figure 59 – Use cases for Input primitive

Use case (a) is the utilization of the Input primitive to indicate the beginning of a workflow, as well as to determine the sources of input data. Simple execution environments are composed by a single source of data, such as a centralized database or a simple input file stored in one of the nodes. More complex environments are composed by several inputs, such as a distributed database or a distributed file system. The Input primitive must also describe how to access and read the input data in order to distribute it to the worker nodes and containers, which will execute the actual computations.

Use case (b) is the utilization of the Input primitive as a reorganization step. For example, after merging results from several nodes and containers, MPSF can be directly

instructed to recalculate the input distribution based on the new results. This is useful, for example, for algorithms that generate a variable amount of outputs depending on the initial conditions of computation. If these outputs are then reused in a subsequent step of the workflow, redistributing the output (which is the input for the next phase) guarantees that the execution continues to be balanced in terms of run time and resource allocation. On the other hand, MPSF also has mechanisms to enforce input redistribution after the completion of each phase, or after the completion of specific phases and primitives (e.g., after every Merge operation).

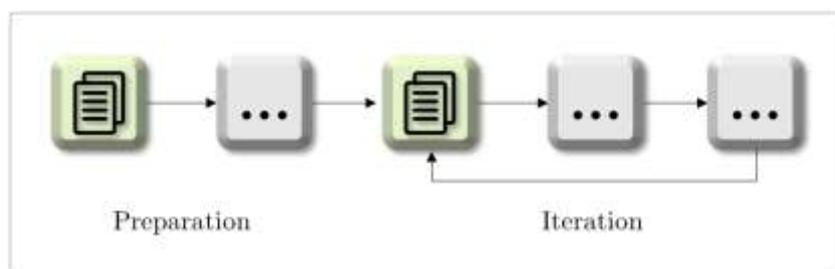


Figure 60 – Example of Input primitives with distinct points for preparation and iteration

Use case (c) is the utilization of the Input primitive to determine the restart point for sequential iterations, often observed in workflows composed by multiple pipelines that iterate over a data set until the convergence criteria are observed, for instance. While in some cases the Input defined to indicate the beginning of the workflow might coincide with the restart point for subsequent iterations (as depicted in Figure 59 for this use case), the iteration point might differ from the initial starting point, as depicted in Figure 60. In this example the initial phases of the workflow correspond to a preparation phase wherein the input data is initially processed. Then, a second Input primitive is used to indicate the beginning of the iterations with its own sub-pipeline of phases.

Use case (d) is the utilization of the Input primitive to indicate the restart point in case of failures or unexpected events. In this case the primitive indicates the workflow phases that can act as a safe convergence point, so that in case of any failure event the computation can be restored to a previous point without losing all progress. Figure 61 illustrates this use case.

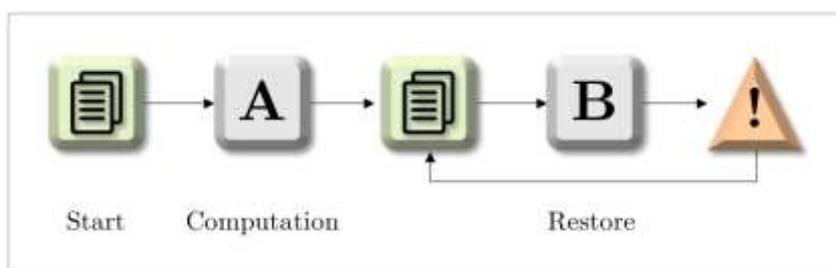


Figure 61 – Example of Input primitive acting as restart point

The first Input primitive (from left to right) is used to indicate the beginning of the workflow, also defining how to retrieve the input data from the sources. There is a computation phase A that processes this input into a set of subproducts that are used by computation phase B. Without the intermediary Input phase, in case of failure during phase B the computation has to be restarted in the beginning of the workflow, before phase A. However, because the output from A can be persistently stored to be subsequently used by B, an intermediary Input phase can be added between A and B to indicate that in case of failure, computation can be restarted after A by reusing its outputs. The intermediary Input phase must define how to persistently store data from A, as well as how to retrieve this data for B in case of failure. For example, the natural transition from A to B could be done via memory, so that the outputs from A are written to memory (volatile) and read by B. The Input phase defines a set of I/O functions to persistently store the output from A (e.g., in a file) and, in case of failure, the files can be read and the data is fed to B. The process can occur in parallel (i.e., persistent copies are created while data is

used by B) or sequentially (i.e., first the backup copies are created, then the workflow pipeline is resumed).

B.1.2 Split Primitive

The Split primitive is used to indicate the workflow phases responsible for distributing data to multiple worker nodes and containers. Figure 62 depicts a simple use case of Split primitive, connected to a Input primitive.

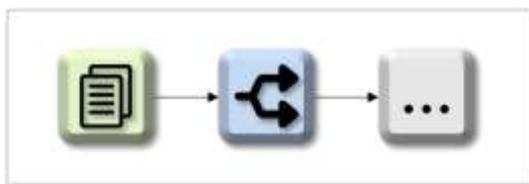


Figure 62 – Simple use case of Split primitive

The Input primitive defines how the sources can be accessed and how their data can be retrieved (e.g., by opening and reading a file, or by using a set of API functions to access a database). The mechanisms defined by the Input primitive are then used by the Split primitive to read the input data and distributed it to the worker nodes and containers. The Split element also defines rules for distributing this data. For instance, it could define a particular set of nodes and containers that should process this data. The rules can be based on performance constraints, resource constraints (e.g., only transfer the input to worker nodes and containers that have a particular resource, such as a GPU device), and even security constraints (e.g., only transfer the input to workers from a particular cluster or subset of nodes that exhibit certain security characteristics, or that possess certain security devices such as a TPM module).

A more generic example of utilization of the Split primitive is observed in Figure 63. In this case the data source is not an explicit Input, but any other primitive supported by the framework. It could be a Compute element, a Merge, or even another Split. In this case the precedent primitive must define how its data can be accessed by other primitives.

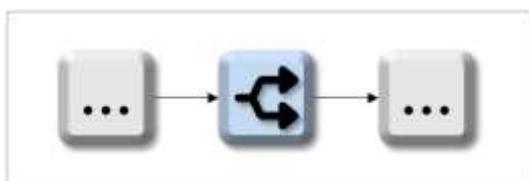


Figure 63 – Generic use case of Split primitive

Figure 64 illustrates a case wherein the Split primitive occurs right after another Split operation. For instance, a data set first is split among a certain level in the node hierarchy (e.g., each node receives a balanced chunk of the initial input), then the second split distributes each chunk for a second level of organization in the hierarchy (e.g., balanced distribution among containers). A Split primitive always forces the framework to execute a scheduling and resource allocation step to (re)distribute data among worker nodes and containers.

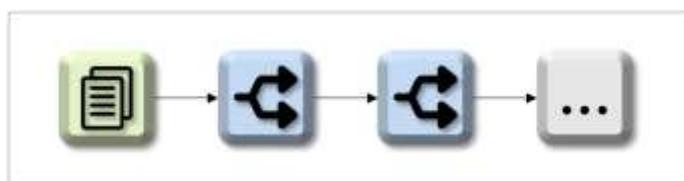


Figure 64 – Example of Split after Split

Figure 65 illustrates an example of a Split primitive after a Compute element. This represents a situation in which the output generated by a Compute phase must be distributed among the working nodes and containers for further processing, for instance. This setup must specify how the worker hierarchy is used to distribute the Compute tasks and the tasks that occur after the Split.

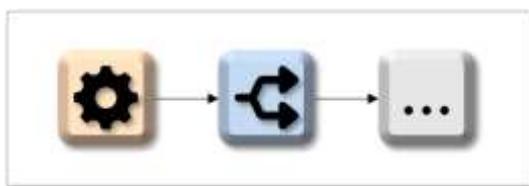


Figure 65 – Split primitive after a Compute element

Figure 66 illustrates an example of Split primitive after a Merge. A Compute phase processes data retrieved using the information from the Input primitive. The results obtained by the worker nodes and containers are merged, resulting in a preliminary set of outputs. Finally, these outputs generated by the Merge component are split again to worker nodes for further processing.

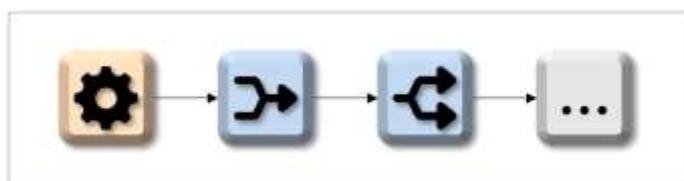


Figure 66 – Split primitive after a Merge

Figure 67 presents a similar Merge-Split setup as observed in Figure 66 but including an Input phase between the Merge and the Split. The difference in this case is that the transition from Merge to Split must include some form of persistent access method, such as a file or database. In the direct Merge-Split transition the data communication can be performed via memory. Including the Input primitive forces the framework to generate a persistent copy of the data being transferred from one phase to the other, even if the actual transition still occurs via memory. This situation is similar to the Input use case (d) from Figure 59, or the example of Input primitive as a restart point presented in Figure 61.

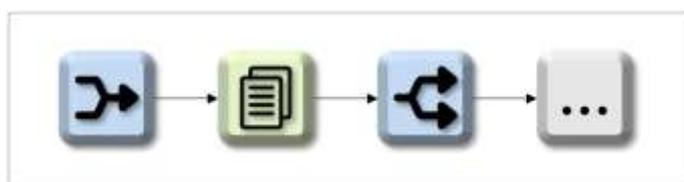


Figure 67 – Similar Merge-Split setup with an intermediary Input phase

Figure 68 shows an example of Split primitive used after an Output component. The Output component in this case functions as a data source from the Split perspective, and the Output component is considered to represent a partial result generated by the pipeline.

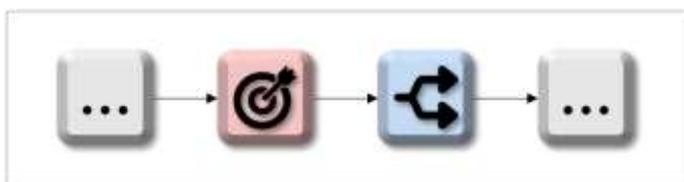


Figure 68 – Output-Split setup

B.1.3 Compute Primitive

The Compute primitive is used to indicate the phases with actual computation executed over the course of the workflow and its pipelines. There is no fixed rule for the complexity of the Compute primitive. In other words, it could represent a simple function or script, or it could represent a full-fledged application or set of applications. As a rule of thumb, the Compute primitive represents a computational element that receives inputs and produces outputs. The methods used to read the inputs and write the outputs are usually defined by the Compute element (e.g., read from command line and write to a file, or read from a file and send the results to a network port). The elements that enclose the Compute component must be able to support these methods for transparent integration of the component to the rest of the workflow. Note that based on this definition, the whole workflow can be interpreted as a single Compute component. This abstraction is used to create nested workflows or composition of workflows.

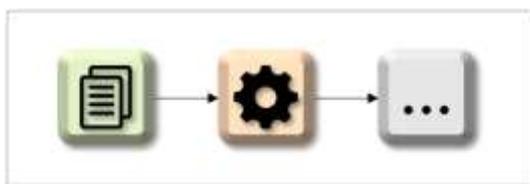


Figure 69 – Example of Compute primitive after an Input

Figure 69 presents an example of Compute primitive right after an Input component. In this case the whole data set defined by the Input primitive is processed by the Compute. The example presented in Figure 70 shows a particular case wherein the Compute primitive is enclosed by an Input and by an Output. This represents a very simple case in which all input data is processed by a simple Compute element and by a single worker node or container, as there is no Split mechanism (and complementary Merge) specified for this workflow.

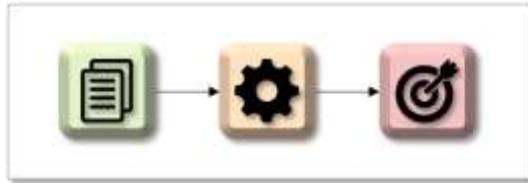


Figure 70 – Example of Compute primitive enclosed by an Input and an Output

MPSF transparently interprets certain primitive transitions into an expanded version that is actually processed by the framework. Figure 71 shows a typical workflow construct (i.e., sequence of primitives) composed by an Input, a Split, and a Compute element. This could represent, for instance, the calculation of the sum of the lines of a matrix. The Input indicates the source of the matrix, the Split primitive divides the matrix into rows and sends one row per Compute node, which calculates a local sum of the elements. Then a further element would be responsible for merging the results and calculating the final value.

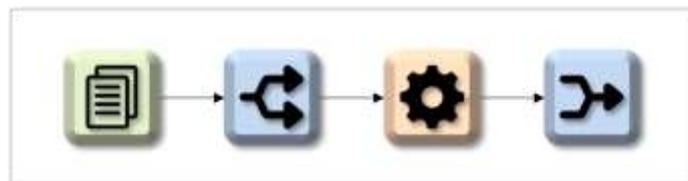


Figure 71 – Typical workflow construct composed by a sequence of Input, Split, Compute, and Merge

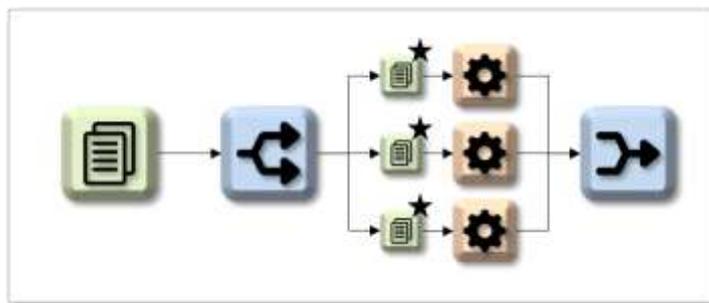


Figure 72 – Construct as internally interpreted by MPSF

Internally, this construct is transformed into the sequence presented in Figure 72. The intermediary Input primitives generated by the framework are marked with a symbol to indicate that not only they were dynamically generated (i.e., not specified by the user), but also that their configuration differs from the original Input primitive specified in the workflow characterization. The original Input relies on a persistent storage solution for regular I/O (e.g., a file that contains the input matrix), while the dynamically generated Input primitives rely on a volatile data source (usually system memory) to integrate the Split phase to the Compute phase. Implementation and run time wise, the matrix is read from a file by the Split element and each working node receives a chunk of input via network or directly via memory.

This dynamically generated construct is different from an explicit construct with an intermediary Input, as depicted in Figure 73. This construct generates the final construct depicted in Figure 74.

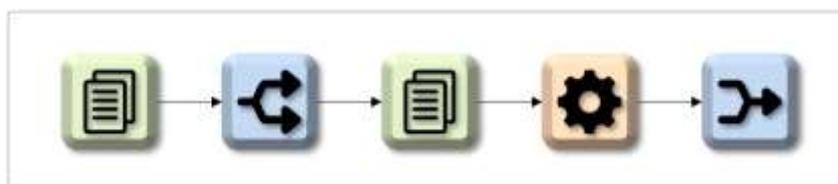


Figure 73 – Workflow construct with explicit Input between Split and Compute

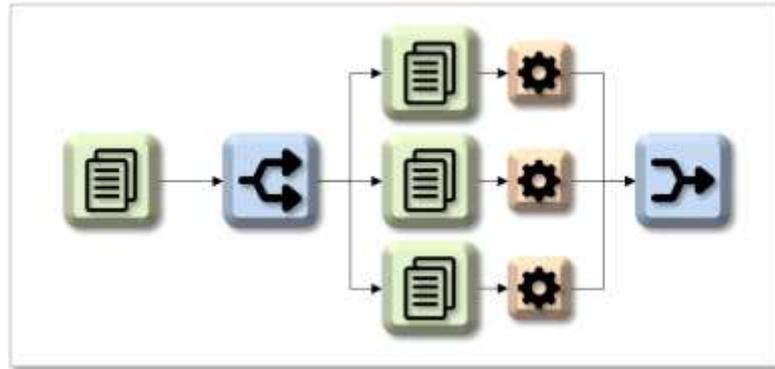


Figure 74 – Derived construct with explicit Input primitives

The Input primitives dynamically generated between the Split and the Compute compose a persistent storage layer between these phases, specifying how the Split output has to be stored in the execution environment.

B.1.4 Merge Primitive

The Merge primitive is used to determine the workflow phases responsible for aggregating data into consolidated data sets. The Merge primitive is used, for instance, to determine a focal point of a workflow wherein the final results are computed and joined into a single value or set of values. This use case is illustrated by Figure 75.

