Um processo de desenvolvimento de software
focado em sistemas distribuídos autônômicos

Pedro Felipe do Prado

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Pedro Felipe do Prado

Um processo de desenvolvimento de software focado em sistemas distribuídos autônômicos

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional. *EXEMPLAR DE DEFESA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Marcos José Santana

USP – São Carlos
Abril de 2017

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

d896p do Prado, Pedro Felipe
Um processo de desenvolvimento de software
focado em sistemas distribuídos autônomicos / Pedro
Felipe do Prado; orientador Marcos José Santana. --
São Carlos, 2017.
138 p.

Tese (Doutorado - Programa de Pós-Graduação em
Ciências de Computação e Matemática Computacional) --
Instituto de Ciências Matemáticas e de Computação,
Universidade de São Paulo, 2017.

1. Processo de desenvolvimento de software. 2.
Sistemas distribuídos. 3. Computação autônoma. 4.
Pesquisa operacional. 5. Avaliação de desempenho. I.
Santana, Marcos José, orient. II. Título.

Pedro Felipe do Prado

**A software development process focused on autonomic
distributed systems**

Doctoral dissertation submitted to the Instituto de
Ciências Matemáticas e de Computação – ICMC-
USP, in partial fulfillment of the requirements for the
degree of the Doctorate Program in Computer Science
and Computational Mathematics. *EXAMINATION
BOARD PRESENTATION COPY*

Concentration Area: Computer Science and
Computational Mathematics

Advisor: Prof. Dr. Marcos José Santana

USP – São Carlos
April 2017

*Dedico este trabalho ao meu irmão,
Rodrigo do Prado,
que sempre dizia que meu doutorado não acabaria nunca.*

AGRADECIMENTOS

Agradeço primeiramente à minha família, por todo suporte emocional e também financeiro, no decorrer desses doze longos anos na Universidade de São Paulo. Sem eles, não teria conseguido chegar até aqui. É realmente gratificante ainda ouvir, com certa estranheza, como as pessoas admiram e reconhecem o valor que a USP possui em todas as áreas de conhecimento.

Gostaria de agradecer ao meu orientador, Professor Doutor Marcos José Santana, que tem me acompanhado desde quando entrei no mestrado, em março de 2010. Foram muitos ensinamentos passados, durante esses sete anos sob a sua orientação. As disciplinas que cursei na pós-graduação foram muito interessantes e proveitosas, sem falar nos seminários do grupo, que permitiram que eu desenvolvesse minhas capacidades de falar em público e responder a questionamentos sobre meu projeto de pesquisa.

Também agradeço a todos os colegas do Laboratório de Sistemas Distribuídos e Programação Concorrente (LaSDPC), aonde realizei meu mestrado e doutorado. Em especial, ao amigo Luis Hideo Vasconcelos Nakamura, que me auxiliou dando conselhos tanto no mestrado, como no doutorado.

Por fim, agradeço ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), pelas bolsas de mestrado e de doutorado. Também agradeço a Coordenadoria de Aperfeiçoamento de Pessoal de Nível Superior (Capes) pelo auxílio financeiro usado por mim em um congresso em Roma, aonde apresentei alguns resultados iniciais do meu doutorado. É realmente gratificante, poder encontrar pessoalmente com cientistas renomados, aqueles que você costuma citar diversas vezes na sua tese.

“Inteligência é a capacidade de se adaptar à mudança.”
(Stephen Hawking)

RESUMO

PRADO, P. F. **Um processo de desenvolvimento de software focado em sistemas distribuídos autônômicos**. 2017. 138 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

Os Sistemas Distribuídos (SDs) tem apresentado uma crescente complexidade no seu gerenciamento, além de possuir a necessidade de garantir Qualidade de Serviço (QoS) aos seus usuários. A Computação Autônômica (CA) surge como uma forma de transformar os SDs em Sistemas Distribuídos Autônômicos (SDAs), com capacidade de auto-gerenciamento. Entretanto, não foi encontrado um processo de desenvolvimento de *software*, focado na criação de SDAs. Na grande maioria dos trabalhos relacionados, simplesmente é apresentado um SD, juntamente com qual aspecto da CA deseja-se implementar, a técnica usada e os resultados obtidos. Isso é apenas uma parte do desenvolvimento de um SDA, não abordando desde a definição dos requisitos até a manutenção do *software*. Mais importante, não mostra como tais requisitos podem ser formalizados e posteriormente solucionados por meio do auto-gerenciamento fornecido pela CA. Esta tese foca na proposta de um processo de desenvolvimento de *software* voltado para SDAs. Com esse objetivo, foram integradas diferentes áreas de conhecimento, compreendendo: Processo Unificado de Desenvolvimento de *Software* (PU), SDs, CA, Pesquisa Operacional (PO) e Avaliação de Desempenho de Sistemas Computacionais (ADSC). A prova de conceito foi feita por meio de três estudos de caso, todos focando-se em problemas NP-Difícil, são eles: (i) otimização *off-line* (problema da mochila com múltiplas escolhas), (ii) otimização *online* (problema da mochila com múltiplas escolhas) e (iii) criação do módulo planejador de um gerenciador autônômico, visando realizar o escalonamento de requisições (problema de atribuição generalizado). Os resultados do primeiro estudo de caso, mostram que é possível usar PO e ADSC para definir uma arquitetura de base para o SDA em questão, bem como reduzir o tamanho do espaço de busca quando o SDA estiver em execução. O segundo, prova que é possível garantir a QoS do SDA durante sua execução, usando a formalização fornecida pela PO e sua respectiva solução. O terceiro, prova que é possível usar a PO para formalizar o problema de auto-gerenciamento, bem como a ADSC para avaliar diferentes algoritmos ou modelos de arquitetura para o SDA.

Palavras-chave: Processo de Desenvolvimento de Software, Computação Autônômica, Avaliação de Desempenho, Sistemas Distribuídos, Pesquisa Operacional.

ABSTRACT

PRADO, P. F. **A software development process focused on autonomic distributed systems.** 2017. 138 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

Distributed Systems (DSs) have an increasing complexity and do not have their management, besides having a quality of service (QoS) to its users. Autonomic Computing (AC) emerges as a way of transforming the SDs into Autonomous Distributed Systems (ADSs), with a capacity for self-management. However, your software development process is focused on creating SDAs. In the vast majority of related works, simply an SD model, along with what aspect of the AC implement, a technique used and the results obtained. This is only a part of the development of an ADS, not approaching from an definition of requirements for a maintenance of software. More importantly, it does not show how such requirements can be formalized and subsequently solved through the self-management provided by AC. This proposal aims at a software development process for the DASs. To this end, different areas of knowledge were integrated, including: Unified Software Development Process (PU), SDs, CA, Operations Research (OR) and Computer Systems Performance Evaluation (CSPE). The proof of concept was made through three case studies, all focusing on NP-Hard problems, namely: (i) off-line optimization (problem of the backpack with multiple choices), (ii) (Problem of the backpack with multiple choices) and (iii) creation of the scheduling module of an autonomic manager, aiming to carry out the scheduling of requests (problem of generalized assignment). The results of the first case study show that it is possible to use OR and CSPE to define a base architecture for the DAS in question, as well as reduce the size of the search space when SDA is running. The second, proves that it is possible to guarantee the QoS of the DAS during its execution, using the formalization provided by the OR and its respective solution. The third, proves that it is possible to use the PO to formalize the self-management problem, as well as the ADSC to evaluate different algorithms or architecture models for the ADS.

Keywords: Unified Process, Autonomic Computing, Performance Evaluation, Distributed Systems, Operations Research.

LISTA DE ILUSTRAÇÕES

Figura 1 – Visão Geral do PU	32
Figura 2 – Fluxos de trabalho e fases do PU	36
Figura 3 – Diferentes modelos de Computação Distribuída	38
Figura 4 – Modelo de arquitetura clássica de Web services	41
Figura 5 – Aspectos da Composição de Web services baseada em QoS	43
Figura 6 – Laço de Controle MAPE-C	46
Figura 7 – Modelo de arquitetura centralizado	47
Figura 8 – Modelo de arquitetura distribuída	47
Figura 9 – Modelo de arquitetura hierárquica	48
Figura 10 – O Processo de modelagem usando Pesquisa Operacional	51
Figura 11 – Função a ser otimizada.	52
Figura 12 – Espaço de decisão do problema	53
Figura 13 – Comportamento de algoritmos em relação ao tamanho da entrada.	59
Figura 14 – Mapeamento dos Requisitos Funcionais	83
Figura 15 – Mapeamento dos Requisitos Não-Funcionais	83
Figura 16 – WSs abstratos, WSs concretos, atributos de QoS e restrição.	94
Figura 17 – Proporção entre pontos que cumprem a restrição e total de pontos, para dois WSs abstratos.	100
Figura 18 – Proporção entre pontos que cumprem a restrição e total de pontos, para três WSs abstratos.	100
Figura 19 – Proporção entre pontos que cumprem a restrição e total de pontos, para quatro WSs abstratos.	101
Figura 20 – Proporção entre pontos que cumprem a restrição e total de pontos, para cinco WSs abstratos.	101
Figura 21 – QoS obtido pela BE e HG, com 2 WSs abstratos.	102
Figura 22 – QoS obtido pela BE e HG, com 3 WSs abstratos.	102
Figura 23 – QoS obtido pela BE e HG, com 4 WSs abstratos.	103
Figura 24 – QoS obtido pela BE e HG, com 5 WSs abstratos.	103
Figura 25 – Análise da influência dos fatores.	106
Figura 26 – Arquitetura usada no Estudo de Caso 2.	108
Figura 27 – Tempo médio de execução do algoritmo BE.	110
Figura 28 – Tempo médio de execução do algoritmo BE.	111
Figura 29 – Tempo médio de execução do algoritmo BE.	111

Figura 30 – Tempo médio de execução do algoritmo BE.	112
Figura 31 – Tempo médio de execução do algoritmo HG.	112
Figura 32 – Tempo médio de execução do algoritmo HG.	113
Figura 33 – Tempo médio de execução do algoritmo HG.	113
Figura 34 – O funcionamento do Escalonador Autônomo de Requisições.	118
Figura 35 – Gráfico dos experimentos 1 ao 16.	120
Figura 36 – Gráfico dos experimentos 17 ao 32.	121
Figura 37 – Gráfico dos experimentos 33 ao 48.	122

LISTA DE TABELAS

Tabela 1 – Funções de agregação dos atributos de QoS	42
Tabela 2 – Modelos de arquitetura de um Sistema Distribuído Autônomo	48
Tabela 3 – Resumo das técnicas usadas para problemas de sistemas distribuídos	49
Tabela 4 – Metodologias para avaliação de desempenho.	57
Tabela 5 – Complexidades assintóticas e exemplos.	58
Tabela 6 – Parâmetros estáticos do Apache Httpd Server.	60
Tabela 7 – Parâmetros dinâmicos do Apache Httpd Server	60
Tabela 8 – Resumo das técnicas usadas por aspecto da computação autônoma	70
Tabela 9 – Qtd. de referências, por ano.	75
Tabela 10 – Qtd de referências, por foco.	75
Tabela 11 – Qtd. de referências, por validação.	76
Tabela 12 – Qtd de referências, por categoria.	76
Tabela 13 – Quantidade de referências, por aspecto da CA	77
Tabela 14 – Papéis e metodologias de ADSC.	85
Tabela 15 – Atividades e artefatos gerados na Fase de Iniciação.	86
Tabela 16 – Atividades e artefatos gerados na Fase de Elaboração	88
Tabela 17 – Atividades e artefatos gerados na Fase de Construção	89
Tabela 18 – Atividades e artefatos gerados na Fase de Transição	90
Tabela 19 – Requisitos Funcionais do Estudo de Caso 1.	95
Tabela 20 – Requisitos Não-Funcionais do Estudo de Caso 1.	95
Tabela 21 – Fatores e níveis escolhidos.	99
Tabela 22 – Resultados obtidos pelo BE e HG, com dois WSs abstratos no fluxo de composição.	104
Tabela 23 – Resultados obtidos pelo BE e HG, com três WSs abstratos no fluxo de composição.	105
Tabela 24 – Resultados obtidos pelo BE e HG, com quatro WSs abstratos no fluxo de composição.	105
Tabela 25 – Resultados obtidos pelo BE e HG, com cinco WSs abstratos no fluxo de composição.	105
Tabela 26 – Fatores, níveis e variável de resposta.	106
Tabela 27 – Requisitos Funcionais do Estudo de Caso 2.	107
Tabela 28 – Requisitos Não-Funcionais do Estudo de Caso 2.	107
Tabela 29 – Fatores e níveis escolhidos.	110

Tabela 30 – Requisitos Funcionais do Estudo de Caso 3.	116
Tabela 31 – Requisitos Não-Funcionais do Estudo de Caso 3.	116
Tabela 32 – Fatores e níveis escolhidos. Política escolhida: escalonamento circular . . .	120
Tabela 33 – Fatores e níveis escolhidos. Política escolhida: escalonamento limite . . .	120
Tabela 34 – Fatores e níveis escolhidos. Política escolhida: escalonamento circular . . .	121
Tabela 35 – Fatores e níveis escolhidos. Política escolhida: escalonamento limite . . .	121
Tabela 36 – Fatores e níveis escolhidos. Política escolhida: escalonamento circular . . .	122
Tabela 37 – Fatores e níveis escolhidos. Política escolhida: escalonamento limite . . .	122

LISTA DE ABREVIATURAS E SIGLAS

AM	Aprendizado de Máquina
ATM	<i>Asynchronous Transfer Mode</i>
BE	Busca Exaustiva
BL	Busca Local
CA	Computação Autônômica
CAD	Computação de Alto Desempenho
CD	Computação Distribuída
CG	Computação em Grade
CGC	Computação em Grupo de Computadores
CN	Computação em Nuvem
CWSbQ	Composição de Web Services baseada em atributos de QoS
EAR	Escalonador Autônomo de Requisições
EAs	Elementos Autônomos
FO	Função-Objetivo
FSC	Federações de Sistemas Computacionais
GA	Gerenciador Autônomo
HG	Heurística Gulosa
HTTP	<i>HyperText Transfer Protocol</i>
IaaS	<i>Infrastructure as a Service</i>
IdC	Internet das Coisas
IP	<i>Internet Protocol</i>
IUR	Identificador Uniforme de Recursos
JVM	<i>Java Virtual Machine</i>
LANs	<i>Local-area networks</i>
MMMO	Mochila Multi-dimensional com Múltiplos Objetivos
MVs	Máquinas Virtuais
OC	Otimização Combinatória
OV	Organização Virtual
PaaS	<i>Platform as a Service</i>
PDS2DA	Processo de Desenvolvimento de <i>Software</i> focado em Sistemas Distribuídos Autônomos

PEs	Políticas Estáticas
PMB	Problema da Mochila Binário
PO	Pesquisa Operacional
QoS	<i>Quality of Service</i>
RBC	Raciocínio Baseado em Casos
RF	Requisitos Funcionais
RG	Recurso Gerenciável
RNF	Requisitos Não-Funcionais
SaaS	<i>Software as a Service</i>
SAs	Sistemas Autônômicos
SDAs	Sistemas Distribuídos Autônômicos
SDs	Sistemas Distribuídos
SGBD	Sistema Gerenciador de Banco de Dados
SOAP	<i>Simple Object Access Protocol</i>
SONET	<i>Synchronous Optical Networking</i>
SOs	Sistemas Operacionais
TME	Tempo Médio de Execução
UDDI	<i>Universal Description, Discovery, and Integration</i>
UML	<i>Unified Modeling Language</i>
VD	Variáveis de Decisão
W3C	<i>World Wide Web Consortium</i>
Web	<i>World Wide Web</i>
WSDL	<i>Web Service Definition Language</i>
WSs	Web services
XML	<i>eXtensible Markup Language</i>

SUMÁRIO

1	INTRODUÇÃO	25
1.1	Contextualização	26
1.2	Motivação e objetivos	26
1.3	Estrutura	28
2	REVISÃO DA LITERATURA	31
2.1	Processo Unificado	31
2.1.1	<i>O Processo Unificado é conduzido por casos de uso</i>	32
2.1.2	<i>O Processo Unificado é centrado na arquitetura</i>	32
2.1.3	<i>O Processo Unificado é iterativo e incremental</i>	33
2.1.4	<i>A vida do Processo Unificado</i>	34
2.2	Sistemas Distribuídos	36
2.2.1	<i>Desafios relacionados aos Sistemas Distribuídos</i>	37
2.2.2	<i>Modelos de Computação Distribuída</i>	38
2.2.3	<i>Web Services</i>	40
2.2.4	<i>Qualidade de Serviço</i>	40
2.2.5	<i>Composição de Web Services com Qualidade de Serviço</i>	42
2.3	Computação Autônoma	43
2.3.1	<i>Aspectos Fundamentais da Computação Autônoma</i>	44
2.3.2	<i>Arquitetura de um Sistema Autônomo</i>	45
2.3.3	<i>Modelos de Arquiteturas de Sistemas Autônomos</i>	46
2.3.4	<i>Computação Autônoma e Sistemas Distribuídos</i>	47
2.4	Pesquisa Operacional	50
2.4.1	<i>Otimização</i>	51
2.4.2	<i>Formulação de Problemas Clássicos de Pesquisa Operacional</i>	52
2.4.3	<i>Problema da Mochila</i>	53
2.5	Avaliação de Desempenho	54
2.5.1	<i>Planejamento de Experimentos</i>	56
2.5.2	<i>Metodologias para Avaliação de Desempenho</i>	56
2.5.3	<i>Complexidade Assintótica de Algoritmos</i>	58
2.5.4	<i>Otimização off-line e Otimização online</i>	59
2.6	Considerações Finais	61