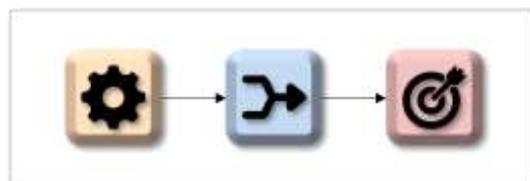


Figure 75 – Example of basic usage of Merge primitive to generate the final results of a workflow

Another use case for the Merge primitive is to reorganize the data and load distribution by adding a Merge-Split composite phase before moving to the next computation phase, as depicted in Figure 76.

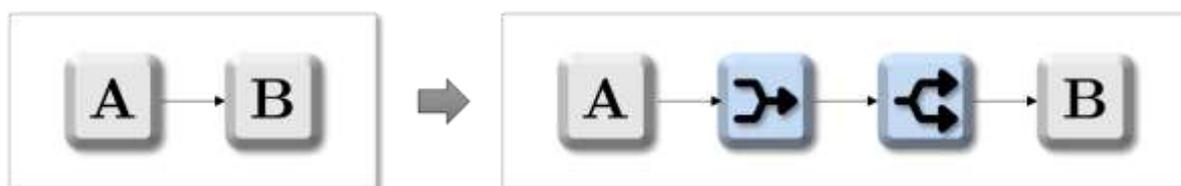


Figure 76 – Intermediary Merge-Split to reorganize load distribution

This construct is useful to redistribute data after a computation that inherently unbalances the load distribution. For example, after a Compute phase A that generates a variable amount of output depending on the inputs processed, if this phase was directly followed by the next Compute phase B there could be a critical imbalance among worker nodes and containers. Having a Merge-Split construct guarantees that a load distribution phase will precede the next computation phase.

To address workflows organized in a hierarchical structure, the Merge primitive can be combined to another Merge component to implement a gradual merging process among nodes and containers. This use case is depicted in Figure 77.

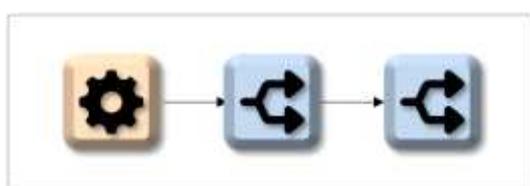


Figure 77 – Nested Merge primitives

For instance, the first Merge could be executed in the context of a node (e.g., merging the outputs generated by all containers from a node) while the second Merge could be executed in the context of a rack or a cluster of nodes. Depending on how the workflow logic is defined, each Merge could implement a different algorithm to actually merge the results.

B.1.5 Output Primitive

The Output primitive defines the workflow phases that produce preliminary and final results for the workflow execution. The use cases for this primitive are summarized in Figure 78.

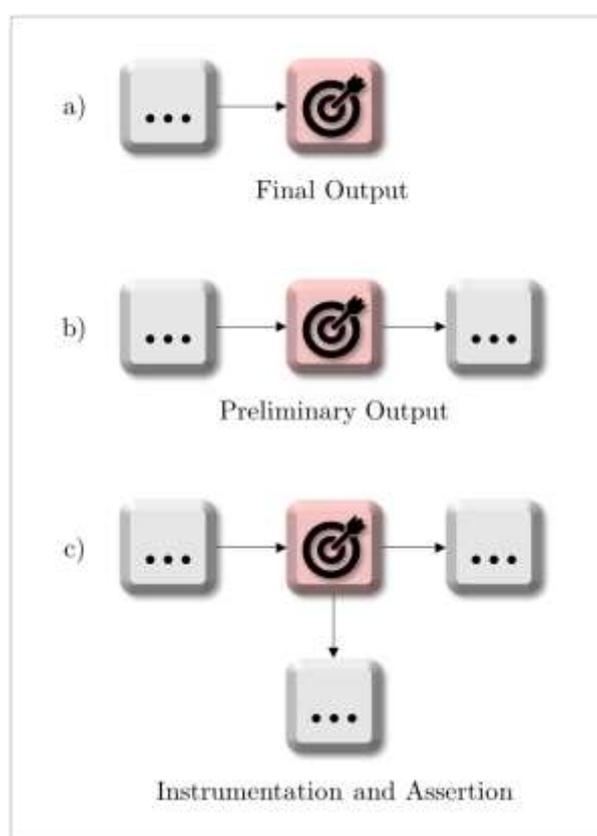


Figure 78 – Exemplar use cases for Output primitive

Use case (a) is the most usual utilization for this primitive. In this case the primitive indicates the final output generated by a workflow or by one of its pipelines. The primitive must also specify how the output will be stored.

Use case (b) represents the utilization of the Output primitive to indicate a preliminary result or output generated by the workflow. In this case the primitive is used to indicate intermediary phases or results that should be stored.

Use case (c) represents the utilization of the Output primitive as a instrumentation or assertion point in the workflow. In this case the primitive is used to store a value or set of values generated by a particular phase of the workflow and then compare it to an exemplar value for validation purposes, or to simply verify its value in a long computation for inspection purposes, for instance.

B.2 Branching and Multiple Paths

Branching is the definition of multiple paths with distinct phases for a workflow. Figure 79 shows a simple example of branching.

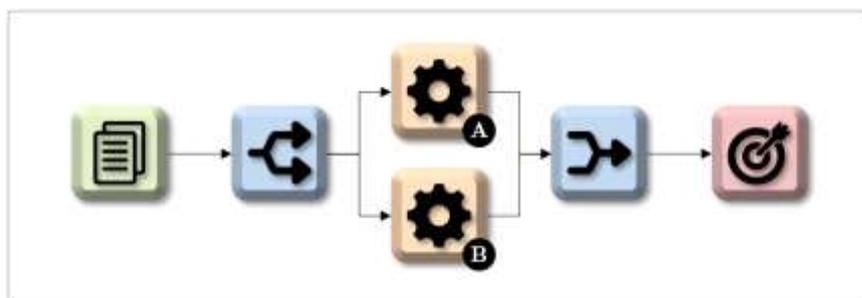


Figure 79 – Example of branching

In this example the Split element has two distinct branches, one going to Compute element A and another to Compute element B. The branching logic in this case is concentrated

in the Split element. For instance, Compute A might be responsible for processing values above a certain threshold while Compute B is responsible for values below this threshold. In this case the Split element processes the input and distributes it accordingly.

The branching and multiple paths rules supported by the framework are depicted in Figure 80.

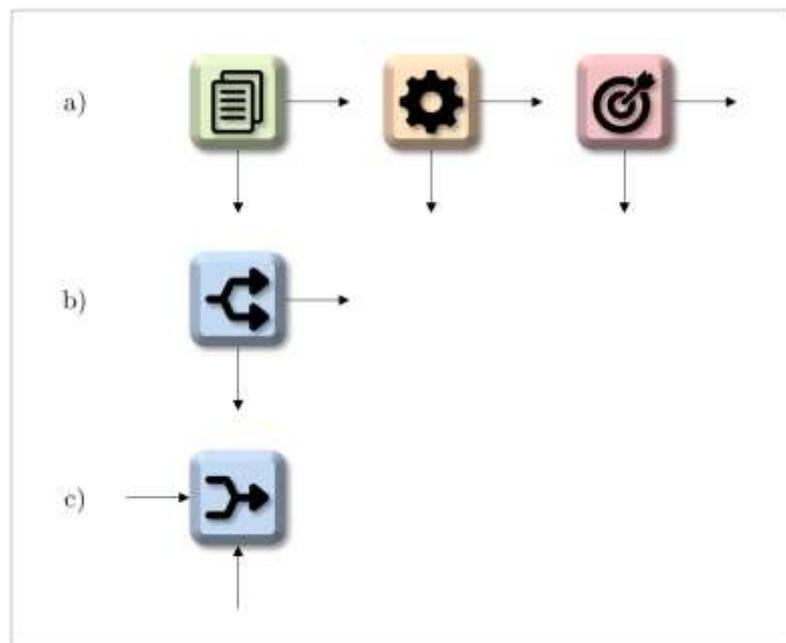


Figure 80 – Supported branching and multi-path rules

Rule (a) applies to the Input, Compute, and Output primitives. These primitives only support multiple outputs. In their case the outputs are replicated for each receiver. Figure 81 shows an example of Input primitive with multiple outputs.

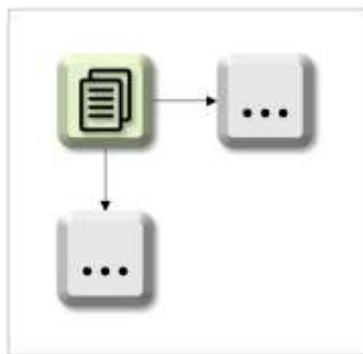


Figure 81 – Example of Input with multiple outputs

In this example each branch receives a copy of the data defined by the Input element. The order that each branch receives the data depends on the internal implementation of the Input primitive. Another similar example is the use case (c) presented in Figure 78, wherein the Output primitive has two output branches, one to define the rest of the computation, and another used for instrumentation and assertion purposes.

Rule (b) from Figure 80 applies to Split constructs. The Split primitive supports the definition of logic for splitting data and distributing it among the worker nodes and containers. Moreover, for Split primitives with branching an additional parameter is used to calculate the number of required workers for each branch. This number can be prefixed in the workflow characterization or dynamically calculated depending on the received input.

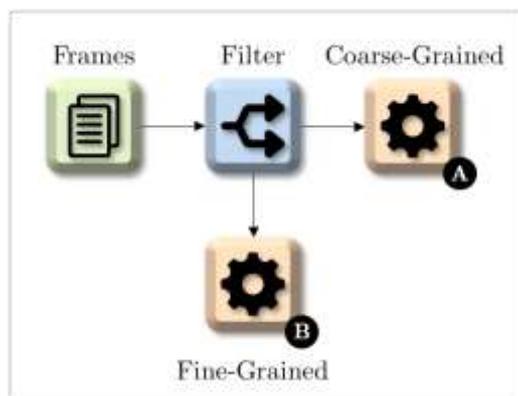


Figure 82 – Example of branching using the Split primitive

Figure 82 shows an example of Split construct with a branch. This could be, for example, an image processing workflow wherein there are two main cases: if there is movement in the image frame to be processed, the frame must be sent to Compute A; else, the frame is sent to Compute B. In this case Compute A could be an algorithm to demarcate the frame regions with movement and log into a surveillance system. In parallel, Compute B could be a finer-grained processing element to further analyze the image to find any subtle changes. Because B is much more expensive than A, it is interesting from a resource saving perspective to only apply B to images in which movement really is not apparent.

In this example the Split element is responsible for doing the initial filtering of frames, thus it must detect movement between frame transitions. Because this is not a cheap operation by itself, the workflow could be augmented by the construct presented in Figure 83.

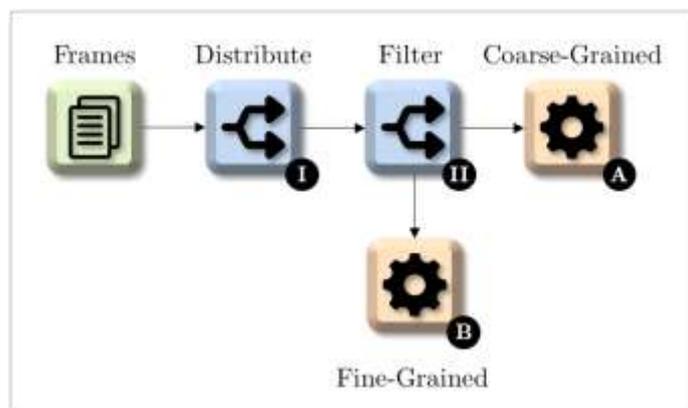


Figure 83 – Enhancement of image processing workflow by adding another Split

Now before the Filter (Split II) there is a Distribute phase (Split I) that distributes the input data among several nodes to perform the filtering, then the input data is redirected to either the Coarse-Grained or the Fine-Grained analysis.

Finally, rule (c) from Figure 80 applies to Merge constructs. In this case the Merge primitive may receive multiple inputs and generate a single output stream. The contents supported by the Merge completely depend on its internal logic.

Loops and other flow control methods are implemented by combining the primitives and the branching capabilities of the framework. For example, a loop can be implemented using the construct presented in Figure 84.

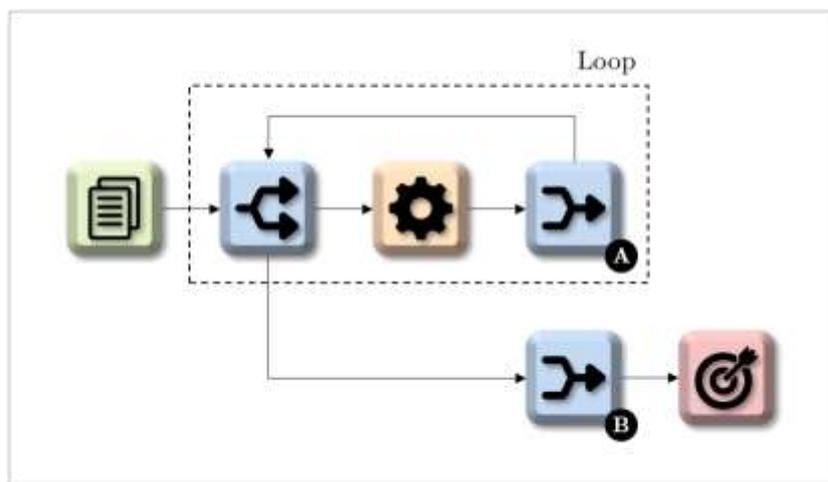


Figure 84 – Example of construct implementing a loop

The loop condition is controlled by the Split element. Internally the Split must be implemented to analyze the input conditions and determine whether the loop must be executed or the flow defined by the Merge B. The Compute element could be an algorithm and the results from Merge A could be a set of values and a convergence factor that controls the loop execution. With this construct the execution of the Compute element will be distributed among the available or selected worker nodes and containers, effectively parallelizing the execution of the algorithm. Finally, when the convergence condition is reached, the results are merged and consolidated.

B.3 Nesting and Workflow Composition

Nesting refers to the composition and reutilization of workflows as part of other workflows. A Compute primitive is, by definition, an element that receives a set of inputs, does some sort of processing over these inputs, and produces a set of outputs. In a more general abstraction, this is what a workflow (or at least one of its pipelines) does: it starts with a set of inputs, then it performs several operations defined by its inner primitives and constructs, and

then produces a final output. Consequently, a workflow can be abstracted into a single Compute element. This is illustrated by Figure 85.

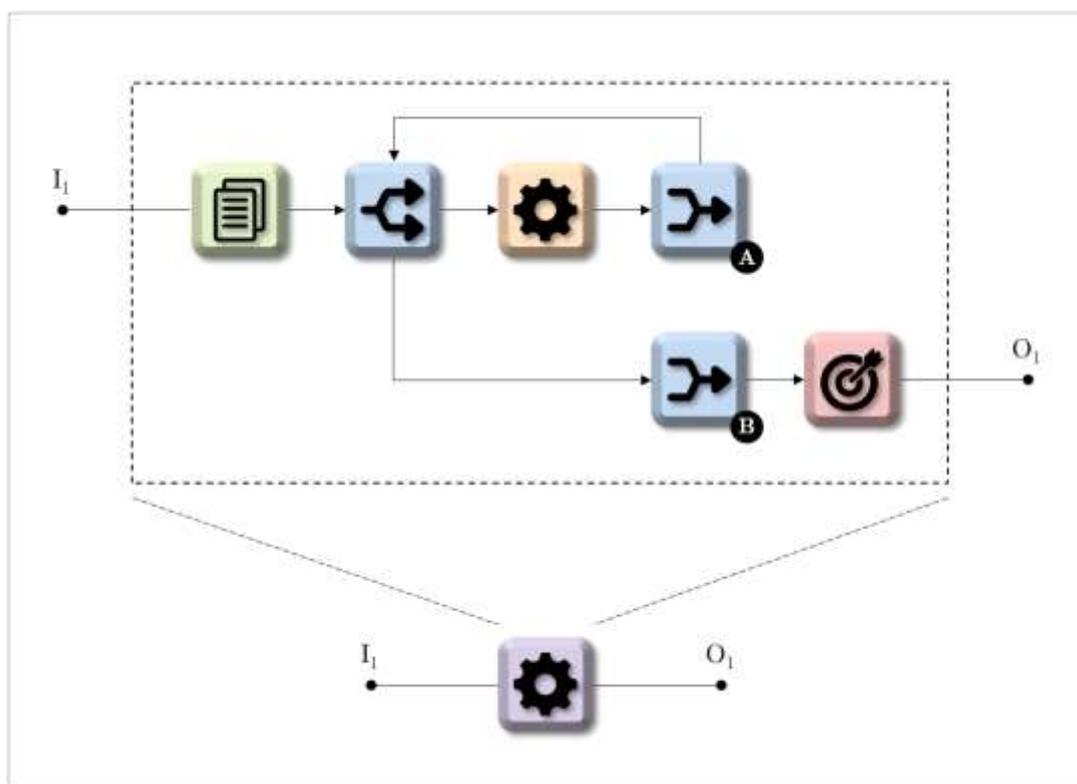


Figure 85 – Abstraction of a workflow (pipeline) into a single Compute element. The abstract Compute element was represented in a different color only for aesthetic differentiation.