3	TRABALHOS RELACIONADOS	63
3.1	Sistemas Distribuídos e Computação Autônoma	63
3.2	Sistemas Distribuídos e Pesquisa Operacional	64
3.3	Sistemas Distribuídos e Avaliação de Desempenho	65
3.4	Algoritmos e Técnicas para Implementação dos Aspectos da Com- putação Autônoma	66
3.4.1	<i>Algoritmos e Técnicas para Autoconfiguração</i>	66
3.4.2	<i>Algoritmos e Técnicas para Auto-otimização</i>	67
3.4.3	<i>Algoritmos e Técnicas para Autocura</i>	68
3.4.4	<i>Algoritmos e Técnicas para Autoproteção</i>	69
3.4.5	<i>Resumo dos Algoritmos e Técnicas</i>	70
3.5	Categorização dos Trabalhos Relacionados a Computação Autôn- oma	70
3.5.1	<i>Frameworks, Middlewares e Ferramentas de Programação</i>	71
3.5.2	<i>Aplicação da Computação Autônoma em Sistema Específico</i>	72
3.5.3	<i>Modelos de Arquitetura de Sistemas Autônomicos</i>	73
3.6	Análise Crítica dos Trabalhos Relacionados	73
3.7	Considerações finais	76
4	PROCESSO DE DESENVOLVIMENTO DE SOFTWARE FOCADO EM SDAS	79
4.1	Considerações iniciais	79
4.2	Motivação e Objetivos	79
4.3	Conceitos Relacionados	80
4.4	Papéis presentes no PDS2DA	84
4.5	Fases do PDS2DA	85
4.5.1	<i>Iniciação</i>	85
4.5.2	<i>Elaboração</i>	86
4.5.3	<i>Construção</i>	88
4.5.4	<i>Transição</i>	89
4.6	Fluxos de Trabalho	90
4.6.1	<i>Fluxo de Trabalho: Requisitos</i>	90
4.6.2	<i>Fluxo de Trabalho: análise</i>	90
4.6.3	<i>Fluxo de Trabalho: projeto</i>	91
4.6.4	<i>Fluxo de Trabalho: implementação</i>	92
4.6.5	<i>Fluxo de Trabalho: testes</i>	92
4.7	Considerações finais	92
5	ESTUDOS DE CASO	93
5.1	Seleção de Web services baseada em QoS	93

5.2	Aplicação do PDS2DA	94
5.3	Estudo de Caso 1: Otimização <i>off-line</i>	95
5.3.1	<i>Definição dos Requisitos</i>	95
5.3.2	<i>Componentes, papéis e arquitetura</i>	95
5.3.3	<i>Modelagem</i>	96
5.4	Análise	98
5.4.1	<i>Inferência</i>	99
5.4.2	<i>Julgamento</i>	106
5.5	Estudo de Caso 2: O Sistema Distribuído Autônomo em Execução	107
5.5.1	<i>Definição dos Requisitos</i>	107
5.5.2	<i>Componentes, papéis e arquitetura</i>	108
5.5.3	<i>Modelagem</i>	109
5.5.4	<i>Análise</i>	109
5.5.5	<i>Inferência</i>	109
5.5.6	<i>Julgamento</i>	114
5.6	Estudo de Caso 3: Gerenciadores Autônomos para o Sistema Dis-	
	tribuído Autônomo	114
5.6.1	<i>Definição dos Requisitos</i>	116
5.6.2	<i>Componentes, papéis e arquitetura</i>	117
5.6.3	<i>Modelagem</i>	117
5.6.4	<i>Análise</i>	119
5.6.5	<i>Inferência</i>	119
5.6.6	<i>Julgamento</i>	123
5.7	Considerações finais	123
6	CONCLUSÕES E TRABALHOS FUTUROS	125
6.1	Conclusões	125
6.2	Dificuldades relacionadas ao projeto	128
6.3	Contribuições	129
6.4	Trabalhos Futuros	130
6.5	Publicações Oriundas do Trabalho	130
	REFERÊNCIAS	133

INTRODUÇÃO

De acordo com (TANENBAUM; STEEN, 2006), no meio da década de 1980 dois avanços tecnológicos colaboraram para a disseminação dos **Sistemas Distribuídos** (SDs): o desenvolvimento de poderosos microprocessadores e a invenção de redes locais de alta velocidade, conhecidas como **Local-area networks** (LANs), que permitia que centenas de máquinas localizadas dentro de um mesmo prédio trocassem dados em poucos microssegundos. O SD mais conhecido atualmente é, provavelmente, a Internet.

Até a década de 1990, a Internet era usada primordialmente por pesquisadores, acadêmicos e estudantes universitários para se interligar a hospedeiros remotos, transferir arquivos de hospedeiros locais para hospedeiros remotos e vice-versa, enviar e receber notícias e enviar e receber correio eletrônico (KUROSE; ROSS, 2008).

Com o surgimento da **World Wide Web** (Web), proposta por Tim Berners-Lee em 1989, esse paradigma mudou radicalmente. A Web proporcionou uma mudança na maneira de comunicação, integração, disponibilização de informações, execuções de negócios, entretenimento e publicidade por meio da Internet.

No início, a Web, por meio de seus navegadores (*browsers*), exibia apenas conteúdo estático, como: páginas de hipertexto, imagens, sons, entre outros. Entretanto, com o advento da chamada Web 2.0, houve uma mudança e ela começou a fornecer conteúdos dinâmicos e maior interatividade com seus usuários (??).

Ao longo dos anos, diversos modelos de **Computação Distribuída** (CD) foram desenvolvidos e usados. Os modelos de CD que foram e/ou são amplamente usados na Internet são: **Computação em Grupo de Computadores** (CGC) (em inglês, *cluster of computers*), **Computação em Grade** (CG), **Computação em Nuvem** (CN) e **Internet das Coisas** (IdC) (em inglês, *Internet of Things* – IoT). Cada modelo de CD possui seu próprio conjunto de características, e o interesse da indústria e da academia por eles tende a mudar drasticamente com o passar dos anos.

A complexidade de gerenciamento dos SDs na Internet cresce de maneira exponencial. Diversas teorias, técnicas, algoritmos e ferramentas foram desenvolvidas para tentar auxiliar a solução dos problemas relacionados ao gerenciamento de tais sistemas. Uma teoria bastante difundida é o uso da **Computação Autônoma** (CA). A CA é composta basicamente de quatro aspectos: autocura, auto-otimização, autoproteção e autoconfiguração. Esses aspectos podem (e devem) ser aplicados para solucionar os problemas de gerenciamento de SDs.

1.1 Contextualização

Segundo (COULOURIS; DOLLIMORE; KINDBERG, 2007), os SDs podem ser definidos como componentes de *hardware* e *software*, conectados por uma rede, que se comunicam e coordenam suas ações por meio de trocas de mensagens. Provavelmente, o SD mais conhecido é a própria Internet.

Esses SDs têm ficado cada vez mais complexos e difíceis de se gerenciar com o passar dos anos. O paradigma inicial, em que um servidor Web atendia requisições estáticas de clientes, mudou radicalmente. Hoje em dia, os SDs dentro das empresas são compostos por diversas aplicações: servidores Web, servidores de aplicação, **Sistema Gerenciador de Banco de Dados** (SGBD), gerenciadores de carga de trabalho, gerenciadores de **Máquinas Virtuais** (MVs) e assim por diante. Além disso, uma única organização geralmente não possui o controle de todo o sistema. Com a difusão da composição de **Web services** (WSs), um único processo de negócio (por exemplo, uma sequência de ações como, comprar uma passagem aérea, reservar um hotel e pagar com um cartão de crédito) pode ser atendido por diversas empresas, ao invés de ser executado inteiramente dentro de apenas uma. Isso significa que a empresa não tem o controle total da aplicação em questão.

Uma das formas de solucionar o problema da crescente complexidade de gerenciamento e consequente inviabilidade de atender esse problema de maneira não automatizada, é a CA.

Apesar da CA já ter sido usada em centenas de trabalhos relacionados, o processo de criação de **Sistemas Autônomicos** (SAs) ou de conversão de um sistema convencional para um sistema autônomico não possui atualmente um processo de desenvolvimento de *software* amplamente difundido e aceito, que auxilie na criação dos mecanismos de auto-gerenciamento e ainda, permita uma avaliação quantitativa das diferentes soluções implementadas.

1.2 Motivação e objetivos

De acordo com (COULOURIS; DOLLIMORE; KINDBERG, 2007), os SDs possuem como maior motivação o compartilhamento de recursos.

Segundo (GANEK; CORBI, 2003), quando se estuda a raiz do motivo pelo qual os sistemas ficaram indisponíveis, 40% é devido a erros cometidos pelos operadores do sistema.

Além disso, segundo (BHAKTI; ABDULLAH; JUNG, 2010) a complexidade e o dinamismo dos sistemas computacionais na Web cresce exponencialmente.

Conforme (BRAZIER *et al.*, 2009), a complexidade dos sistemas computacionais atuais tem crescido além da capacidade que os administradores humanos conseguem gerenciar. Essa complexidade tem três fontes: (i) componentes individuais dos sistemas computacionais, tais como: gerenciadores de carga de trabalho, SGBDs, etc., que estão cada vez mais difíceis de se configurar, gerenciar e manter, tendo em vista que cada nova versão possui mais recursos e parâmetros configuráveis; (ii) com o uso de WSs e da composição de WSs, os sistemas tem se tornado heterogêneos e não estão mais sob o controle de uma única empresa ou organização e (iii) o pior de todos, mesmo dentro de uma mesma empresa, os sistemas computacionais são heterogêneos, formados por *softwares* de diferentes distribuidores, sendo difíceis de se configurar e manter. Resumindo, a complexidade atual dos sistemas computacionais é maior do que a soma das complexidades de suas partes.

Em (COULOURIS; DOLLIMORE; KINDBERG, 2007) são citados alguns dos principais problemas dos SDs, são eles: heterogeneidade, segurança, escalabilidade, tratamento de falhas, concorrência e transparência.

A CA pode fornecer soluções para os principais problemas e desafios presentes nos SDs. Para citar alguns exemplos, o tratamento de falhas pode ser solucionado com o aspecto de autocura e problemas de segurança podem ser solucionados com o aspecto de autoproteção.

A **Pesquisa Operacional** (PO) tem sido usada como uma forma de modelar diversos problemas relacionados aos SDs, bem como uma forma de solucionar tais problemas (MENASCÉ; BARBARÁ; DODGE, 2001); (CANFORA *et al.*, 2005); (CHECIU *et al.*, 2011); (PRADO *et al.*, 2013) e (EWING; MENASCE, 2014).

Além disso, alguns trabalhos recentes tem usado a PO como uma forma de formalizar e/ou resolver os problemas de SAs (ARDAGNA *et al.*, 2012) e (ALOMARI; MENASCE, 2013). Sendo assim, fica claro que o uso de PO pode contribuir para o desenvolvimento de **Sistemas Distribuídos Autônômicos** (SDAs).

O planejamento de experimentos e a avaliação de desempenho tem sido usado durante décadas, como forma de avaliar diferentes soluções ou implementações para SDs (ZULKERNINE; MARTIN, 2011) e (GERGIN; SIMMONS; LITOIU, 2014).

Ademais, uma parcela considerável dos trabalhos relacionados tem usado alguma forma de avaliação de desempenho no desenvolvimento de SAs, seja por meio de aferição, criação de protótipos e experimentos ou simulação.

Dessa forma, fica claro que os SDAs podem se beneficiar de uma aplicação planejada e correta de técnicas de planejamento de experimentos e avaliação de desempenho.

Por fim, os trabalhos relacionados consultados possuem uma ou mais das seguintes

limitações:

- Aborda apenas um aspecto específico da CA ou no máximo dois.
- É focado em um determinado modelo de CD (*cluster*, *grade*, *nuvem*, etc.).
- Possui uma dependência de uma terminada linguagem de programação e/ou paradigma de programação.
- Fornece etapas para o desenvolvimento de SAs em alto nível, sem explicar a real necessidade de cada etapa e como avaliar as diferentes soluções para o problema de auto-gerenciamento de maneira quantitativa.

Sendo assim, o objetivo geral desta tese de doutorado se concentra na criação de um **Processo de Desenvolvimento de *Software* focado em Sistemas Distribuídos Autônomicos** (PDS2DA). Para cumprir esse objetivo geral, foram definidos os seguintes objetivos específicos, onde se busca criar um processo de desenvolvimento de *software* que:

- Possa ser aplicado a qualquer aspecto da CA ou simultaneamente a dois ou mais;
- Possa ser empregado em qualquer modelo de CD;
- Não seja limitado a uma determinada linguagem de programação e/ou paradigma de programação;
- Permita avaliar quantitativamente diferentes soluções para o problema de auto-gerenciamento do SA e
- Auxilie no desenvolvimento de novos SDAs ou na conversão de um SD em um SDA.

1.3 Estrutura

Esta tese está organizada da seguinte forma:

No Capítulo 2 é apresentada a revisão da literatura. Nela são apresentados os conceitos relacionados ao PU, SDs, CA, PO e ADSC. A ideia não é aprofundar nesses temas, mas sim fornecer uma visão geral, necessária para o entendimento do trabalho que foi desenvolvido.

No Capítulo 3 são apresentados os trabalhos relacionados. A ideia é ter um panorama do que já foi feito em termos de SDs e CA. Nele serão mostradas os diferentes algoritmos e técnicas usados para implementar a CA, bem como alguns exemplos da integração das diferentes áreas do conhecimento do PDS2DA.

O Capítulo 4 mostra o Processo de Desenvolvimento de Software focado em Sistemas Distribuídos Autônomicos. São mostradas suas fases e fluxos de trabalho, como as áreas de conhecimento são integradas e seus principais benefícios.

O Capítulo 5 mostra os três estudos de caso realizados. Eles serviram como prova de conceito da viabilidade do Processo de Desenvolvimento de *Software* focado em Sistemas Distribuídos Autônomicos (PDS2DA) e também como guia de como usá-lo.

Por fim, o Capítulo 6 mostra as conclusões, as dificuldades encontradas e os trabalhos futuros.

REVISÃO DA LITERATURA

Neste capítulo, são mencionados os conceitos referentes as diferentes áreas de conhecimento que compõe o PDS2DA: **Processo Unificado de Desenvolvimento de *Software*** (PU), **Sistemas Distribuídos** (SDs), **Computação Autônômica** (CA), **Pesquisa Operacional** (PO) e **Avaliação de Desempenho de Sistemas Computacionais** (ADSC).

2.1 Processo Unificado

De acordo com (JACOBSON; BOOCH; RUMBAUGH, 2005), existe uma tendência em que os *softwares* se tornem cada vez maiores e mais complexos. Isso ocorre em parte devido ao fato de que os computadores estão ficando mais potentes a cada ano, levando os usuários a esperar mais deles. Outro fator impactante, é o crescente uso da Internet para troca de toda sorte de informações – de texto plano até imagens ou arquivos multimídia. Existe uma crescente demanda por *softwares* que atendam cada vez mais as necessidades dos usuários, o que traz como consequência, a necessidade da criação de *softwares* mais complexos.

Primeiramente, o PU é um processo de desenvolvimento de *software*. Um processo de desenvolvimento de *software* é o conjunto de atividades necessárias para transformar os requisitos do usuário em um sistema de *software*. A Figura 1 mostra o funcionamento do PU (JACOBSON; BOOCH; RUMBAUGH, 2005). Entretanto, o PU é mais do que um simples processo; ele é um *framework* genérico de processo que **pode ser especializado** para uma ampla gama de classes de sistemas de *software*, para diferentes áreas de aplicação, diferentes tipos de organizações, diferentes níveis de competência, e diferentes tamanhos de projetos.

Além disso, O PU é **baseado em componentes**, o que significa que o sistema de *software* em construção será criado usando-se **componentes de *software*** interconectados via interfaces bem definidas. O PU usa a linguagem *Unified Modeling Language* (UML) na criação de todos os seus diagramas, de fato, a UML é uma parte essencial do PU. Entretanto, a principal diferença

Figura 1 – Visão Geral do PU

entre o PU e os demais processos de desenvolvimento de *software*, pode ser resumida em três frases: **conduzido por casos de uso**; **centrado na arquitetura**; e **iterativo e incremental** (JACOBSON; BOOCH; RUMBAUGH, 2005).