The exemplar workflow in Figure 85 implements a simple loop (controlled by the Split primitive). The inner Compute processes the input data and the Merger A generates the final products of an iteration. When the convergence condition is reached, the Split primitive redirects the execution flow to the Merger B, which is responsible for generating the final results managed by the Output primitive. Two connection points were added to this workflow, one attached to the Input element, and another attached to the Output element. The connection points are named (I_1 and O_1 , respectively), allowing them to be identified when the workflow

is reused. Figure 86 shows an example of workflow reusing the construct presented in Figure 85.

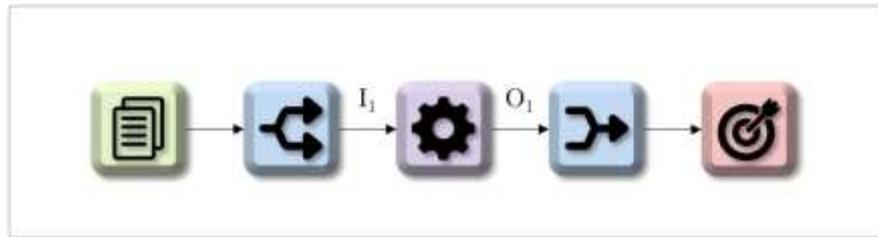


Figure 86 – Example of reutilization of a workflow

This workflow distributes the input data set among several worker nodes and containers and then uses the base construct to obtain the final outputs. The Merge primitive could be a simple function to join all results in a single list of outputs.

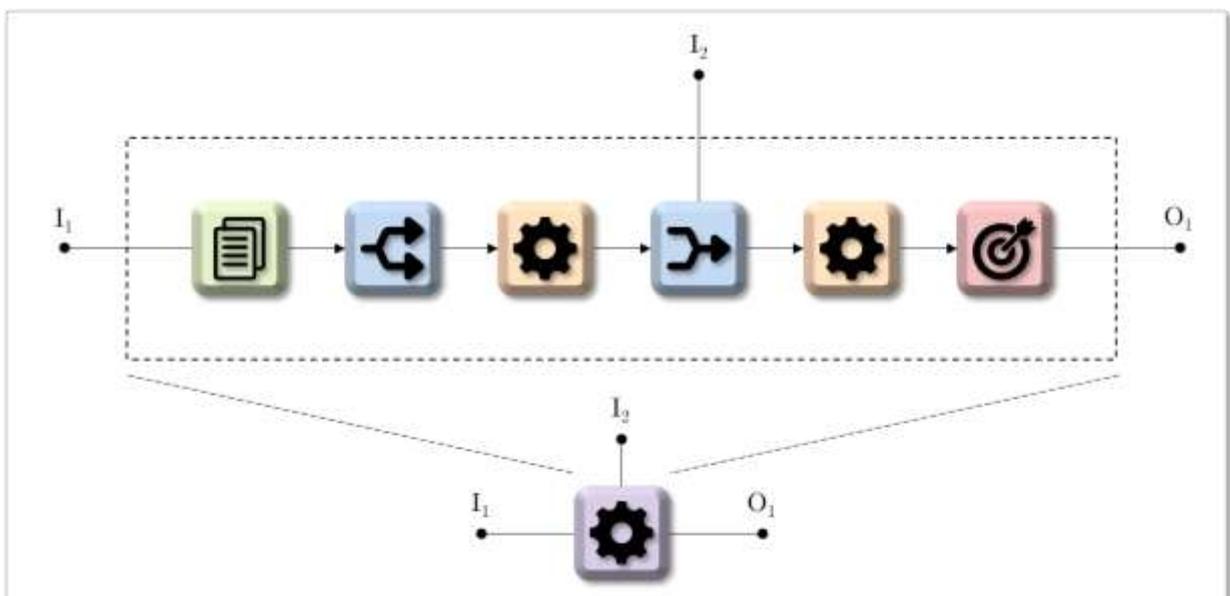


Figure 87 – Another example of workflow transformed into a single Compute construct

Figure 87 shows another example of workflow transformed into a single Compute construct. In this case the workflow receives two inputs, one with an Input primitive associated

(e.g., a persistent method for exchanging data among phases), and another with a Merge element (e.g., data transfer via memory or network). Figure 88 presents an example of workflow using the construct from Figure 87, including how the connectors are used to distribute data.

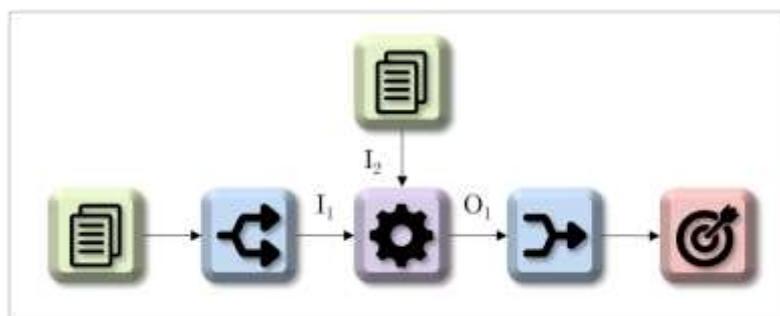


Figure 88 – Example of workflow using the construct from Figure 87

B.4 Transfer and Computation Costs

The workflow characterization is also used to determine the costs of transfers and computation. These costs are expressed as a function of the amount of input data, similar to the notation used to express algorithm complexity.

B.4.1 General Notation

Figure 89 presents an example of general notation used to describe the costs of transfers and computations in a workflow. Each phase has an associated computation cost, while each connection between phases has a transfer cost.

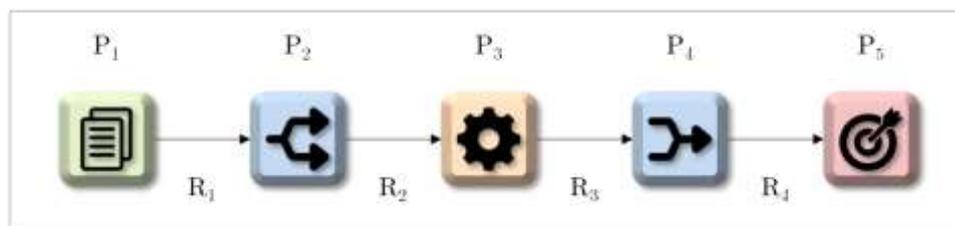


Figure 89 – Example of general notation to indicate cost of transfers and computation

In this example P_1 is the cost to prepare the sources to be read while R_1 is the time to read these sources. P_2 is the time to Split the data sets read from Input and R_2 is the time to send each resulting chunk to a worker node. This particular cost is usually expressed as the time to send the chunks to a particular worker node – not as a sum of all transfers to all nodes. Then, P_3 is the time spent by the main Computation defined in this workflow and R_3 is the time to send the results to the Merge function. P_4 is the time to merge these results, R_4 is the time to transfer these results to the final output, and P_5 is the time to prepare the Output for receiving these results.

B.4.2 Cost Functions

Cost can be represented using three different methods:

- **Fixed value:** A fixed value is set for each phase and transition. This method is useful for workflows and constructs that are used frequently with a similar set of inputs, thus it is easier to predict the duration of each phase.
- **Function:** The cost of each phase and transition is expressed using a mathematical function. This function can be defined by the user who characterized the workflow or it can be automatically obtained by the framework via incremental experiments.

- **Complexity notation:** A simplified version of the Function method, the complexity notation can be used to quickly create a characterization model for the workflow.

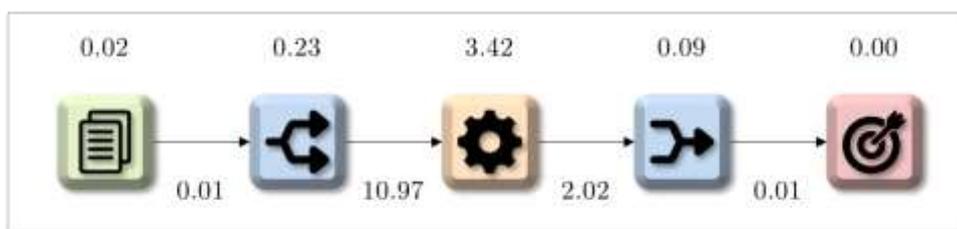


Figure 90 – Example of workflow cost characterization with fixed values

Figure 90 shows an example of workflow characterization with fixed costs (e.g., in seconds). These costs are usually collected by the framework after several executions of the workflow, representing the average duration of each phase and transaction.

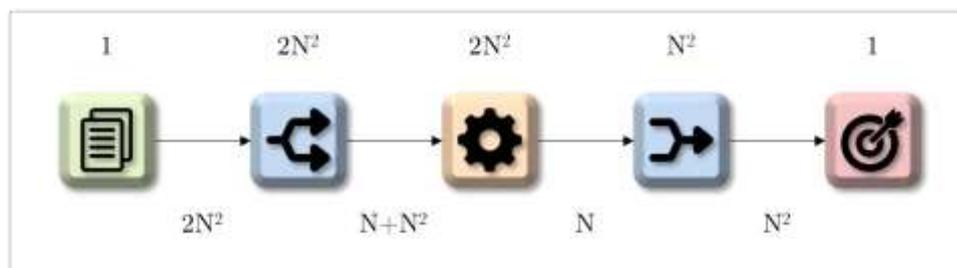


Figure 91 – Example of workflow with costs based on functions

Figure 91 shows an example of workflow characterization with function-based costs. This illustrates the multiplication of square matrices of side N . The access to the source files is considered to be of constant time. Two matrices $N \times N$ are transferred to the Split primitive, leading to a transfer cost proportional to $2N^2$. In this implementation each row of the first matrix is sent to a different worker, while the whole second matrix has to be sent to the worker (thus each worker calculates one row of the resulting matrix). This corresponds to N elements

from the first matrix and N^2 elements from the second. Each worker multiplies the row of the first matrix by each column of the second matrix. This leads to N multiplies and N adds per row x column. This is repeated for N columns, leading a total of $2N^2$ operations. Each worker will produce one row of the final matrix, which corresponds to N elements. The Merge primitive receives N^2 elements to consolidate into a single matrix. Finally, the time to prepare the Output to write the results is considered to be constant.

Note that in the function-based notation the inherent properties of each computation or transfer must not be represented in the costs. For example, the transfer from Input to Split in Figure 91 is a file I/O access while the transfer from Split to Compute is via network (slower). However, the bandwidth for each case differs from system to system, while the amount of data sent among each phase does not differ – for this particular problem the matrix size will always be $N \times N$, the Compute cost will always be proportional to $2N^2$, etc. Moreover, from an algorithm viewpoint, it is simpler for the developer to determine the costs solely based on the inputs to the workflow than having to decide from the beginning which is the source or destination of each chunk of data. In this sense, using the characterization presented in Figure 91 and changing the Input source from a file to a network stream can be done transparently – the only parameter that changes is the bandwidth for reading the source, not the amount of data.

B.4.3 Framework Automation

MPSF provides two levels of automation for the workflow characterization regarding the definition of costs:

- **Verification:** Given a workflow characterization provided by the user, MPSF is able to execute the workflow using incremental steps (e.g., gradually increasing the inputs) to verify that the cost characterization initially provided is accurate enough. In this sense the user must define how the workflow can be executed in this incremental fashion, defining for example which variables must be increased and how they should be increased. The verification guarantees that the costs defined in the characterization will match to a certain extent the actual values observed during the runs. This is fundamental to allow the framework to correctly estimate the duration of the phases and constructs, information that is used in the scheduling and rescheduling activities.
- **Calculation:** Give a workflow characterization provided by the user without the costs, MPSF is able to determine the costs of each phase and between phases by executing the benchmark in an incremental fashion (similar to the verification mechanism). In this case MPSF is able to generate either the fixed value notation or the function-based notation. For fixed value the benchmark is executed using the set of inputs provided by the user and a database is created containing the inputs used and the duration of each phase and transition. For the function-based option MPSF is configured to execute several runs of the benchmark in order to collect a series of points describing the behavior of the application. Then, for each phase and transition these points are used to create a curve that approximates the behavior in each case (either polynomial or other types of functions, including their combinations).

B.4.4 Data Structure Notation

Besides costs as a function of data size, the workflow characterization can also be built based on the actual data structures produced and exchanged among phases. Compute primitives can also specify the number of instructions as a function of the data structures. An example of this use case is observed in Figure 92.

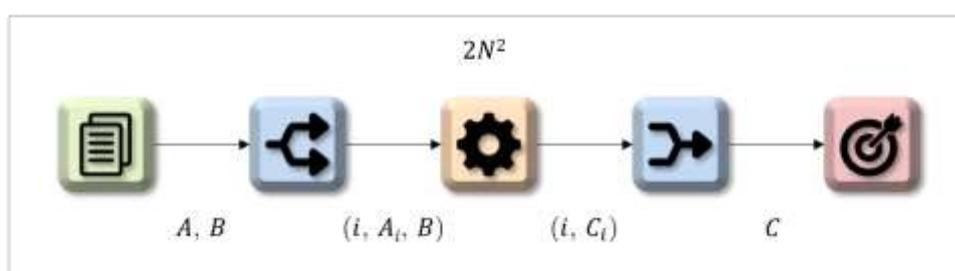


Figure 92 – Example of workflow costs based on the data structures

This example also represents a matrix multiplication construct. Matrices A and B of size $N \times N$ are multiplied to generate matrix C . The framework is able to determine the cost of transferring data for each transition by calculating the amount of data transferred using the data structures and their sizes. The Split produces N tuples containing the row i being processed, the row A_i from matrix A , and the matrix B .

B.5 Summary of Primitives and Transitions

B.5.1 Primitives

	<i>Primitive</i>	Input
	<i>Description</i>	Primitive that defines a phase for data collection. This primitive is used to determine one or more sources of data to be fed to the pipeline. Moreover, an Input element also defines a staging point for the computation, used for the definition of a iteration point and also for restarting purposes (e.g., after a failure).
	<i>Implementation</i>	Specification must define how data is accessed, such as what functions, scripts, or APIs to use. In case of volatile access (e.g., network and memory) with multiple outputs (see <i>branching</i>) the implementation must either locally replicate data for each receiver or create a persistent copy.
	<i>Branching</i>	Multiple outputs; each destination receives a copy of the data. Implementation defines the order that each destination receives the copy or a random order is selected by the framework.
	<i>Cost</i>	Cost to prepare the data (not to transfer, as transfer cost is defined in the phase transition). For example, this cost could be related to finding a particular set of files, executing a database query, or filtering a network stream.

	<i>Primitive</i>	Split
--	------------------	--------------

	<i>Description</i>	Distributes the input to one or more destinations. Typical use case is to distribute Input data to several worker nodes.
	<i>Implementation</i>	Implementation must define how data is split and sent to destinations.
	<i>Branching</i>	This primitive supports multiple outputs by defining logic on how to do the branching, filtering and/or selecting input data to send to a particular destination. This is useful, to implement conditional branches and loops.
	<i>Cost</i>	The cost (compute time) to execute the split function and prepare to send to destinations.

	<i>Primitive</i>	Compute
	<i>Description</i>	Executes one or more computation phases over the input, generating a set of outputs. The Compute primitive is also used to define reusable workflows (<i>nesting</i>).
	<i>Implementation</i>	Specification must reference to one or more external applications that implement the logic inherent to the primitive.
	<i>Branching</i>	Supports multiple outputs (each destination receives a copy of the results).
	<i>Cost</i>	The cost (compute time) to execute the functions or constructs defined by the primitive.

	<i>Primitive</i>	Merge
--	------------------	--------------

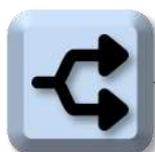
	<i>Description</i>	Consolidates a set of inputs into a single output.
	<i>Implementation</i>	Specification must define the logic used to merge the inputs.
	<i>Branching</i>	Supports multiple inputs. Internal logic must define how these inputs are merged together.
	<i>Cost</i>	The cost (compute time) to execute the merge operations.