2.1.1 O Processo Unificado é conduzido por casos de uso

Segundo (JACOBSON; BOOCH; RUMBAUGH, 2005), um sistema de *software* é criado para servir os seus usuários. Sendo assim, para criar com sucesso um sistema de *software*, é preciso saber o que o usuário precisa e deseja. O termo usuário, refere-se tanto a seres humanos usando o sistema, como também a outros sistemas (fora do sistema sendo desenvolvido) que interagem como sistema sendo desenvolvido. O usuário interage com o sistema (por exemplo, inserindo um cartão bancário e uma senha) e o sistema responde ao usuário, executando uma sequência de ações (por exemplo, verificando se o usuário existe e se a senha está correta). Essa interação é denominada caso de uso.

Um caso de uso é uma parte das funcionalidades do sistema que fornece ao usuário um resultado de valor. Os casos de uso capturam os **Requisitos Funcionais** (RF). O conjunto formado por todos os casos de uso do sistema formam o modelo de casos de uso.

A descoberta dos requisitos tem dois objetivos: descobrir as reais necessidades dos usuários e representá-los de maneira compreensível para usuários, clientes e desenvolvedores. As reais necessidades significam que tais requisitos proverão o valor esperado por seus usuários. A representação de maneira compreensível significa que deve ser usada uma linguagem que seja compreensível por todos os interessados no sistema que será desenvolvido.

2.1.2 O Processo Unificado é centrado na arquitetura

De acordo com (JACOBSON; BOOCH; RUMBAUGH, 2005), a arquitetura do *software* incorpora os aspectos estáticos mais significativos e os aspectos dinâmicos do sistema. A arquitetura se desenvolve a partir das necessidades da empresa, sendo percebida pelos usuários e outras partes interessadas, e sendo refletida nos casos de uso. Entretanto, também é influenciada por muitos outros fatores, como a plataforma em que o *software* irá executar (por exemplo, arquitetura dos computadores, **Sistemas Operacionais** (SOs), SGBDs, componentes disponíveis, considerações de implantação, sistemas legados, e **Requisitos Não-Funcionais** (RNF). A arquitetura é uma visão do sistema como um todo, mostrando as características importantes e deixando

os detalhes de lado. Três conceitos importantes relacionados a arquitetura são (JACOBSON; BOOCH; RUMBAUGH, 2005):

- **O que é arquitetura:** é o que o arquiteto do sistema especifica na descrição da arquitetura. Essa descrição permite que o arquiteto controle o desenvolvimento do sistema de um ponto de vista técnico. A arquitetura do *software* foca tanto nos elementos estruturais do sistema, como subsistemas, classes, componentes e nós, como também nas colaborações que ocorrem entre tais elementos, por meio de interfaces.
- **Como ela é definida:** é desenvolvida de forma iterativa durante a fase de elaboração até os requisitos, análise, projeto e testes. Os casos de uso significativos e uma gama de outras entradas são usadas para criar a base da arquitetura, ou o “esqueleto” do sistema. A gama de entradas inclui os requisitos funcionais, *middlewares*, quais sistemas legados serão usados, requisitos não funcionais e assim por diante.
- **Como ela é descrita:** a descrição da arquitetura é uma visão dos modelos do sistema, visões do modelo de casos de uso, análise, projeto, implementação e implantação. A descrição da arquitetura descreve as partes do sistema que são importantes para todos os interessados no sistema.

2.1.3 O Processo Unificado é iterativo e incremental

De acordo com (JACOBSON; BOOCH; RUMBAUGH, 2005), o desenvolvimento de um produto de *software* é uma grande empreitada que pode levar meses, um ano ou mais. Sendo assim, é prático dividir o trabalho em fatias pequenas ou miniprojetos. Cada miniprojeto é uma **iteração** que resulta em um **incremento**. Iterações se referem a passos no fluxo de trabalho, e incrementos, no crescimento do produto. Para ser eficiente, as iterações devem ser feitas de maneira controlada; isso é, elas devem ser selecionadas e executadas de maneira sistemática.

Desenvolvedores baseiam a escolha do que será implementado em cada iteração em dois fatores. Primeiro, a iteração deve lidar com um grupo de casos de uso que, juntos, estendem a usabilidade do produto. Segundo, a iteração lida com os riscos mais importantes. As iterações sucessivas constroem seus artefatos levando-se em consideração os artefatos existentes na iteração anterior. Cada iteração é um miniprojeto, então a partir dos casos de uso selecionados deve-se executar todos os passos do desenvolvimento - análise, projeto, implementação e testes - que são realizados na forma de código executável. Entretanto, nem todo incremento é necessariamente aditivo. Principalmente nas fases iniciais, os desenvolvedores podem estar substituindo um projeto superficial por um mais detalhado ou sofisticado. Em fases posteriores, incrementos são tipicamente aditivos (JACOBSON; BOOCH; RUMBAUGH, 2005).

Para ser efetivo, o processo de desenvolvimento de *software* deve ter uma sequência clara e bem definida de **marcos** (do inglês, *milestones*) que fornecem aos gerentes e a equipe do

projeto a permissão para passar para a próxima fase do desenvolvimento do *software*. Em cada fase, o processo move-se por uma série de iterações e incrementos que fornecem que levam a esses critérios. Cada fase possui seus próprios critérios (JACOBSON; BOOCH; RUMBAUGH, 2005):

- **Início:** nesta fase, o critério essencial é a viabilidade do projeto. Sendo que o foco deve ser: (i) identificar e reduzir os principais riscos relacionados ao sistema; (ii) criar uma arquitetura candidata, por meio dos principais casos de uso identificados e (iii) fazer uma estimativa inicial de custos, cronograma, esforços e qualidade desejada.
- **Elaboração:** nesta fase, o critério essencial é a habilidade de construir o sistema de uma maneira economicamente viável, para isso, são considerados os seguintes aspectos: (i) identificação e redução dos principais riscos que afetam o sistema em construção; (ii) especificação da maioria dos casos de uso que atendem aos requisitos funcionais; (iii) preparar um plano de projeto detalhado suficientemente para guiar a construção do sistema e (iv) conclusão que o projeto é viável financeiramente.
- **Construção:** nesta fase, o critério essencial é um sistema capaz de ser operado, no ambiente do usuário, avaliado por: uma série de iterações, levando a entregas periódicas e incrementais, sendo que no decorrer desta fase, a viabilidade do sistema é sempre evidente de maneira executável.
- **Transição:** nesta fase, o critério essencial é um sistema que atenda completamente os requisitos funcionais e não-funcionais, avaliado por: (i) modificar o produto para solucionar problemas não encontrados nas fases anteriores e (ii) corrigir defeitos. Resumidamente, um defeito é uma anormalidade do sistema, como uma falha no *software* ou um problema descoberto em uma reunião de revisão. Um defeito pode ser usado para capturar qualquer problema que os desenvolvedores devem registrar como um sintoma de um problema, devendo ser rastreado e solucionado.

2.1.4 A vida do Processo Unificado

O PU repete sobre uma série de ciclos que compõem a vida de um sistema. Cada ciclo termina com uma versão do produto para os clientes. Cada ciclo é composto de quatro fases: iniciação, elaboração, construção e transição. Cada fase é subdividida em iterações, como foi mencionado previamente.

Cada versão do produto é composta por uma série de modelos, que são atualizados a cada iteração, são eles (JACOBSON; BOOCH; RUMBAUGH, 2005):

- Um **modelo de casos de uso**, contendo todos os casos de uso até o momento e suas relações com os usuários.

- Um **modelo de análise** com dois propósitos: para aperfeiçoar os casos de uso com mais detalhes e para fazer uma alocação inicial do comportamento do sistema para um conjunto inicial de objetos que fornece tal comportamento.
- Um **modelo de projeto** que define: **(i)** a estrutura estática do sistema como subsistemas, classes e interfaces e **(ii)** os casos de uso realizados como colaborações entre os subsistemas, classes e interfaces.
- Um **modelo de implementação** que inclui componentes (representando código fonte) e o mapeamento de classes para componentes.
- Um **modelo de implantação**, que define os nós computacionais e o mapeamento entre os componentes para esses nós.
- Um **modelo de teste**, que descreve os casos de teste que verificam os casos de uso.
- E, é claro, uma **representação da arquitetura**.

Ademais, o PU possui fluxos de trabalho, cada um focando em determinadas atividades e artefatos gerados. São eles ([JACOBSON](#); [BOOCH](#); [RUMBAUGH](#), 2005):

- **Requisitos:** O objetivo essencial é guiar o desenvolvimento do sistema correto. Isso é atingido por meio da descrição dos requisitos do sistema (as condições ou capacidades que o sistema deve possuir) de maneira clara o suficiente para que seja possível um consenso entre todas as partes interessadas no sistema (clientes, desenvolvedores, usuários, etc.).
- **Análise:** Na análise, os requisitos capturados são analisados por meio de refinamento e estruturação. Isso é feito visando obter um entendimento mais preciso e obter uma descrição dos requisitos que seja fácil de manter e auxilie em todas as fases do desenvolvimento. Visando esse entendimento mais aprofundado dos requisitos, a linguagem do usuário é deixada de lado, dando lugar a linguagem do desenvolvedor, mais formal e precisa.
- **Projeto:** O sistema é moldado e encontra-se sua forma (incluindo sua arquitetura) que atende a todos os requisitos (funcionais, não-funcionais e demais restrições). Uma entrada essencial para este fluxo são os modelos criados durante a análise. O modelo de análise prove um entendimento detalhado dos requisitos.
- **Implementação:** Inicia-se com o resultado do que foi feito no projeto e implementa-se o sistema em termos dos seus componentes, ou seja, códigos fonte, *scripts*, binários, executáveis e assim por diante. Felizmente, a maior parte da arquitetura do sistema é capturada durante o projeto. O principal objetivo da implementação é desenvolver a arquitetura e o sistema como um todo.

- Testes: São verificados os resultados das implementações de cada entrega, tanto as intermediárias como as finais, bem como a versão final do sistema.

De acordo com (JACOBSON; BOOCH; RUMBAUGH, 2005), o modelo de casos de uso é desenvolvido no decorrer de diversos incrementos, sendo que em cada iteração serão adicionados novos casos de uso e/ou detalhados os já existentes. A Figura 2 mostra como a captura dos requisitos e seus artefatos resultantes assumem diferentes formas durante as diferentes fases e suas iterações. Além disso, mostra também como os diferentes fluxos de trabalho evoluem no decorrer das fases do PU:

Figura 2 – Fluxos de trabalho e fases do PU

2.2 Sistemas Distribuídos

De acordo com (COULOURIS; DOLLIMORE; KINDBERG, 2007), um SD é aquele no qual os componentes de *hardware* ou *software*, localizados em computadores interligados em rede, se comunicam e coordenam suas ações apenas enviando mensagens entre si. Essa definição simples abrange toda a gama de sistemas nos quais computadores interligados em rede podem ser distribuídos de maneira útil.

Além disso, computadores conectados por meio de uma rede podem estar a qualquer distância geográfica imaginável, desde poucos metros (podendo estar ligados por cabos de rede) até outro continente (como é o caso da Internet).

2.2.1 Desafios relacionados aos Sistemas Distribuídos

Segundo (COULOURIS; DOLLIMORE; KINDBERG, 2007), pode-se citar alguns desafios relacionados a criação e manutenção de SDs, são eles:

- **Heterogeneidade:** eles devem ser construídos a partir de uma variedade de redes, SOs, *hardware* e linguagens de programação diferentes. Os protocolos de comunicação da Internet mascaram a diferença existente nas redes e o *middleware* (uma camada de *software* que fornece uma abstração de programação) pode cuidar das outras diferenças.
- **Sistemas abertos:** os SDs devem ser extensíveis – o primeiro passo é publicar interfaces dos componentes, mas a integração de componentes escritos por diferentes programadores é um desafio real.
- **Segurança:** a criptografia pode ser usada para proporcionar proteção adequada para os recursos compartilhados e para manter informações sigilosas em segredo, quando são transmitidas em mensagens por uma rede. Os ataques de negação de serviço ainda são um problema.
- **Escalabilidade:** um SD é considerado escalável se o custo da adição de um usuário for um valor constante, em termos dos recursos que devem ser adicionados. Os algoritmos usados para acessar dados compartilhados devem evitar gargalos de desempenho e os dados devem ser estruturados hierarquicamente para se obter os melhores tempos de acesso. Os dados que são acessados frequentemente podem ser replicados.
- **Tratamento de falhas:** qualquer processo, computador ou rede pode falhar, independentemente dos outros. Portanto, cada componente precisa conhecer as maneiras possíveis pelas quais os componentes de que depende podem falhar e ser projetado de forma a tratar de cada uma dessas falhas apropriadamente.
- **Concorrência:** a presença de múltiplos usuários em um SD é uma fonte de pedidos concorrentes para seus recursos. Em um ambiente concorrente, cada recurso deve ser projetado para manter a consistência nos estados de seus dados.
- **Transparência:** o objetivo é tornar certos aspectos da distribuição invisíveis para o programador de aplicativos, para que este se preocupe apenas com o projeto de seu aplicativo em particular. Por exemplo, ele não precisa estar preocupado com sua localização ou com os detalhes sobre como suas operações serão acessadas por outros componentes, nem se será replicado ou migrado. As falhas de rede e de processos podem ser apresentadas aos programadores de aplicativos na forma de exceções – mas elas devem ser tratadas.

2.2.2 Modelos de Computação Distribuída

De acordo com (TANENBAUM; STEEN, 2006), uma classe importante de SDs é aquela usada para tarefas de **Computação de Alto Desempenho** (CAD). De maneira geral, pode-se dividir em três subgrupos: Grupo de Computadores (*cluster*), Grade Computacional (GC) e Computação em Nuvem (CN). Além disso, devido aos avanços do protocolo IPV6 e o aumento da conectividade dos dispositivos, um novo modelo de CD tem surgido, conhecido como Internet das Coisas. A Figura 3 mostra os diferentes modelos de CD.

Figura 3 – Diferentes modelos de Computação Distribuída

De acordo com (TANENBAUM; STEEN, 2006), esses modelos de CD podem ser definidos da seguinte forma:

- Grupo de Computadores (Computer Cluster):** No GdC, o *hardware* é formado por uma coleção de PCs ou estações de trabalho similares, fortemente conectados por meio de uma conexão de rede LAN. Além disso, cada nó executa a mesma versão do SO. Se tornaram populares quando a relação preço/desempenho dos PCs e estações de trabalho melhoraram. Em determinado momento, passou a ser atrativo do ponto de vista financeiro e técnico, criar um supercomputador, conectando uma série de PCs simples, por meio de uma LAN. Em praticamente todos os casos, um GdC é usado para executar programas que necessitam

de processamento intensivo (*cpu-bound*), sendo que os programas executam de maneira paralela em múltiplas máquinas.

- **Computação em Grade (CG):** A organização do SD é bem diferente de um GdC. Uma GC é construída com o conceito de **Federações de Sistemas Computacionais (FSC)**, sendo que cada FSC pode cair num ambiente administrativo diferente, e pode ser muito diferente em termos de *hardware*, *software* e tecnologia de rede. As GCs, ao contrário dos GdCs possuem em sua maioria sistemas heterogêneos. Nenhuma suposição é feita em termos de SOs, redes, políticas de segurança, entre outros. Uma questão-chave em uma GC é que os recursos computacionais de diferentes organizações são apresentados de maneira conjunta, visando permitir a colaboração de um grupo de pessoas ou instituições. Tal colaboração é realizada por meio de uma **Organização Virtual (OV)**. As pessoas pertencentes à mesma OV tem direitos de acessos aos recursos fornecidos para essa OV.
- **Computação em Nuvem (CN):** Na CN, a computação é fornecida como um serviço. Segundo (BUYYA; CALHEIROS; LI, 2012) as principais categorias de serviços são: (i) **Software as a Service (SaaS) (Software como um serviço)**: trata-se da capacidade do provedor em fornecer, ao usuário, aplicativos em execução em uma infraestrutura de nuvem. **Platform as a Service (PaaS) (Plataforma como um serviço)**: corresponde a capacidade do provedor fornecer ao usuário a possibilidade de implantar aplicações na infraestrutura da nuvem, que são criadas usando linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo provedor e (iii) **Infrastructure as a Service (IaaS) (Infraestrutura como um serviço)**: é fornecida ao usuário a capacidade de provisionamento de processamento, armazenamento, redes e outros recursos computacionais fundamentais. O usuário é capaz de implantar e executar *softwares* arbitrários, os quais podem incluir sistemas operacionais e aplicativos. Provedores de IaaS normalmente oferecem uma infraestrutura virtualizada como serviço ao invés do *hardware* real diretamente.
- **Internet das Coisas:** Segundo (COETZEE; EKSTEEN, 2011), devido a vários avanços tecnológicos, a sociedade está mudando para um paradigma de "sempre conectado". As redes de computadores (tanto com fio e sem fio) estão em toda parte, padrões abertos são definidos e usados (por exemplo, IPV6) permitindo esquemas de endereçamento específicos. Conceitos associados a chamada "Internet do Futuro" estão sendo pesquisados e aplicados. Um novo conceito associado a "Internet do Futuro" é a chamada Internet das Coisas. A "Internet das Coisas" descreve uma visão aonde objetos se tornam parte da Internet: cada objeto é unicamente identificado, e acessível por meio da rede, sua posição e estado é conhecida, sendo que serviços e inteligência são adicionados a essa Internet expandida, fundindo o mundo físico e o mundo digital, ocasionando um impacto na vida profissional, pessoal e social.