	<i>Primitive</i>	Output
	<i>Description</i>	Defines a final or preliminary result point for the workflow, typically used to determine the end of a workflow or an instrumentation point.
	<i>Implementation</i>	Specification must define how the outputs are stored in the system (e.g., by writing the contents to a file).
	<i>Branching</i>	Support to multiple outputs, each destination receives a copy of the output.
	<i>Cost</i>	The cost (compute time) to store the outputs using the I/O method defined in the implementation.

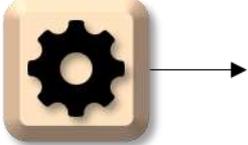
B.5.2 Transitions

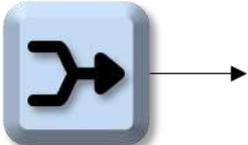


<i>Description</i>	Transition used to connect an Input to another element that receives the data read using the method specified in the primitive.
<i>Cost</i>	Cost of sending the data from the input actual location to the destination. Typical usage is to connect the Input to a Split element. In this case the cost is usually relative to the time spent on accessing the Input (e.g., from a file) and connecting it to the Split primitive (e.g., the cost to access the file and then write the contents on memory).



<i>Description</i>	Transition used to connect a Split to its destinations.
<i>Cost</i>	Cost of sending data from the Split primitive to its destinations. Typical use case is to connect the Split to a Compute element. In this case the data is sent to the worker nodes and containers, thus the cost in this case is the time to send data over the network, for instance.

	
<i>Description</i>	Transition used to transport the outputs produced by a Compute primitive to the destinations.
<i>Cost</i>	Cost of sending the results to the destinations. Typical utilization involves the time to write the output to the memory, or to send this output via network to a Merge primitive.

	
<i>Description</i>	Transition for moving the results of the Merge operation to the destinations.
<i>Cost</i>	Cost of sending the results of the Merge to the destination, usually an Output primitive.

	
<i>Description</i>	Transition used to write the Output results to one or more destinations further in the workflow pipeline.
<i>Cost</i>	Cost of writing the results to the destinations.

B.6 Mapping Primitives

An important component of the workflow characterization via the proposed primitives is the integration of these primitives to their actual implementation (i.e., code and programs that executes each phase). The typical mapping between primitives and their implementation is done via independent pieces of software, each piece being responsible for a specific phase of the workflow. Figure 93 depicts the representation of a workflow that implements matrix multiplication.

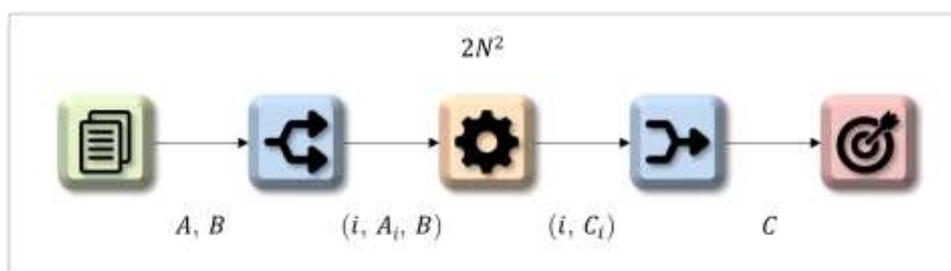


Figure 93 – Exemplar matrix multiplication workflow

This workflow is using the data structure notation to represent data movement and costs. Matrices A and B , $N \times N$, are sent to the splitter, which generates N tuples (chunks) containing the index of a row from A , the row A , and the matrix B . The row (or vector) A_i is then multiplied by each column of B . This represents N multiplications plus N sums (counting the initial sum of zero plus the first multiplication), leading to $2N$ operations. This is repeated for every column of B , leading to $2N^2$ operations. Each chunk leads to the generation of a vector with N elements described by the tuple containing the row i of the result matrix C . These chunks are joined by the merger element, which finally generates the result matrix.

The mapping of these primitives could be implemented as depicted in Figure 94, wherein input and output are mapped to files and the inner elements are mapped to actual programs.

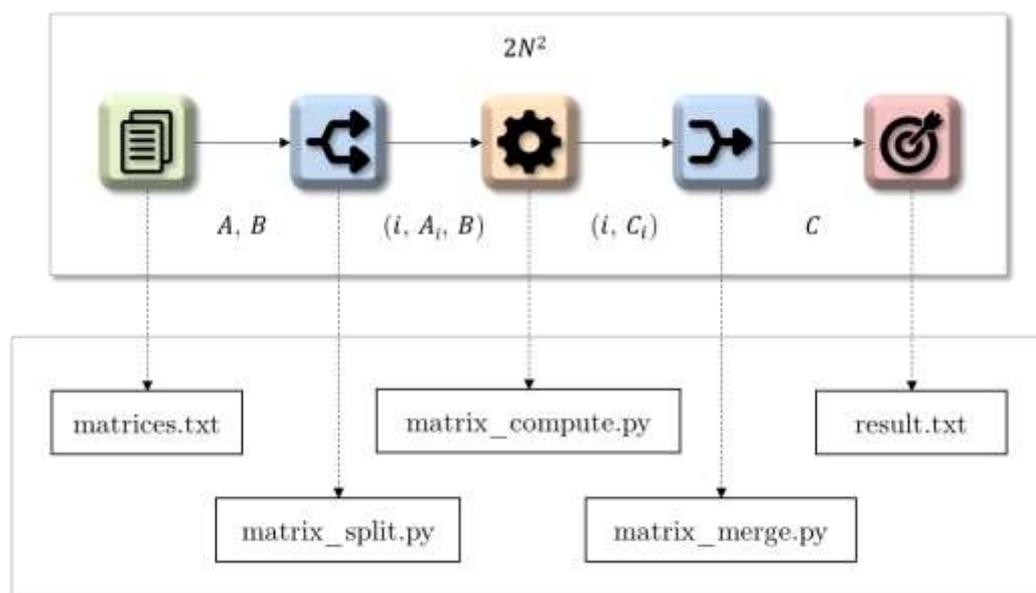


Figure 94 – Example of mapping of primitives into implementation

In this example the split, compute, and merge elements are associated to Python programs (extension `.py`). The workflow could be mapped as follows:

- **Input:** A text file containing the two matrices to be used in the calculation.

File: `matrices.txt`

```
1.0  1.0  1.0
2.0  2.0  2.0
3.0  3.0  3.0
```

```
1.0  2.0  3.0
1.0  2.0  3.0
1.0  2.0  3.0
```

This file represents two matrices separated by an empty line. Elements of the same row are separated by simple spaces.

- **Split:** A program that reads the matrices and separates the first matrix into rows, distributing the data accordingly:

Program: matrix_split.py

```
1   import sys
2
3   args = sys.argv[1:]
4   input_name = args[0]
5   file = open(input_name, 'r')
6
7   A = []
8   B = []
9
10  for line in file:
11      if not line: break
12      row = map(float, line.split(' '))
13      A.append(row)
14
15  for line in file:
16      row = map(float, line.split(' '))
17      B.append(row)
18
19  file.close()
20
21  counter = 0
22  for row_a in A:
23      output_name = 'chunk_%d' % (counter)
24      file = open(output_name, 'w')
25
26      file.write('%s\n' % ' '.join(row_a))
27      file.write('\n')
28
29      for row_b in B:
30          file.write('%s\n' % ' '.join(row_b))
31
32      file.close()
33      counter += 1
```

Description of the program as follows:

- Lines 1 to 5: Read the name of the file that contains the input matrices, then open this file to be read.
- Lines 7 and 8: Prepare the matrices as lists (of rows) that will receive lists (with column elements).
- Lines 10 to 13: Read the first lines of the input file to construct the first matrix, A . Line 11 defines the rule (blank line) to stop reading elements for A and start reading elements for B .
- Lines 15 to 17: Read the subsequent lines of the input file to construct the second matrix, B .
- Line 19: Close the input file as it is not necessary anymore.
- Line 21: Prepare the counter that controls the index i being handled.
- Lines 22 to 24: Iterate over rows of A and create an output file that includes the index i (controlled by variable `counter`) and open this file for writing.
- Lines 26 and 27: Write the row i of matrix A and add a new line to separate it from matrix B .
- Lines 29 and 30: Write matrix B to the output file.
- Lines 32 and 33: Close the output file and iterate the counter.

At the end of execution the result is N files (N being the side size of the matrices A and B) containing the i -th row from A and the matrix B .

- **Compute:** A program that multiplies the i -th row from A by B .

```
Program: matrix_compute.py

1   import sys
2
3   args = sys.argv[1:]
4   chunk_name = args[0]
5   file = open(chunk_name, 'r')
6
7   a = []
8   B = []
9
10  line = file.readline()
11  a = map(float, line.split(' '))
12  line = file.readline()
13
14  for line in file:
15      row = map(float, line.split(' '))
16      B.append(row)
17
18  file.close()
19
20  c = [0.0] * len(a)
21  for j in range(len(a)):
22      for i in range(len(B[j])):
23          c[j] += a[j] * B[i][j]
24
25  output_name = '%s_output' % (chunk_name)
26  file = open(output_name, 'w')
27  file.write('%s\n' % (' '.join(c)))
28  file.close()
```

Description of the program as follows:

- Lines 1 to 5: Read the name of the input chunk file and open this file to be read.
 - Lines 7 and 8: Initialize vector a and matrix B .
 - Lines 10 to 12: Read vector a from input file.
 - Lines 14 to 18: Read matrix B , then close input file.
 - Lines 20 to 23: Initialize result vector c with zeroes, calculate each of its elements (by multiplying a_j by each element B_{ij}).
 - Lines 25 to 28: Write results to output file.
- **Merge:** A program that receives all chunks and create the final matrix C .

Program: matrix_merge.py

```

1   import os
2
3   pre = 'chunk_'
4   pos = '_output'
5   rows = {}
6
7   for name in os.listdir('.'):
8       if pre in name and pos in name:
9           index = int(name.replace(pre, '').replace(pos, ''))
10
11          file = open(name, 'r')
12          row = map(float, file.readline().split(' '))
13          rows[index] = row
14
15  result_name = 'result.txt'
16  file = open(result_name, 'w')
17  for row in sorted(rows):
18      file.write('%s\n' % (' '.join(rows[row])))
19  file.close()

```

Description of the program as follows:

- Lines 1 to 5: Prepare rules to identify files that contain result chunks (files starting with `chunk_` and ending with `_output`), and prepare the `rows` variable that will store the rows read from these files.
 - Lines 7 to 9: Sweep over files from directory and identify ones that represent result chunks. Get index of row by parsing the file name.
 - Lines 11 to 13: Read row from each file and add it to the `rows` variable, identifying the row index in this variable.
 - Lines 15 to 19: Prepare result file and print rows to this file in order.
- **Output:** The output element can be implemented as a simple file with the name of the file that will contain the results, or it can also be a command line argument to be read when the merge component is executed.

This example shows how a matrix multiplication application can be executed via the framework by providing the adequate mapping to MPSF primitives. As observed in the code excerpts provided, the implementation of the programs, especially the matrix multiplication code, was slightly adapted to how the framework distributes tasks. In this way MPSF is able to distribute tasks with the finest granularity possible, in the sense that tasks can be distributed at cloud, cluster, node, and container level. In contrast, if the developer already has a program that implements the algorithm and does not want to modify it, the framework is still able to distribute tasks in a coarse-granularity approach. For instance, if multiple matrices have to be multiplied, the original matrix multiplication code is maintained and the framework distribute tasks so that each task is responsible for one matrix multiplication operation.

B.7 Comparison to Literature

In terms of workflow modeling the literature provides two main methods (and extremes):

- The simple DAG used by most of the proposed works in the field, especially for cloud resource management, such as (PANDEY et al., 2010), (CHEN; ZHANG, 2009), (RODRIGUEZ; BUYYA, 2014), (POOLA et al., 2014a), and many others.
- The extensive proposal of components by the Workflow Patterns Initiative (WORKFLOW, 2016). The current specification has over 100 patterns for control-flow, data, and resource perspectives. Another comparable specification is the one from the Business Process Model and Notation (BPMN, 2016), with over 110 patterns and elements.

The first approach is extremely simplistic, usually modeling workflows with DAGs wherein phases are represented by vertices of the DAG with a cost in the form of a fixed value. Phase transitions (edges) represent the data movement, which in most cases is not directly considered in the performance models. While this approach might be enough to model basic interactions between phases and basic mapping from these phases to actual execution, this approach lacks depth to adequately characterize the workflows, their internal interactions, and the inherent costs of these interactions.

The second approach is much more comprehensive, with numerous elements to represent every possible interaction and function in the system. However, mapping actual implementation to workflow description is usually a problem of identifying which elements of the framework are comparable to the code implemented. Having a very large number of

elements is overwhelming from a practical viewpoint, as a lot of time will be consumed just to find the right match – and integration is still pending.

The approach adopted and presented in this section is based on the idea proposed by (BHARATHI et al., 2008) of using a subset of functions of these existing large frameworks. Bharathi et al. proposed a method for characterization of scientific workflows based on five primitives: process, pipeline, data distribution, data aggregation, and data redistribution. The model used by MPSF goes beyond in terms of simplification, defining only elements that either split, process, or merge data. Two other elements were then added to represent the beginning and end of the workflow. As an auxiliary component, these two elements can also be used to represent partial phases in the workflow. Compared to the model from (BHARATHI et al., 2008), the pipeline construct is simply a sequence of processes and the redistribution construct is a combination of merge (aggregation) and split (distribution). All these constructs are supported by MPSF while providing a simpler set of operations that are easy to remember and also to correlate with existing software, effectively easing the process of integrating existing software into MPSF, or the process of creating new software to be integrated to MPSF.

APPENDIX C – RESOURCE CHARACTERIZATION

The resource characterization is used to organize and describe the resources that compose the cloud infrastructure. Resources include compute power (CPU, GPU, accelerators, and others), memory, storage, and networking. The next subsections provide details on how the resources are organized internally by the framework, what are the properties that are included in their specification, and how the execution of workflows is conducted from logical and practical viewpoints.

C.1 Resource Hierarchy

Figure 95 presents an overview of the basic hierarchy of cloud resources.

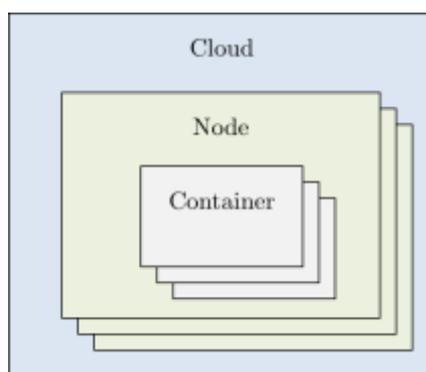


Figure 95 – Basic internal hierarchy of a cloud and its resources

The quintessential element in this hierarchy is the cloud node, usually represented by a physical machine. The cloud infrastructure is a composition of several nodes that form a pool of shared resources used for executing the workflows managed by the framework.

The atomic element in this hierarchy is the cloud container (e.g., a virtual machine or an actual container, such as Docker). Cloud containers are created within a node to subdivide the node resources to facilitate resource distribution across workflows and tasks due to their finer granularity. The actual implementation of these containers might vary from cloud to cloud and is inherently transparent to the framework. Whether they are created based on virtual machines on top of a hypervisor or based on Linux or Docker containers, for MPSF the only requirement for both nodes and containers is that each element must be accessible via some property or method (e.g., via IP address, or via a specialized controller implemented by the cloud).

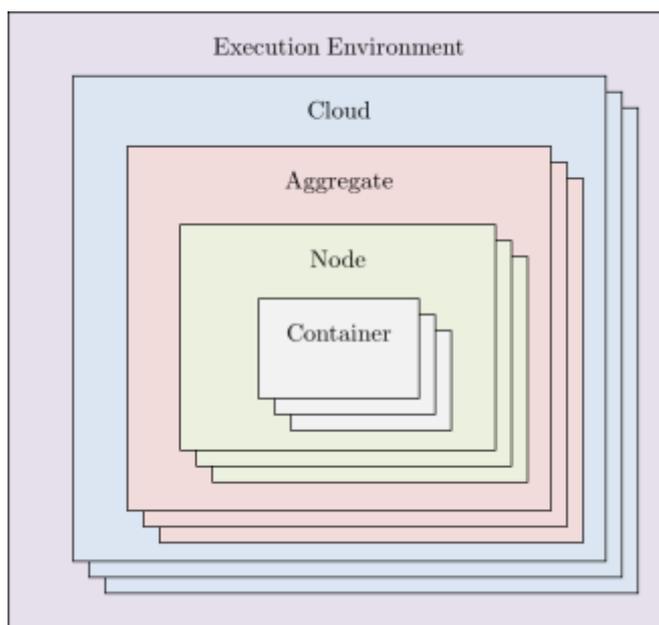


Figure 96 – Complete architecture including the Execution Environment and its components

Figure 96 presents the full hierarchy of resources considered in the framework. Two hierarchical tiers were added to the basic model presented on Figure 95:

- **Execution Environment:** Comprises the whole set of resources from multiple clouds, thus addressing multicloud scenarios (including hybrid clouds). The Execution Environment represents the complete set of resources that are available for workflow execution and, consequently, for management and scheduling by the framework.
- **Aggregate:** Represents the combination of several nodes. An aggregate typically represents a set of nodes under the same network or subnetwork. The specification of an aggregate might be provided by the cloud administrator (responsible for configuring the execution environment) or automatically obtained by the framework. In the latter case the framework executes a series of tests to determine which nodes belong to the same aggregate in terms of network hops and communication distance (e.g., measured bandwidth).