2.2.3 Web Services

A *World Wide Web Consortium* (W3C) define um WS como uma aplicação identificada por um Identificador Uniforme de Recursos (IUR), cujas interfaces e ligações são definidas, descritas e descobertas utilizando-se uma linguagem padrão, a *eXtensible Markup Language* (XML) (Linguagem de Marcação Extensível) (FERRIS; FARRELL, 2003). Os WSs constituem uma implementação da AOS (ERRADI; MAHESHWARI, 2005). Pelas definições apresentadas, pode-se observar que diferentes explicações sobre WSs estão disponíveis. Uma delas menciona que as interações entre WSs ocorrem tipicamente com chamadas *Simple Object Access Protocol* (SOAP) (Protocolo de Acesso Simples a Objeto), um protocolo de comunicação baseado em XML para a interação de aplicações (PAPAZOGLU; GEORGAKOPOULOS, 2003). O SOAP é apresentado como um arcabouço para uma nova geração de aplicações de CD, independente de plataforma e de linguagens de programação.

A principal função desse protocolo é encapsular as chamadas aos métodos remotamente distribuídos, que serão transportados, por sua vez, por algum protocolo, tal como o *HyperText Transfer Protocol* (HTTP) (Protocolo de Transferência de HiperTexto), utilizado para comunicação entre um *browser* e um servidor Web (COMER, 2000). Além disso, as descrições de interfaces dos WSs são expressas usando a *Web Service Definition Language* (WSDL) (Linguagem de Definição de Web services) (THOMAS; THOMAS; GHINEA, 2003). Na literatura sobre WSs é apresentado também um protocolo para serviços de diretório, que contém as descrições dos WSs, denominado *Universal Description, Discovery, and Integration* (UDDI) (Descrição, Descobrimiento e Integração Universais). Esse protocolo funciona como um **registro de serviços** sendo um importante componente da AOS (FARKAS; CHARAF, 2003).

A **arquitetura clássica** de WSs é baseada nas interações entre três entidades que são o **provedor de serviços**, o **registro de serviços** e o **consumidor de serviços**. As interações envolvem a **publicação**, a **pesquisa** e a **ligação**.

O provedor de serviços define uma descrição de serviços para o WS e o publica nos registros de pesquisa ou a disponibiliza diretamente para serviços consumidores. Os serviços consumidores utilizam operações de pesquisa para localizar a descrição do serviço localmente ou nos registros de pesquisa e usa a descrição do serviço para conectar-se ao provedor de serviço e invocar, ou interagir, com a implementação do WS. O provedor de serviços é uma construção lógica uma vez que um serviço pode ter características de provedor de serviço e de consumidor de serviço (KREGER, 2001). A Figura 4 mostra a arquitetura padrão dos WSs.

2.2.4 Qualidade de Serviço

O termo *Quality of Service* (QoS) (Qualidade de Serviço) pode ser visto de diversas formas de acordo com a área de aplicação. Por exemplo, do ponto de vista de uma rede de comutação de circuitos, refere-se à probabilidade de sucesso em estabelecer uma ligação a

Figura 4 – Modelo de arquitetura clássica de Web services

Fonte: Adaptado de (KREGER, 2001).

um determinado destino. Do ponto de vista de uma rede de comutação de pacotes a QoS refere-se à capacidade de uma rede prover melhores serviços para um dado tráfego selecionado, podendo utilizar várias tecnologias incluindo *Frame Relay*, *Asynchronous Transfer Mode* (ATM), *Ethernet*, redes 802.11, *Synchronous Optical Networking* (SONET) e redes *Internet Protocol* (IP) (Protocolo da Internet) (VESEGNA, 2001). Entretanto, nesta tese de doutorado abordaremos a QoS em SDs, por isso a QoS em nível de rede não será explorada.

No contexto de WSs, QoS é referenciada como um conjunto de **propriedades não-funcionais**, tais como desempenho, confiabilidade, disponibilidade e segurança. Com a difusão dos WSs, medidas de QoS são utilizadas como uma aferição para diferenciar WSs (KALEPU; KRISHNASWAMY; LOKE, 2003). No entanto, garantir QoS em aplicações Web não é uma tarefa trivial devido às características imprevisíveis da Internet (FARKAS; CHARAF, 2003). Assim, alguns parâmetros de QoS, de acordo com (KALEPU; KRISHNASWAMY; LOKE, 2003) são:

- **Disponibilidade:** é um aspecto de qualidade de serviço no qual um WS está presente, ou pronto, para uso imediato, representado como uma porcentagem de tempo disponível de um serviço em um período de observação e está relacionado com sua confiabilidade. Para serviços frequentemente acessados, um pequeno valor de período de observação fornece uma aproximação mais precisa para sua disponibilidade.
- **Custo:** o valor monetário cobrado pelo provedor do WS pelo acesso feito ao serviço.
- **Tempo de resposta:** corresponde ao tempo que uma requisição demora a partir do momento em que ela é iniciada para ser atendida até o momento em que o requisitante recebe a resposta dessa requisição. Esse tempo inclui toda sobrecarga e possíveis atrasos que ocorram na rede.

- **Reputação:** corresponde a um parâmetro medido pela avaliação que os clientes do WS atribuem a ele. No final de cada acesso, isso pode ser coletado por meio de uma enquete sobre a satisfação do cliente.
- **Confidencialidade:** de acordo com (KUROSE; ROSS, 2008) esse atributo de QoS faz parte da segurança do WS e significa que somente o remetente e o destinatário pretendido devem poder entender o conteúdo da mensagem transmitida.

Nos trabalhos relacionados, alguns outros atributos de QoS são mencionados, mas os citados acima ocorrem com certa frequência e por isso foram selecionados como exemplos.

2.2.5 Composição de Web Services com Qualidade de Serviço

De acordo com (WANG; TIAN, 2009) a popularidade crescente do uso de WSs para SDs contribui para a importância da descoberta de serviços. No entanto, podem existir vários WSs com a mesma funcionalidade, de forma que é necessário utilizarem-se de critérios adicionais para a seleção dos WSs que serão considerados em uma composição. A **Composição de Web Services baseada em atributos de QoS (CWSbQ)** é uma questão prática bastante atrativa e desafiadora. Tendo em vista que cada WS possui seus próprios atributos de QoS, são necessárias funções de agregação para tais atributos, de forma que seja possível avaliar os valores de QoS da composição como um todo (KO; KIM; KWON, 2008) e (ZHI-PENG *et al.*, 2009). A Tabela 1 apresenta um exemplo desses atributos de agregação:

Tabela 1 – Funções de agregação dos atributos de QoS

Disponibilidade	$\prod_{i=1}^{i=n} disponibilidade(WS_i)$
Custo	$\sum_{i=1}^{i=n} custo(WS_i)$
Tempo de Resposta	$\sum_{i=1}^{i=n} tempoResposta(WS_i)$
Reputação	$\sum_{i=1}^{i=n} reputacao(WS_i) * 1/n$
Confidencialidade	$\sum_{i=1}^{i=n} confidencialidade(WS_i) * 1/n$

Na Tabela 1 pode-se observar que os atributos de QoS custo e tempo de resposta da composição são apenas a somatória dos valores individuais de cada WS. O atributo disponibilidade é dado pela multiplicação dos valores de cada WS. Finalmente os atributos de reputação e confidencialidade representam uma média dos valores de cada WS.

Essas funções de agregação são utilizadas quando o fluxo de execução da composição é sequencial, sendo que para diferentes tipos de fluxos são utilizadas outras funções de agregação. De acordo com (CANFORA *et al.*, 2005) e (ARDAGNA; PERNICI, 2005) o problema de composição de WSs baseado-se em atributos de QoS, é um problema de otimização combinatória, com complexidade **NP-Difícil**, pois é equivalente ao problema da **Mochila Multi-dimensional com Múltiplos Objetivos (MMMO)**.

Segundo (ZHANG; MA, 2009) alguns WSs candidatos (WSs concretos) com as mesmas funções e diferentes valores de atributos de QoS são descobertos para cada tarefa a ser executada (WSs abstratos). Se o fluxo tiver 10 WSs abstratos e 15 WSs concretos para cada WS abstrato, têm-se 15^{10} possíveis planos de composição.