These tiers define the complete hierarchy managed by MPSF. Execution environments comprise the resources of one or more cloud environments, while aggregates allow the combination of the resources from several nodes.

C.2 Execution Spaces

Each layer of the resource hierarchy is the representation of a different physical organization tier in the execution environment. The logical counterpart of the resource hierarchy is the execution space, which defines a group of resources comprising one or more layers of the hierarchy that share the same set of goals and requirements. The execution space is an abstraction for the working group or set of resources dedicated to the execution of a workflow or a set of workflows with interrelated properties. While the typical configuration of execution space involves the allocation of containers for possibly different hierarchical levels,

it is also possible to allocate a whole aggregate or even cloud, for instance, as part of an execution space.

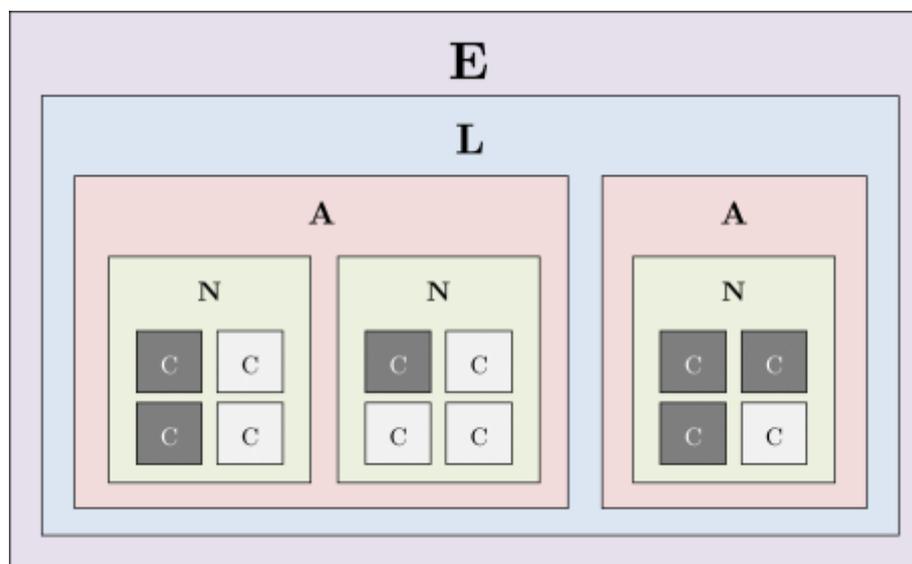


Figure 97 – Example of execution space

Figure 97 shows an example of execution space composed by six containers (darker highlighting; containers identified as *C*) from three different nodes (identified as *N*). Two nodes belong to a same aggregate (*A*) while the other node belongs to a second aggregate. In terms of physical topology this could be an example of nodes from different networks. The aggregates belong to the same cloud (*L*), which is the only cloud that composes the execution environment (*E*). In this sense the execution space is a logical structure that allows the framework to transparently allocate and manage resources from different physical layers or different physical elements in the hierarchy.

The creation of an execution space is controlled via two methods: user configuration and framework configuration. The user configuration (i.e., of a workflow) might define how execution spaces should be created in order to reflect certain properties and requirements of the

workflow. For example, if a workflow has two main branches and one requires special security features (e.g., a certain library or a certain security device only available in specific nodes), the workflow specification must define the distinct execution spaces required by its phases. During resource allocation the framework interprets the properties and requirements and translates them into node allocation, thus creating the adequate execution spaces.

The other method that controls the creation of execution spaces is the configuration of the framework itself. For instance, the framework can be configured to create simple execution spaces by leveraging the resources of the same aggregate (due to lower network cost) if possible – else it starts allocating nodes and containers from other aggregates.

The execution space defines a set of shared properties among all nodes and containers associated to it, as well as to all tasks and processes in execution. In other words, the execution space defines a shared environment for the execution of the workflow and its phases. This facilitates the development of applications in the sense that the developer does not have to be aware of the locality of a certain resource or file, for example. By accessing data through the execution space the locality is handled by the framework, effectively providing a transparent programming and runtime interface for applications and workflow phases.

C.3 Segmentation and Logical Hierarchies

The combination of execution spaces to the original resource hierarchy requires the definition of a new method to organize resources that consider this new logical layer. This is provided by the segmentation of resources comprising the initial hierarchy (physical plus virtual resources) to the execution spaces (logical subdivision), leading to the creation of logical hierarchies.

The logical hierarchies are used to organize resources to provide support for multi-tier branching and splitting – sequential splits that require proportional support in terms of hierarchical layers. For instance, workflows that implement a sequence of branches or splits (and the mirrored merges) require multiple hierarchical levels to adequately distribute data or compute load. Figure 98 presents an example of multiple sequential splits.

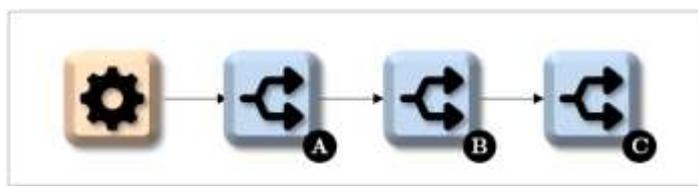


Figure 98 – Example of multiple sequential splits

In Figure 98 each split can be distributed in a specific hierarchical layer. For example, Split A can be distributed in the layer of nodes (each chunk is distributed to one node), Split B can be distributed in the layer of containers (each chunk is distributed to one container of the same node), and Split C can be distributed in the layer of execution spaces (each chunk is distributed to one execution space composed by a set of containers of the same node).

The segmentation and creation of logical hierarchies is essential to implement multi-tiering and integration to other distributed computing frameworks, such as MPI⁸ and OpenMP⁹. MPI, for instance, allows the implementation of distributed computing via message passing throughout several nodes. An MPI program can be distributed to nodes via MPI and then the load can be redistributed inside each node to different containers via MPSF. In parallel, OpenMP implements threading, which is parallelization in the node to container level. In this

⁸ <https://www.open-mpi.org/>

⁹ <http://openmp.org/wp/>

sense, MPSF can be used in a higher tier of the hierarchy, distributing several OpenMP tasks across a cluster. All tasks are supervised by MPSF and the execution spaces guarantee the isolation of execution and communication for each set of related tasks.

C.4 Resource Properties

Resources comprise several distinct types of hardware and software properties of a system. However, the main objective of the resource characterization in MPSF is to provide a description of the execution environment in order to map tasks and workflows onto the resource hierarchy. Consequently, this description essentially focuses first on the performance capabilities of these resources, and then on the capabilities related to specific requirements (e.g., computing capabilities and security features).

Resource properties are defined in two abstraction levels: application-independent and application-dependent. Application-independent properties are measured using benchmarks, provided by the cloud administrator, or collected from system tools. Examples of these properties are presented in Table 36.

Table 36 – Examples of resource properties

<i>Element</i>	<i>Performance Properties</i>	<i>Requirement Properties</i>
Node	<ul style="list-style-type: none"> – Compute capacity (FLOPS) – Memory capacity (GB) – Memory bandwidth (GB/s) – Memory latency (ns) – File system capacity (GB) – File system bandwidth (MB/s) – File system latency (ms) 	<ul style="list-style-type: none"> – Presence of Graphics Processing Unit – Presence of Trusted Platform Module – Support to security features/libraries – Support to distributed programming frameworks (e.g., MPI, OpenMP)
Network	<ul style="list-style-type: none"> – Link bandwidth (GB/s) – Link latency (ms) 	<ul style="list-style-type: none"> – Support to security properties (e.g., firewall, DoS protection) – Support to Software Defined Network

This table presents a distinction between performance properties and requirement properties. Performance properties are related to the performance measurements that are used to calculate the duration of phases and data movement during the execution of the workflow. In contrast, requirement properties represent requirements defined by the user for the execution of the workflow, or requirements inherent to the workflow. For instance, the user might request that the workflow must be executed in a specific execution space in a specific set of nodes that are compliant to a security conditions (e.g., be part of a certain cluster due to non-disclosure and confidentiality agreements). Or, the workflow might specify that it requires GPUs for the execution.

Performance properties are independent of the workflow or application being executed on the cloud. In contrast, the application-dependent properties are measured as a function of the application being executed on the infrastructure. In particular, each phase of a workflow might have a different performance property, and the value of the property might vary from node to node. Application-dependent properties are used to compute the expected duration of a particular workflow phase executed on a specific node or set of nodes. In this sense, the property that represents the cornerstone of this calculation is the data processing rate for a particular workflow phase and a particular node or container. This processing rate is defined as a multiple of bytes per time (e.g., MB/s) and represents the amount of data that is processed over a period. This information is then used to calculate the expected duration of this phase running on this node. For instance, if the data rate for a container C for a phase P is 10 MB/s and the input chunk received by this container has 1,000 GB, the expected duration for this computation (phase) is 100 seconds. The processing rate combined to the transfer rate between nodes

represent the fundamental properties used in the calculations for input distribution, resource allocation, and workflow scheduling.

C.5 Comparison to Literature

The identified works in the literature usually provide some form of classification of resources. (JENNINGS; STADLER, 2015) distinguishes compute, networking, storage, and power as resource types. (MANVI; SHYAM, 2014) divides resources in physical and logical. (SINGH; CHANA, 2016) provides a hierarchical organization of resources, as presented in Figure 99.

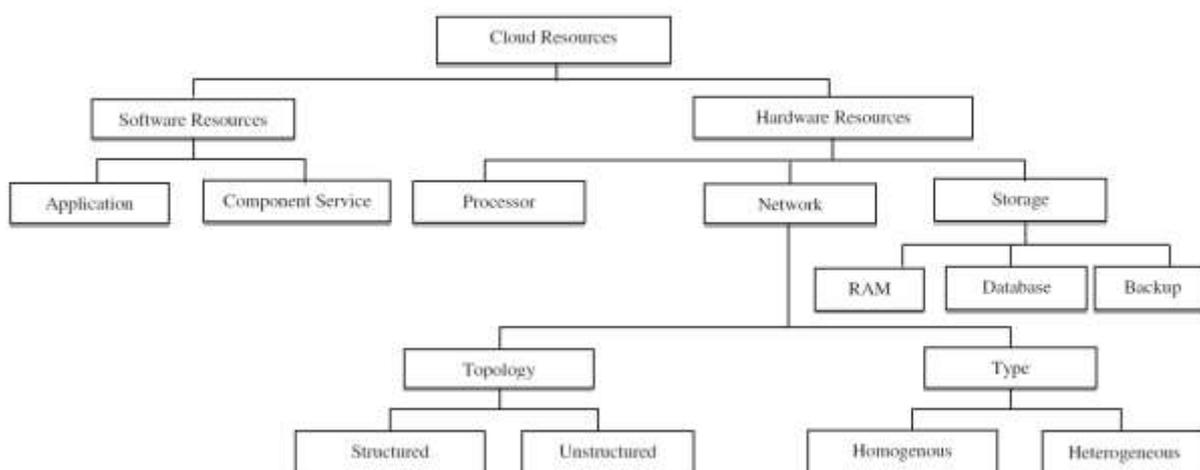


Figure 99 – Classification of resources as defined by (SINGH; CHANA, 2016)

Even the classification proposed by (SINGH; CHANA, 2016) does not address the hierarchy of computing resources, such as organization tiers between different layers of computation (e.g., cluster, nodes, containers). This hierarchy is used to further organize the execution of workflow tasks, as well as defining working spaces for different sets of tasks and workflows.

APPENDIX D – RELIABILITY MODEL

An essential aspect of cloud resource management is to distribute tasks and allocate resources considering the reliability aspects of the system. Most cloud resource management solutions address reliability without basing the operations in measurable metrics related to properties such as failure rates. Several authors and works highlight the challenges and potential gaps in terms of cloud management and cloud resource management in terms of reliability. (BALA; CHANA, 2011) state that workflow scheduling is one of the key issues in the management of workflow execution in cloud environments and that existing scheduling algorithms (at least at that time) did not consider reliability and availability aspects in the cloud environment. (SINGH; CHANA, 2016) directly addressed this issue by stating that the hardware layer must be reliable before allocating resources. While several subsequent works addressed these aspects, there still are gaps in the methodology. For instance, (CHEN; ZHANG, 2009) implement a solution that considers a reliability factor but there is no explicit model on how to calculate this factor based on actual hardware and software reliability related metrics, such as hardware failure and software interruption rates.

(FARD et al., 2012) define a reliability factor by assuming a statistically independent constant failure rate, but this rate only reflects the probability of successful completion of a task – there is no clear connection between this concept and a factual and measurable metric from hardware and software point of view. (HAKEM; BUTELLE, 2007) also propose a reliability-based resource allocation solution by defining a reliability model divided in processor, link, and

system. While the model is based on exponential distributions which could be related to metrics such as mean time between failures (MTBF) and failure in time (FIT).

Other solutions such as the one from (POOLA et al., 2014) use reliability-related methods such as checkpointing to decrease application failures, but in this particular case, for instance, the performance implications of having these mechanisms is not fully appreciated. The I/O cost in terms of storage and time to implement checkpointing are far from negligible (APEX, 2016). Still on reliability, (WANG; CHEN, 2012) state that the main two strategies to calculate reliability factors is to either establish a reputation threshold or to treat nodes independently and multiply their probability of success.

Still, the reliability approach proposed by the authors does not address measurable metrics to calculate these factors. Moreover, on one side there are the solutions only address failures after their occurrence, not before. For instance, (POOLA et al., 2014a) uses checkpointing to recover from failures but there is no mechanism in place to calculate the probability of failures and attempt to avoid nodes with higher probability of failure, or at least designate a smaller portion of tasks to this node. On the other side, solutions calculate reliability factors based on theoretical metrics that might not reflect the specificities of each node and there are no clear mechanism to combine prevention and recovery. In that sense, (HWANG; KESSELMAN, 2003) provide a deeper analysis of fault-tolerance techniques for grid computing that could be applied to cloud computing. The authors clearly state that the requirements for implementing failure recovery mechanisms on grids comprise support for diverse failure handling strategies, separation of failure handling policies from application codes, and user-defined exception handling. In terms of task-level failure handling techniques the authors consider retrying (straightforward and potentially least efficient of the enlisted

techniques), replication (replicas running on different resources), and checkpointing. Checkpointing is actively used in real scientific scenarios while replication usually leads to prohibitive costs (APEX, 2016), as in several cases running one replica is expensive enough in terms of resource demand. In addition, in terms of workflow-level failure handling, the authors propose mechanisms such as alternative task (try a different implementation when available), workflow-level redundancy, and user-defined exceptions that are able to fallback to reliable failure handling. In terms of evaluation the authors propose parameters such as failure-free execution time, failure rates, downtime, recovery time, checkpointing overhead, among others. These are measurable metrics that can be used to model and represent the failure behavior of systems and workflows. These are crucial metrics that must be used to determine the probability of failure of a node, for example.

MPSF proposes a model that is based on measurable properties and uses the results of the model to augment the load distribution. Reliability refers to the probability that a component or system will continue to perform its intended function under stated operating conditions (ELLERMAN, 2012). The reliability model provides tools to augment the load distribution and consequently node utilization by calculating distribution factors based on measurable metrics related to the reliability of the nodes and of the system. This section presents the metrics and mechanisms used to implement this model, as well as how this model is integrated to the remaining components of the resource management framework.

D.1 Reliability Concepts and Metrics

The calculation of reliability values of integrated circuits is typically based on the Arrhenius High Temperature Operating Life (HTOL), which stresses the component at an

elevated temperature and high voltage for a period of time (ELLERMAN, 2012). The most common method to calculate the reliability level of a component is based on parameters such as Failures In Time (FIT), Mean Time To Failure (MTTF), and Mean Time Between Failures (MTBF). The mean time to failure is simply the average time a component will function before it fails. The mean time between failures can be used either interchangeably with MTTF or it can consider the Mean Time To Repair (MTTR) of the component. In this case, the following relation is valid:

Equation 44 – Relation between MTBF, MTTF, and MTTR

$$\text{MTBF} = \text{MTTF} + \text{MTTR}$$

The time to failure is usually expressed as a function of a failure rate λ , which is a measure of failures per unit of time. The useful life failure rate is based on exponential life distribution (VIGRASS, 2001). The failure rate typically decreases over early life, then stabilizes until wear-out shows an increasing failure rate, which is beyond useful life. This behavior is characterized by the bathtub curve, as presented in Figure 100.

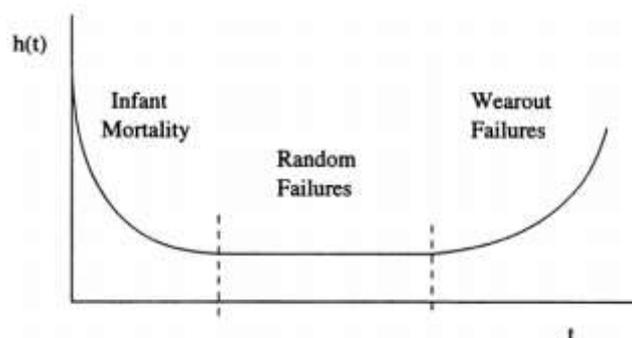


Figure 100 – Bathtub curve (KLUTKE; KIESSLER; WORTMAN, 2003)

The infant mortality represents early failures observed in components. After this initial failures there is usually a stabilization of failures at a lower degree, and then finally failure rates start to increase again due to useful life.

In parallel, the FIT value is a measure of the failure rate in 10^9 device hours. In other words, if a device has 1 FIT = 1 failure in 10^9 device hours. The MTTF is derived from these values by representing the life distribution for the population of devices under operation or expected lifetime of an individual component. The following expressions are observed:

Equation 45 – Relation between MTTF and λ

$$\text{MTTF} = \frac{1}{\lambda}$$

In this sense, the MTTF is the time where 63.2% of the population has failed, according to the exponential distribution. For example, if $\lambda = 10$ FITs, then MTTF = 100 million hours. These parameters are combined to probabilistic models to allow the calculation of failure expectancy for a certain period of time, for instance. In this sense, one model typically used is the Poisson distribution:

Equation 46 – Poisson distribution for reliability calculations

$$P(k, t) = \frac{\left(\frac{t}{\lambda}\right)^k e^{-t/\lambda}}{k!}$$

Where $P(k, t)$ is the probability of failure occurring k times in time t . The probability of zero failures in time t is calculated as:

Equation 47 – Probability of zero failures

$$P(0, t) = \frac{\left(\frac{t}{\lambda}\right)^k e^{-t/\lambda}}{k!} = \frac{\left(\frac{t}{\lambda}\right)^0 e^{-t/\lambda}}{0!} = e^{-t/\lambda}$$

And the probability of occurring at least one failure:

Equation 48 – Probability of at least one failure

$$P(k > 0, t) = P(k \geq 0, t) - P(0, t) = 1 - e^{-t/\lambda}$$

Note that λ simply is the expected value of the Poisson distribution. In other words, the failure rate is the expected value of a long-run average value of repetitions of the failure experiments. Moreover, the variable t is usually expressed as a factor of the interval λ . For instance, if $\lambda = 5 h$ and $t = 1 h$, then $\lambda t = 1/5 = 0.2$.

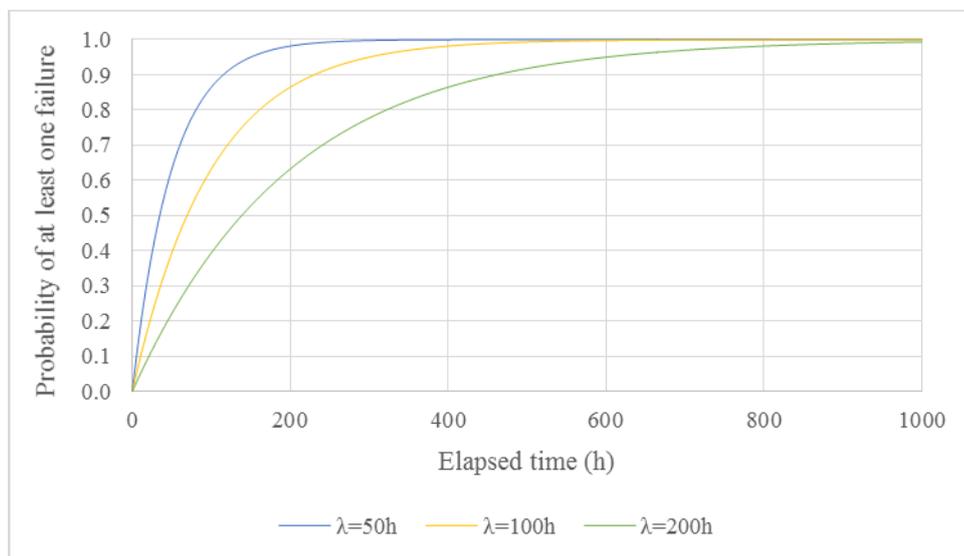


Figure 101 – Probability of at least one failure for different failure rates

Figure 101 shows three curves with different failure rates plotted over a period of time. For $\lambda = 50h$, $P(t = 50h) = 0.6321$ and the $P(t) \geq 0.9$ for $t = 95h$. For $\lambda = 100h$,

$P(t = 100h) = 0.6321$ and $P(t) \geq 0.9$ for $t = 190h$. Finally, for $\lambda = 200h$, $P(t = 200h) = 0.6321$ and $P(t) \geq 0.9$ for $t = 380h$.

In addition, Figure 102 shows the probabilities of different number of failures based on a failure rate of $\lambda = 50$ h. For $t = 35$ h the probability of zero failure is around 50% while the probability of one failure already is around 35%. For $t = 50$ h, which is the failure rate, the probability of zero and one failure are the same, around 37%. For $t = 100$ h the probability of zero failure is 14% while the probabilities of one and two failures are around 27% each. Finally, for $t > 145$ h the probability of at least one failure is higher than 95%.

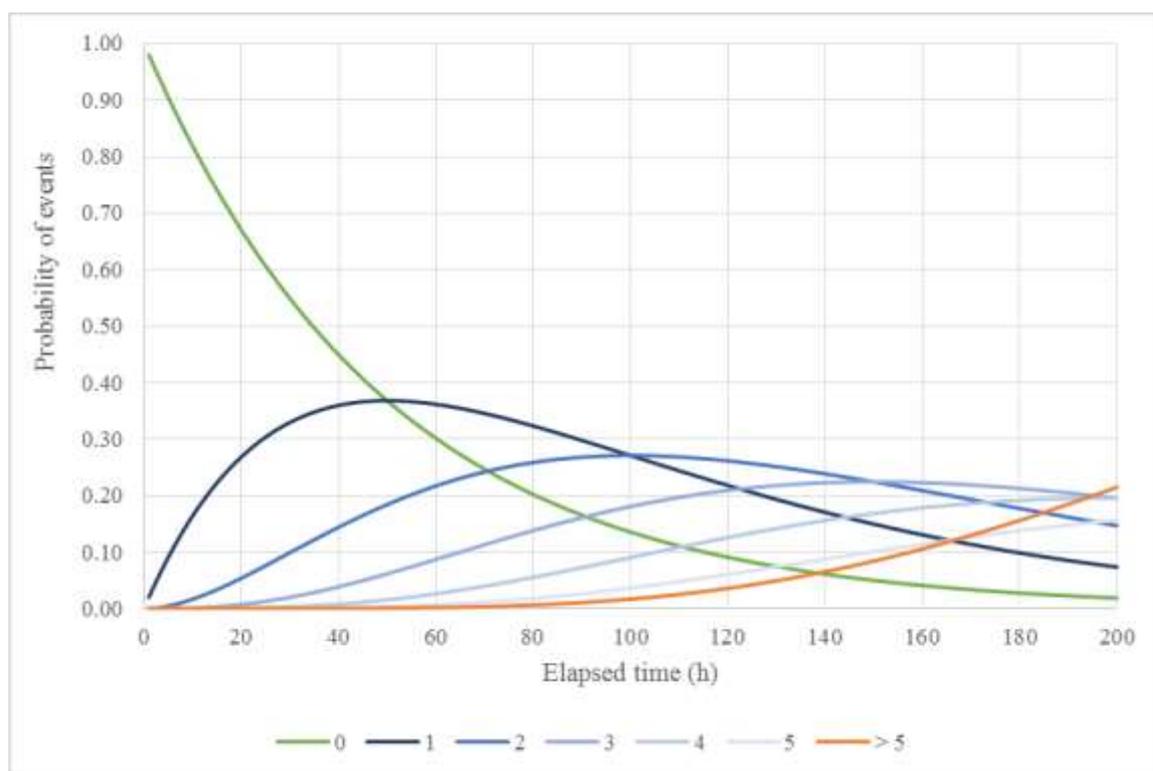


Figure 102 – Probabilities of zero to five failures and probability of more than 5 failures

D.2 Probabilistic Reliability Model for Workflow Execution

The reliability model used in MPSF for workflow execution determines the probability of node failures and then uses this information either to allocate the tasks over an execution space or to perform changes in the execution spaces. There is an intrinsic tradeoff between redistributing tasks and final performance. Redistribution means avoiding sending tasks to a specific node that is expected to fail, or sending smaller tasks that can be rapidly recovered or re-executed in case of failure. On the other hand, any redistribution activities that lead to avoiding a node that in the end does not fail, leads to performance losses. Consequently, it is important to balance the redistribution activities and their effectiveness in terms of increasing overall performance.

Reliability metrics are obtained either via specification or measurements. Specification metrics are typically provided by the manufacturer or measured by third-parties. Backblaze¹⁰, for instance, provides an extensive analysis of several hard disks in terms of reliability, aggregating information about over 68 thousand drives (BACKBLAZE, 2016).

Measured metrics are internally defined by MPSF based on the behavior of the system. These failures account both for hardware failures and also software failures, during the execution of specific workflows and workflow phases. This information is combined to the metrics obtained via resource specification to calculate the final failure probabilities for augmenting the scheduling decision.

Finally, the actual integration between these metrics and the resource management activities is executed via thresholds that determine when certain policies must be executed. MPSF defines the following levels of policies regarding reliability:

¹⁰ <https://www.backblaze.com/blog/hard-drive-failure-rates-q2-2016/>

1. **Do nothing.** The node/container reliability is still acceptable and it might continue to receive new tasks and workflows of any duration, complexity, and cost.
2. **Avoid mission-critical and expensive tasks.** The node/container is acceptable but not completely reliable either due to the identification of error messages that typically preclude more serious failures, or due to service time. Mission-critical tasks or expensive tasks (e.g., with long duration) should be avoided unless all other resources are not available.
3. **Cheap tasks and auxiliary functionalities only.** The reliability of the node/container is severely compromised, either due to the identification of metrics that lead to potential failures or due to advanced service time. This node/container should only execute cheap tasks whose repetition is not extremely costly or run auxiliary functionalities such as replaying tasks for redundancy purposes.
4. **Auxiliary functionalities only.** The node/container is about to fail. This condition is detected via system tests, such as detection of bad blocks and sectors in the hard drive, or numerous memory and disk issues. The node/container should be used only for auxiliary tasks, such as recovery-related, relaying information, or side-tasks related to processing workflows.
5. **Avoid resource.** The node/container is beyond the failure rate, typically measured with a probability of over 95% of failure. Resource must be avoided and possibly proactively replaced, if possible.

Each of these reliability levels is associated to a percentage, except for the last level which is usually fixed to 90% to 95% of failure probability.

D.3 Task Cost from Reliability Perspective

The cost of a task from reliability perspective is essentially a matter of time spent on the task and potential time loss in case of failure. For instance, a task with makespan of 20 minutes that observes a node failure at $t = 15 \text{ min}$ leads to a time loss of 15 minutes, but the maximum time loss for this particular task is 20 minutes. On the other hand, a task with makespan of four hours and a failure at $t = 15 \text{ min}$ also leads to a time loss of 15, but the loss potential for this task is much higher due to its makespan. The basic calculation to determine the potential loss of a task is based on the probability of failure during the execution of the task. Then, this probability is compared to a threshold that determines if the task should be relocated or not.

In a modern system with a failure rate of $\lambda = 24 \text{ h}$ (APEX, 2016), if a task is allocated in the first four hours of the operation of the system, the probability of at least one failure is lower than 20%. However, if the same task is allocated at $t = 720 \text{ min}$ (30 h), there is a 40% chance of failure of this system. Note that if the task is 60 min long, in the first case the probability of failure goes from 15% to 19%, while in the second case it goes from 39% to 42%.

D.4 Comparison to Literature

As presented in Chapter 2, most if not all solutions covering reliability aspects do not use clear metrics based on measurable parameters such as failure rates. The reliability model proposed use these metrics and combine them to internal thresholds that determine whether a task is cheap or expensive, and then whether the task should be relocated or not based on the reliability level calculated on top of the defined thresholds. The thresholds can be fixed to certain percentages or they can be set to an initial value that is dynamically modified based on the decisions taken and the outcome.

APPENDIX E – SCHEDULING FRAMEWORK

The scheduling framework is based on three main components: phases, triggers, and thresholds. Phases define the resource management steps wherein scheduling and load distribution must take place. Triggers define when scheduling decisions must be initiated. Finally, thresholds define when a scheduling decision must be actually executed. These and other auxiliary components are presented in this section.

E.1 Scheduling Phases

A typical sequence of events for resource management and workflow execution is depicted in Figure 103.



Figure 103 – Typical event order for resource management and task deployment

The process starts with the launch of a workflow, which can be represented by the submission of the workflow characterization via some management interface. The workflow

characterization is received by MPSF which immediately retrieves the list of resources and the list of current execution spaces. The execution spaces define sets of resources that already are working as a unit to execute tasks. Based on the available resources, the already existing execution spaces, and the requirements of the workflow to be executed, the framework determines if it is possible to assign an execution space to the workflow or if it is necessary to create a new space, either by adding more nodes to an existing space or simply creating a new space from a set of idle resources. Finally, the tasks are deployed onto the execution space and several metrics are calculated, such as expected makespan and the accuracy of this metric.

The main issue of having only one initial scheduling phase in this process is related to handling performance fluctuations and unexpected events, such as failures. If the resource allocation is fixed to an initial setup and a failure occurs, having less nodes to complete a task can drastically affect the makespan. Moreover, without any dynamic resource redistribution method and implementing only a static redistribution when all other nodes have completed their jobs, the makespan significantly increases.

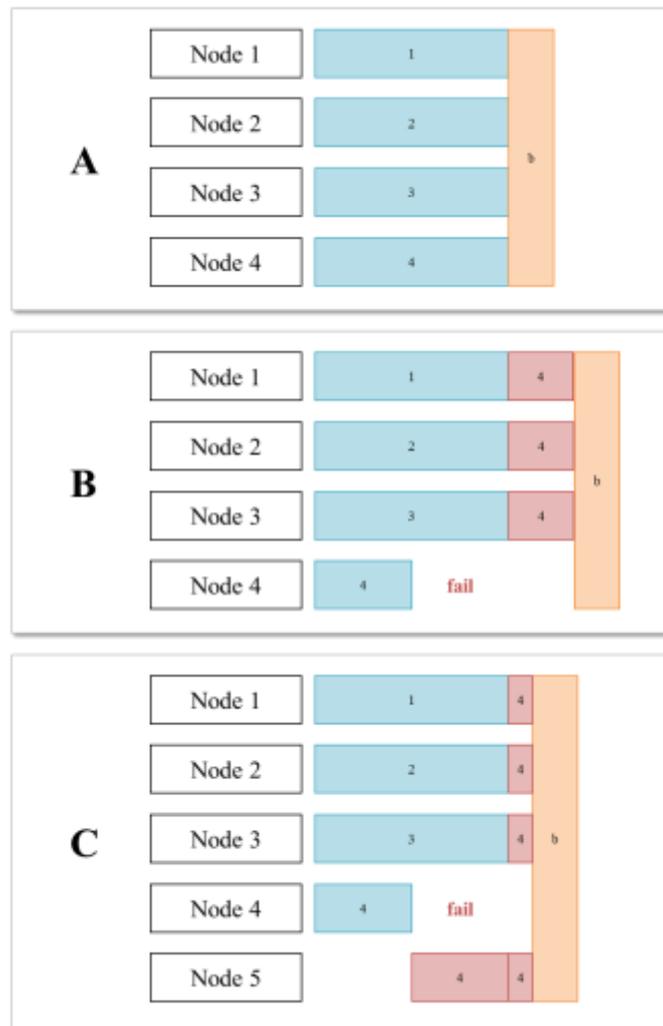


Figure 104 – Load redistribution in case of failure

Figure 104 presents an example in which four nodes execute a similar task. Scenario A shows normal execution, each node executing its own task and a barrier b at the end, which could be representing a final merge or the results. In scenario B node 4 fails around the middle of execution. The task from node 4 is redistributed at the end of the execution of the other nodes, leading to a delay of around 33% compared to the original execution of tasks, as the load from node 4 is distributed among nodes 1, 2, and 3. Scenario C shows the same situation of scenario B in which node 4 fails, but dynamic redistribution mechanisms are able to intervene and redistribute the load from node 4 first to an auxiliary node 5, and then to nodes 1, 2, and 3 when

they have finished their computation. In this sense, node 5 performs half of the computation while the other nodes are active, then the other half of the load is distributed among nodes 1, 2, 3, and 5, leading to a delay of only 12.5% (0.25 of extra load compared to the original 2.0 load per node).

In terms of performance fluctuations, the scenarios presented in Figure 105 depict situations wherein load redistribution can reduce makespan.

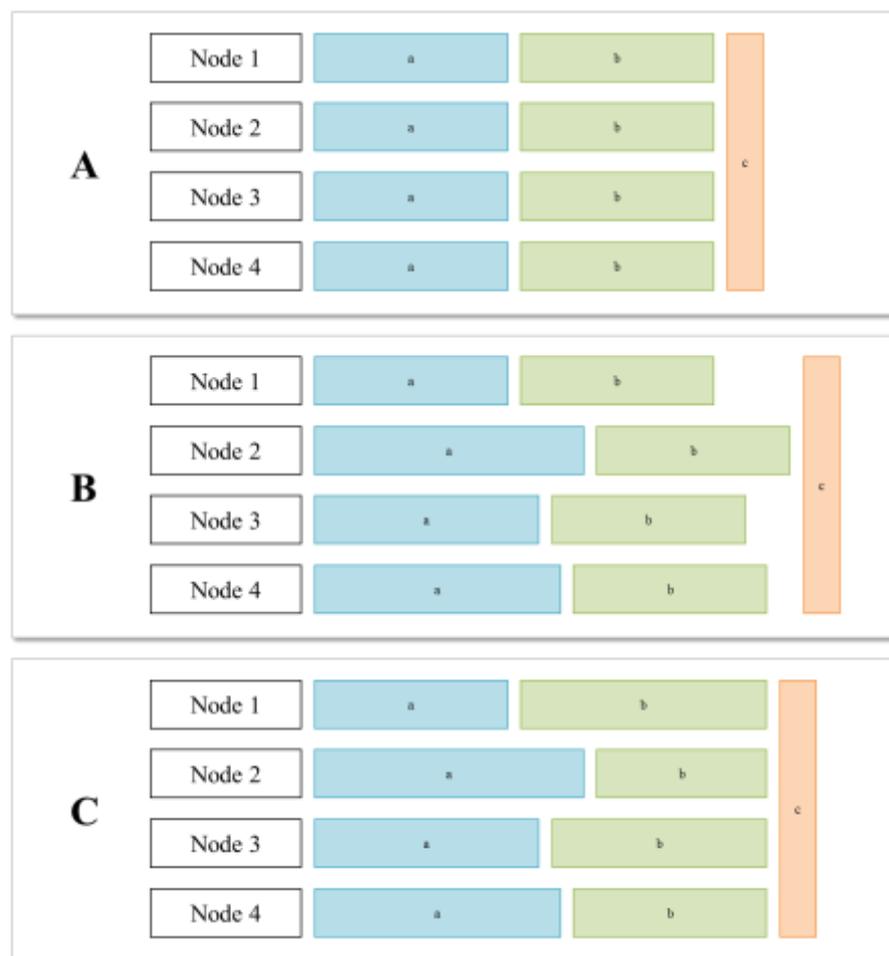


Figure 105 – Handling performance fluctuations

Scenario A represents normal execution based on a load distribution that considers the capabilities of each node and container. However, while the resources are used the observed performance differs from the expected. This is depicted in Scenario B, which shows nodes 2,

3, and 4 with discrepancies between the expected behavior and the observed performance. This delays the final result aggregation, thus increasing total makespan. A redistribution function implemented after finishing the first task is able to relocate the data in order to send more input to the nodes with more stable performance. Moreover, the probabilistic performance model can be used to determine factors to apply to each node, effectively redistributing data.

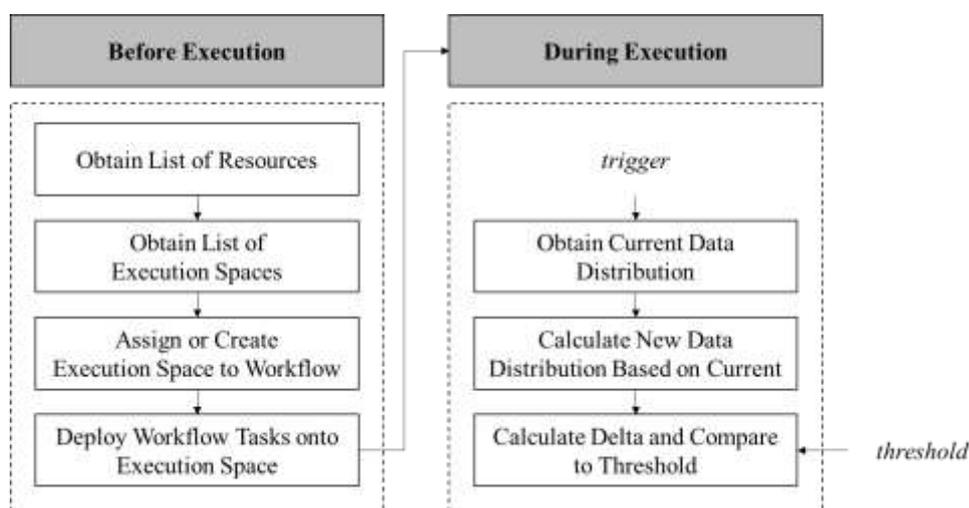


Figure 106 – Phases before and during execution

Figure 106 shows an enhanced version of the phases presented in Figure 103 including phases during the execution of the workflow. Based on specific triggers the framework obtains the current data distribution and calculates a new distribution based on the current one, to avoid a large delta between distributions. This delta is computed, which is essentially the amount of data that has to be relocated in order to deploy the new distribution. Then, this delta is compared to a threshold that determines if deploying the new configuration makes sense from a cost tradeoff perspective. If it does, the new configuration is deployed.

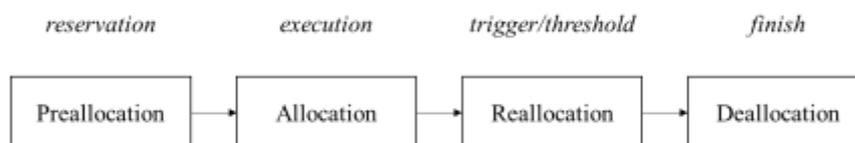


Figure 107 – Complete resource allocation life cycle

Figure 107 presents the complete resource allocation life cycle with two extra phases: preallocation and deallocation. Preallocation is related to the reservation of resources for future execution of a workflow. This is accomplished by creating a reserved execution space for a particular workflow based on performance and other operational requirements, including ones related to security. The allocation and reallocation phases, as discussed before, are related to the execution of the workflow and then to the triggers and thresholds configured for controlling the reallocation tasks. Finally, the deallocation refers to freeing resources to be used by other workflows. Note that both allocation and reallocation involve two steps: calculation and deployment. The calculation, which is based on the performance and reliability models to determine data distribution, usually is referred as provisioning in the literature, while the deployment is referred as scheduling. Moreover, for the reallocation phase, the calculation is controlled by the trigger, while the deployment is controlled by the threshold. Note also that the reallocation phase might occur several times during the execution of a workflow. The number of times is controlled by the triggers configured. The definition of multiple phases wherein resource allocation can be defined and redefined justifies the definition of MPSF as a **multiphase** scheduling framework.

E.2 Scheduling Triggers and Thresholds

As discussed in Section E.1, the triggers and thresholds are used to control the reallocation activities defined by MPSF. Figure 108 shows the resource allocation life cycle and the details of the reallocation phase with the inner steps.

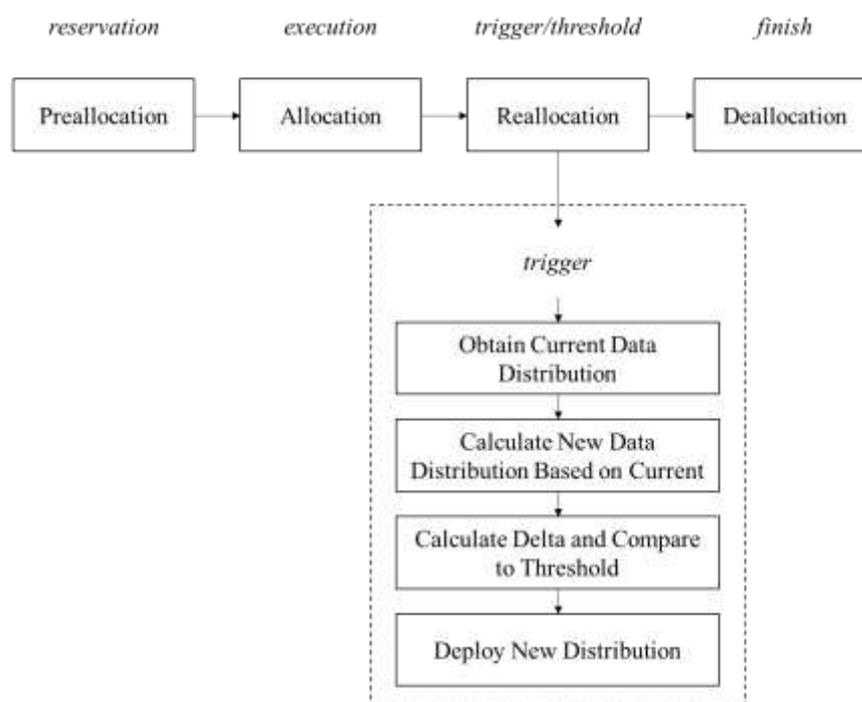


Figure 108 – Detail of reallocation showing inner steps

Three main types of triggers are defined in MPSF

- **Time-based triggers:** Triggers that are activated after certain intervals based on absolute counters and relative counters related to workflows and workflow phases. For instance, every three hours all tasks in execution must be recalculated (not necessarily redeployed). Further conditions can be established to avoid certain tasks that just started. For example, this rule could be extended to every three hours recalculate tasks that are older than one hour. Time-based triggers can also be relative to the execution of

workflow and workflow phases, as depicted in Figure 109. The first trigger is the absolute one described before. The next trigger is workflow-related, so that every 2.5 hours the tasks under execution relative to this workflow should be recalculated. Finally, the last trigger is phase-related, meaning that every 2.0 hours the tasks for a specific phase should be recalculated (tasks A and B). Note that short tasks would not have any recalculation process. The recalculations are symbolized by the lightning bolts.

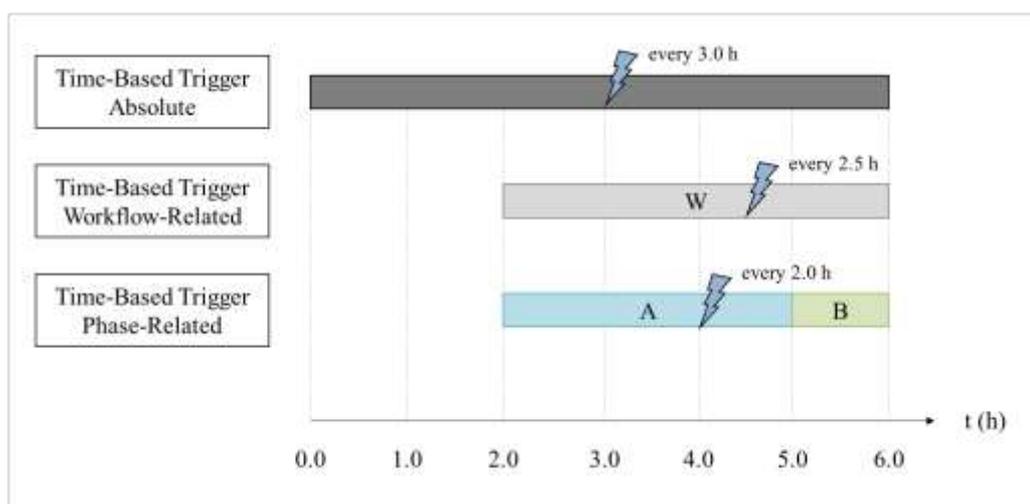


Figure 109 – Examples of time-based triggers

- **Resource-based triggers:** Triggers that are based on the resource utilization of the system. One metric used in this sense is the amount of idle resources. If the percentage of idle nodes and containers is superior to a certain percentage and there is another certain percentage of nodes being used with active tasks, the distributions are recalculated and possibly the execution spaces are redefined. Another metric is the total utilization of resources – if it is beyond a certain percentage, the distributions are recalculated in order to consolidate certain tasks in smaller execution spaces without affecting performance. This operation is similar to the defragmentation of a hard disk.

- **Workflow-based triggers:** The workflow-based triggers are started during specific workflow phases, which are controlled by specific primitives. Two main primitives may trigger recalculations: Input primitives and Split primitives. They represent the beginning of a series of tasks and the distribution of data to accomplish those tasks, respectively. Therefore, applying a recalculation when one of these primitives is encountered during workflow execution might lead to resource savings and performance optimization, especially due to performance fluctuations observed from when the workflow started to its current time.

Triggers are used by MPSF to determine when to recalculate the load distribution. Thresholds are used to determine when to deploy the new load distribution. MPSF considers two main types of thresholds: absolute and relative.

- **Absolute threshold:** A fixed percentage that controls whether a new load distribution should be deployed or not. For instance, if MPSF is working with a static threshold of 80%, this means that if the new data distribution leads to a performance improvement of 80% or more, the new distribution is deployed. Additionally, if MPSF is working with a static performance threshold of 80% and a cost threshold of 15%, this means that if a new distribution leads to a performance increase of 80% or more **OR** if the cost of deploying the new distribution is less than 15%, this distribution is deployed.
- **Relative threshold:** A variable percentage depending on the tradeoff between performance gain and cost of deploying the new configuration. This relation is expressed either in terms of a subtraction or a division. For instance, if the relative threshold is calculated via subtraction and is equal to 70%, this means that if the performance gain of a new distribution is 80% and the cost is 5%, the final value is 75%,

which means that the new configuration is deployed. If the cost was 15%, however, the final percentage would be 65%, thus the configuration would not be deployed.

All trigger and threshold-related percentages are configurations made to MPSF. These configurations can be preset to specific values or they can be left in their default values and then dynamically adjusted by MPSF via a simple positive-feedback mechanism based on counters and correct assumptions. For instance, if the current absolute performance threshold is 40%, a recalculation decision is accepted with a predicted improvement of 45%, and the final observed performance is indeed superior than 40%, the threshold is either kept or slightly decreased to embrace suggestions that predict lower gains. On the other hand, if the same scenario is observed but the final performance is inferior than the expected, or maybe even worse than the original (without redeployment), the threshold is automatically increased by MPSF (typically between 1% to 5% steps, although the increases are also configurable). All percentages are implemented using this positive-feedback system, although they can all be set to a fixed value as well.

E.3 Operational Aspects

In terms of operational aspects, the scheduling phases control the main flow of the scheduling operations. As discussed before, the scheduling activities are organized in two main steps: calculation and deployment. Calculation refers to the utilization of performance and reliability models to determine the optimal distribution of load and data to the nodes of an execution space, as well as determining the creation or modification of current execution spaces. Then, the deployment refers to the actual distribution of tasks to elements of the execution space.

The utilization of the performance model is organized in two steps. First step comprises the utilization of the final model to determine the distribution of shared and chunk data. The second step comprises the utilization of the probabilistic model to determine correction factors to be applied to this distribution. During the first allocation activities performed by MPSF the factors should not affect the distribution. After a few successful allocations, and after acquiring enough knowledge in the form of markers and the Bayesian system, the factors can be manually or automatically increased by MPSF.

The reliability model is used as a complement to the performance model first by also providing factors to redistribute data. In this sense, nodes with a higher chance of failure are assigned low factors, reducing the amount of data that they receive. Moreover, the classification in five levels is used to determine if a specific node should be completely avoided in terms of assigning unique tasks. These nodes are still used for reliability purposes, such as replaying tasks or redundancy.

APPENDIX F – ARCHITECTURE

Based on the characterization of workflows, resources, based on the performance and reliability models, and based on the scheduling primitives defined for MPSF, an architecture is defined to encompass all these elements while providing infrastructural functionalities to implement all these mechanisms. This section presents the requirements for this architecture and the components that compose it.

F.1 Architectural Requirements

The Architectural Requirements (AR) are directly related to the Solution Requirements (SR), in particular the ones referring to SR6 in Table 9.

- P1-SR6: The architecture must define components that implement the controls based on data movement to address the requirements of data-intensive workflows.
- P2-SR6: The architecture must define components to address implementation-related issues comprising heterogeneous resources and large-scale management.
- P3-SR6: The architecture must provide components associated to the management framework to implement the controls for hybrid/multicloud scenarios.

An additional requirement is defined to cover the functionalities related to the scheduling framework:

- P4-SR6: The architecture must define components to implement the phases, triggers, and thresholds defined by the scheduling framework.

Finally, the architecture must provide additional functionalities to support the operations of the framework.

- AUX1: The architecture must support mechanisms for resource discovery, enabling automated detection of resources added to the cloud environment being managed by MPSF.
- AUX2: The architecture must support automation of resource characterization.
- AUX3: The architecture must support automation of workflow characterization, providing mechanisms that automate the process of definition of costs and other properties to workflow phases.

F.2 Architectural Components

A summary of the proposed architecture for MPSF is presented in Figure 110.

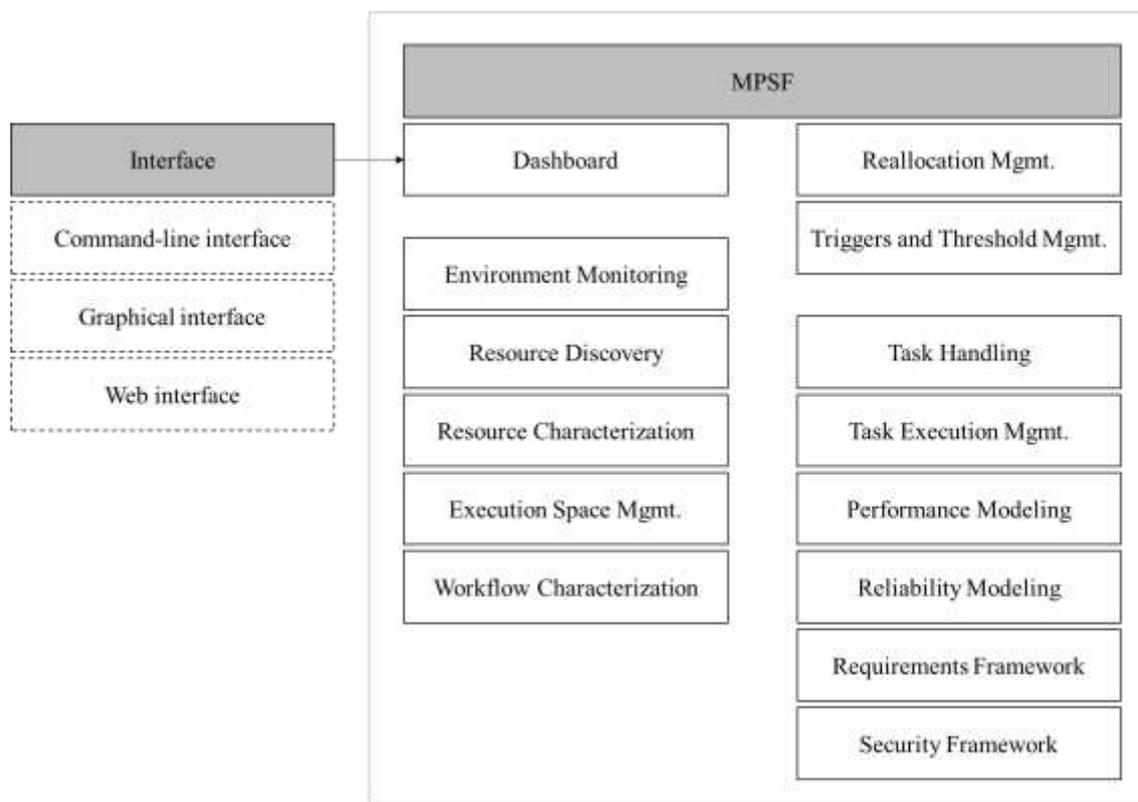


Figure 110 – MPSF Architecture

An interface is used to communicate with MPSF to submit new workflows and configure the framework. The interface from MPSF-side is provided by the Dashboard component. Then three groups of components are defined:

- **Resource and Workflow Management Group:** This group comprises components to monitor the environment and collect metrics related to resource utilization, components to provide resource discovery and resource characterization capabilities, as well as components for management of execution spaces and for characterizing workflows.
- **Workflow Allocation Group:** This group comprises components used to allocate workflow tasks to nodes and containers for execution. This includes components to handle new tasks to be executed, to manage the tasks being executed, components that

provide an interface to the performance and reliability models, as well as components that interpret the requirements defined by the user for executing a workflow, with particular controls for security requirements.

- **Reallocation Group:** This group comprises specialized functionalities to handle reallocation of tasks and load. This includes the management of reallocation operations and the management of triggers and thresholds used to control these operations.

F.2.1 Dashboard

The dashboard from an architectural point of view is a component that defines a set of interfaces to allow users and administrators to communicate with the framework and issue commands. These commands are organized in two groups: commands for task **execution** and for framework **configuration**.

- **Execution:** Commands related to the execution of tasks and workflows. For instance, the user specifies the workflow to be executed and the requirements related to performance (e.g., desired makespan) and other operational aspects, such as security (e.g., use a specific set of nodes that provide a certain security feature).
- **Configuration:** Commands used to define certain operational parameters of the framework, such as the values of triggers and thresholds, values for percentages in terms of node utilization and load distribution, etc.

F.2.2 Environment Monitoring

Part of the Resource and Workflow Management Group, the Environment Monitoring component provides all functionalities related to obtaining information about the cloud environment, including hardware and software (e.g., tasks in execution and underlying software). This monitoring is used to control certain triggers, for instance, which are based on the resource levels of nodes. Moreover, this monitoring is used in the allocation and reallocation calculations, as well as in the probabilistic performance model which relies on the results of the execution of tests that assess the conditions of the system.

F.2.3 Resource Discovery

Also part of the Resource and Workflow Management Group, the Resource discovery component provides functionalities to automate the process of detecting new resources added to the environment. This functionality is particularly important for large environments with components being constantly added or removed due to failures. Moreover, this also facilitates the modification of components, such as installing new versions of software.

The implementation of the resource discovery mechanism relies on the definition of a protocol that establishes how nodes are discovered. MPSF defines two types of nodes in this sense: Information Servers and Information Clients. Information Servers contain the information about the nodes that compose an environment and constantly listen to the network looking for new nodes. When a new node is added to the cloud and the MPSF module is added to this node, a broadcast message is sent to the network to indicate that a new node is available. If the Information Servers are known, however, then this message can be unicasted to a particular node.

F.2.4 Resource Characterization

Also part of the Resource and Workflow Management Group, the Resource Characterization component provides automated mechanisms to define the performance and reliability properties of a node or a container. Two sets of information are determined regarding the performance characterization: a **general** characterization and a **specific** characterization.

- **General characterization:** Calculated using a set of predefined benchmarks to stress specific aspects of the node, such as processing, memory bandwidth, storage bandwidth, and networking. For each aspect a specific benchmark is defined, as well as a set of inputs to be used to create a performance map for each case. For instance, LINPACK¹¹ is typically used to measure computational capacity, while STREAM¹² is used to measure memory bandwidth. I/O capacity is measured via intensive reading and writing programs, and networking is measured between nodes of the same network and among different networks.
- **Specific characterization:** The specific characterization is defined for a specific workflow. In this case a workflow specification is provided to MPSF which executes workflow phases on different nodes. For each node the performance metrics such as execution time, processing rate, I/O rates, as well as memory and network utilization are computed and stored in a common database. This database is then used to improve performance prediction and to calculate the data distribution for workflow execution.

¹¹ <https://www.top500.org/project/linpack/>

¹² <https://www.cs.virginia.edu/stream/>

The method used in both cases is based on the repetition of several experiments using the benchmarks and workflow phases, varying input parameters such as load and operational settings.

F.2.5 Execution Space Management

This component is responsible for handling all operations related to execution spaces, including their creation, scheduling, modification, and termination. Execution spaces, as discussed before, are used to organize the execution of workflows and tasks under the same namespace. The execution spaces are also used to provide a common infrastructure for the execution tasks, allowing for example the definition of variables and parameters that can be read throughout all nodes and containers that are part of the same execution space.

The operations supported for execution spaces are:

- **Creation:** Definition of a new execution space comprising a set of nodes and containers.
- **Scheduling:** Assigning a set of tasks and/or workflows to execute in a specific execution space.
- **Modification:** Change the properties and the components of an execution space.
- **Termination:** Disassociate the execution space from its components, allowing these components to be freely allocated to other execution spaces. Note that components might be part of more than one execution space without necessarily having to terminate one of the execution spaces the component is part of.

F.2.6 Workflow Characterization

This component is responsible for automating the process of characterizing a workflow. As discussed before, the workflow characterization can be defined in terms of fixed values or variables that depend on the complexity of the operations (i.e., to transfer and process data) or based on the data structures. The representation in terms of complexity and data structures allow a more precise calculation of the time required to transfer data and to process it during the execution of a workflow. In addition, the automated characterization capabilities provide a method to validate these numbers by executing a series of tests varying the inputs to assess the sensitivity in terms of performance based on the input.

The process of characterizing the workflow is similar to the automated characterization of resources. In this case, each phase and transition is tested with different inputs and other settings (which have to be specified in the workflow description). The variation of the inputs is measured against the variation of performance, represented in terms of execution time, memory footprint, network utilization, etc. All these parameters are then recorded in a database that contain the performance variation according to the inputs provided to the workflows. Finally, based on these numbers, the variation is adjusted to a curve that represents the general behavior of the task. The form of the curve is selected among a set of generic curves, such as $f(x) = ax + b$, $f(x) = a/x$, $f(x) = ae^{bx}$, etc. Several curves are predefined in the framework and then adjusted using non-linear regression methods such as the Gauss-Newton algorithm:

Equation 49 – Gauss-Newton algorithm for non-linear regression

$$\beta^{(s+1)} = \beta^{(s)} + (J_f^T \cdot J_f)^{-1} \cdot J_f^T \cdot r(\beta^{(s)})$$

Where $\beta^{(s)}$ is a vector representing the coefficients $C = \{a, b, \dots\}$ that compose the regression function f , J_f is the Jacobian matrix of the function f , and r is the vector of residuals calculated as:

Equation 50 – Residuals

$$r_i(\beta) = y_i - f(x_i, \beta)$$

In essence, the residuals represent the difference between the actual value y_i and the value calculated using the regression function f , which is obtained by using the value x_i and the coefficients represented by β . The Jacobian matrix is defined by the partial derivatives of function f as follows:

Equation 51 – Jacobian matrix elements

$$(J_f)_{ij} = \frac{\partial f_i(\beta^{(s)})}{\partial \beta_j}$$

With these expressions it is possible to implement an iterative method to adjust a set of points (x_i, y_i) , which are the inputs and respective performance results obtained by the automated execution of the workflow phases, to the regression functions f . Then, for each function and set of coefficients the residuals are calculated and the best fit is obtained by verifying which function leads to the smallest residuals.

F.2.7 Task Handling

The first component of the Workflow Allocation Group, the Task Handling component is responsible for receiving new tasks and initiating the allocation process. In terms of implementation, the task handling is provided by several listeners that constantly receive requests and translate those requests into a set of parameters that are sent to the Task Execution Management component.

F.2.8 Task Execution Management

This is the main component for Workflow Allocation, responsible for managing the process of using the performance and reliability models in order to determine the load distribution based on the workflow properties, available resources, and current status of execution spaces.

F.2.9 Performance Modeling and Reliability Modeling

Two auxiliary components that provide an interface with the performance and reliability models. These models and their sub-mechanisms define sets of parameters that orchestrate the operation of the models, such as how to combine phases to implement pipelining, how to update factors for the Bayesian system, etc. These architectural components provide the programming mechanisms to control these parameters, as well as to define basic configurations of the models to be used to calculate the data distributions.

F.2.10 Requirements Framework

This architectural component defines mechanisms to interpret the requirement properties defined for a workflow execution request and for the resources description provided to the framework. As discussed in Table 36, requirement properties are related to constraints such as the necessity of having a particular resource (e.g., GPU) in place, or support to certain functions inherent to the network (e.g., SDN and NFV). Each requirement is related to a certain mechanism to verify the resource. For instance, if the workflow requires GPUs for a particular phase, the execution space dedicated to this workflow must include at least a few nodes with GPUs. The verification test that detects the presence of GPUs and also specific characteristics (e.g., driver version, architecture, etc.) is controlled by this component of the architecture.

F.2.11 Security Framework

The last architectural component for Workflow Allocation, the Security Framework comprises a series of mechanisms to address security requirements related to specific nodes to be used for a workflow phase, or general security requirements that must be addressed during the execution of the workflow. MPSF implements two sets of security policies: one based on nodes and another based on capabilities.

The node-specific policies represent groups of nodes (usually within certain networks) that might execute a specific code. For instance, only nodes from a specific sub-network of the cloud can receive and run a proprietary code that must not be collocated with other workflows.

The capabilities-based policies represents requirements related to the availability of certain mechanisms, such as Trusted Platform Modules (TPMs) and specific software.

F.2.12 Reallocation management

Architectural component that provides the mechanisms to reallocate tasks based on triggers and thresholds to control when recalculations must be executed and when the new distribution must be deployed, respectively.

F.2.13 Trigger and Threshold Management

A specialized architectural component to manage the triggers and thresholds used in the system. All configurations related to these mechanisms are controlled by this component.

F.3 Integration to Computing Frameworks

MPSF can be transparently integrated to other distributed computing frameworks. Three frameworks were selected due to their prominence in modern workflows and systems such as CORAL¹³ and APEX¹⁴: MPI, OpenMP, and CUDA. This section presents the integration points provided by MPSF and how these points are used to integrate these frameworks into MPSF.

F.3.1 MPI

MPI is a standardized message-passing system to implement parallel computing architectures, and it is implemented by efforts such as OpenMPI¹⁵ and MPICH¹⁶.

¹³ <https://asc.llnl.gov/coral-info>

¹⁴ <http://www.lanl.gov/projects/apex/>

¹⁵ <https://www.open-mpi.org/>

¹⁶ <https://www.mpich.org/>

MPI processes or ranks are created to distribute load and tasks among nodes, cores, and threads. Moreover, the implementations provide support to the definition of affinity configurations for the processes, establishing for instance that within a node there must be one process per core or per thread. These options facilitate the integration with MPSF in the sense that MPSF is able to control all aspects of the resource hierarchy. For example, to execute an MPI program using several nodes of a cloud, each instance of the program can be configured to run one MPI process per core while MPSF is configured to run one MPI program instance per node. After all instances are done a final aggregation phase is used to consolidate the results. In this sense, the integration between MPSF and MPI depends on programming the MPI side to be restricted to one node or a controlled small set of nodes and the definition of an additional aggregation phase to occur after the MPI instances are finished. Because MPSF is able to dynamically distribute load, including while process are in execution, the load for each MPI instance can be dynamically balanced while it is being distributed to the processes. This implements an additional load balancing layer on top of MPI while remaining completely transparent to the MPI instance. Affinity is defined using *rankfiles*, which designate how MPI processes (ranks) should be distributed among nodes, cores, and threads. In this case, each MPI instance would define a rankfile to distribute ranks among cores or threads of the same node.

F.3.2 OpenMP

OpenMP¹⁷ is an API that supports multi-platform parallel programming in C/C++ and Fortran. The utilization of OpenMP from MPSF point of view is completely transparent, meaning that an OpenMP program can be executed on the infrastructure managed by MPSF as

¹⁷ <http://openmp.org/wp/>

a simple program or workflow phase. In this sense, OpenMP provides the intra-node-level parallelism (threading) while MPSF provides inter-node-level parallelism (among nodes from one or more clouds).

F.3.3 CUDA

CUDA¹⁸ is a parallel computing platform defined by NVIDIA to harness the compute power of their GPUs. Similar to OpenMP, the utilization of CUDA from MPSF point of view is completely transparent. CUDA programs can be executed in nodes managed by MPSF and the GPU requirement is addressed by the Requirements Framework architectural component, as well as by the resource characterization provided to MPSF.

¹⁸ http://www.nvidia.com/object/cuda_home_new.html