De acordo com o trabalho de (KO; KIM; KWON, 2008) é esperado que existam centenas de WSs concretos com as mesmas funcionalidades e diferentes valores para os atributos de QoS. Além disso, o uso de WSs vem crescendo a cada ano, dessa forma, seria razoável imaginar um cenário com cerca de 150 WSs concretos para cada WS abstrato, supondo-se um fluxo com 10 WSs abstratos se teria 150^{10} possíveis composições, algo claramente inviável de se calcular através de algoritmos de busca exaustiva.

Em (FANJIANG *et al.*, 2010), a composição de WSs é separada em **seleção baseada em QoS** e **criação da orquestração**. A seleção baseada em QoS, é a atividade mencionada acima, sendo um problema de otimização combinatória. A criação da orquestração, define a ordem em que os Web services deve ser executada, o fluxo de dados entre os Web services e a estrutura de cada atividade. Isso geralmente é feito usando-se uma linguagem de modelagem de processos de negócios, como a WS-BPEL (*Web Services Business Process Execution Language*). Nesta tese, focaremos na seleção de Web services baseada em QoS. A Figura 5 exibe os diferentes aspectos envolvidos na composição de Web services baseada em QoS.

Figura 5 – Aspectos da Composição de Web services baseada em QoS

Fonte: Adaptado de (FANJIANG *et al.*, 2010).

2.3 Computação Autônômica

A ideia da CA surgiu em 2001 quando Paul Horn, Vice-Presidente e Diretor de Pesquisa da IBM apresentou o tema e sua importância na Academia Nacional de Engenharia da Universidade de Harvard. Em sua fala, ficou claro que o próximo grande desafio da área de

Tecnologia da Informação está relacionado com a complexidade de gerenciar os diversos recursos computacionais presentes nas empresas, tanto o *hardware*, como o *software*.

De acordo com (GANEEK; CORBI, 2003) a automação do gerenciamento de recursos computacionais não é um problema novo para os cientistas da computação. Por décadas, componentes dos sistemas e *softwares* têm evoluído para lidar com a crescente complexidade de controlar os sistemas, compartilhar recursos, e realizar o gerenciamento operacional. A diferença entre as medidas já adotadas há várias décadas atrás e a nova forma de lidar com esses problemas de gerenciamento, que tem sido denominada de CA está relacionado ao fato do crescimento em larga escala, o alcance mais amplo e missões mais críticas relacionadas ao atendimento das demandas das empresas.

2.3.1 Aspectos Fundamentais da Computação Autônômica

Segundo (JACOB; LANYON-HOGG; YASSIN, 2004), o termo “autônômica” vem de uma analogia com o sistema nervoso central autônômico do corpo humano, que se ajusta a muitas situações automaticamente, sem uma ajuda externa. Ao subirmos um lance de escadas, nossos batimentos cardíacos aumentam. Se estivermos com calor, transpiramos. Não é necessária uma ação consciente para que isso aconteça, elas reações simplesmente ocorrem sem nossa intervenção. Similarmente, a ideia da CA é que os sistemas computacionais possam se auto gerenciar sem intervenção humana. De acordo com (JACOB; LANYON-HOGG; YASSIN, 2004), os sistemas autônômicos são constituídos de quatro atributos, são eles:

- **Autoconfiguração** (*Self-configuring*): com a capacidade de se autoconfigurar dinamicamente, um ambiente de TI pode se adaptar imediatamente, com o mínimo de intervenção na implantação de novos componentes ou mudanças nos componentes existentes.
- **Autocura** (*Self-healing*): permite que o sistema possa detectar operações problemáticas e então iniciar uma ação corretiva. Isso pode significar que o elemento irá modificar seu próprio estado ou ainda modificar o estado de outros componentes.
- **Auto-otimização** (*Self-optimizing*): se refere à habilidade do sistema alocar eficientemente seus recursos e utilização, de forma a atender as necessidades dos usuários finais com um mínimo de intervenção. No curto prazo, a auto-otimização aborda principalmente a capacidade de gerenciar o desempenho do sistema. No longo prazo, componentes auto-otimizáveis podem aprender com a experiência e automaticamente e pro-ativamente se adaptar ao contexto em que estão inseridos.
- **Autoproteção** (*Self-protection*): permite que pessoas autorizadas tenham acesso aos dados corretos e nos momentos corretos. Além disso, também toma ações apropriadas para automaticamente se tornar menos vulnerável à ataques de pessoas não autorizadas.

2.3.2 Arquitetura de um Sistema Autônômico

De acordo com (KEPHART; CHESS, 2003), um sistema computacional autônômico possui um ou mais **Elementos Autônômicos** (EAs) que interagem entre si. A arquitetura desse EA é dividida em: **Gerenciador Autônômico** (GA) e **Recurso Gerenciável** (RG). O GA é responsável por controlar seus respectivos RGs, baseando-se em um laço de controle fechado “*closed control loop*”, conhecido como MAPE-C. De uma forma geral, pode-se dizer que as atividades exercidas por cada elemento do laço de controle MAPE-C são:

- **Monitorar** (*Monitor*): esse elemento permanece coletando dados brutos do sistema autônômico em questão. Quais dados são relevantes e devem ser coletados depende da natureza do sistema autônômico e também quais aspectos da CA estão sendo implementados. Por exemplo, um servidor Web que possui o aspecto de ser auto-otimizável, provavelmente possuirá um monitor que coleta dados sobre os tempos de resposta de cada requisição atendida.
- **Analisar** (*Analyze*): em um determinado intervalo de tempo, o componente de Análise verifica os dados coletados pelo monitor. No exemplo do servidor Web, o componente responsável pela análise verificará periodicamente o tempo médio de resposta das requisições atendidas, passando essa informação para o componente Planejador, caso seja necessária alguma ação ser tomada.
- **Planejar** (*Plan*): esse componente é que decide quais ações deverão ser tomadas, quando é necessário modificar o sistema para que ele atinja um estado desejado. Por exemplo, o Planejador pode decidir que é necessário aumentar o número de *threads* disponíveis ou recusar algumas requisições que estão aguardando na fila.
- **Executar** (*Execute*): esse componente é o que realmente toma as atitudes. Após receber o plano que foi elaborado pelo componente Planejar, ele executa as ações que foram definidas.
- **Conhecimento** (*Knowledge*): esse componente possui o conhecimento sobre o sistema e sobre suas regras de funcionamento e possíveis gatilhos para ações. O conhecimento pode estar armazenado de inúmeras formas diferentes, variando de um simples banco de dados até complexas ontologias e regras semânticas. O armazenamento de atributos de QoS de WSs usando web semântica foi abordado nos trabalhos de (NAKAMURA *et al.*, 2015) e (NAKAMURA *et al.*, 2014).

A Figura 6 mostra o funcionamento do laço de controle MAPE-C. É importante salientar que esse laço de controle é permanente e continua ativo enquanto o sistema autônômico estiver em execução.

Figura 6 – Laço de Controle MAPE-C

2.3.3 Modelos de Arquiteturas de Sistemas Autônomicos

De acordo com (GUINEA; SAEEDI, 2012), existem basicamente duas formas de se implementar o gerenciamento autônomo: **centralizado** ou **distribuído**. No **gerenciamento centralizado**, as decisões são executadas de forma ordenada e os riscos de conflitos são baixos, o que torna o sistema como um todo mais previsível. Ele pode fornecer soluções ótimas ou sub-ótimas, tendo em vista que analisa o sistema de uma maneira global. As desvantagens são: ponto único de falha, falta de escalabilidade, risco de se tornar o gargalo do sistema, entre outras.

No **gerenciamento distribuído**, muitos elementos do sistema compartilham informações que podem ser usadas para tomar decisões. Nessa abordagem, nenhum elemento tem conhecimento total do sistema como um todo. As vantagens são: maior escalabilidade e não possuir um ponto único de falha. As desvantagens são: sobrecarga de comunicação, maior probabilidade de conflitos (pois não há um único gerenciador que conhece todo sistema) e ainda, por não ter uma visão global do sistema, a probabilidade de conseguir uma solução ótima ou sub-ótima é reduzida.

Essa visão de modelos de arquitetura centralizado ou distribuído é interessante, porém incompleta. De fato, em alguns trabalhos relacionados, o que foi implementado é um modelo de arquitetura que parece ser um híbrido entre centralizado e distribuído. Esse modelo de arquitetura é frequentemente denominado de **modelo hierárquico**. No trabalho de (AL-SHISHTAWY *et al.*, 2008), é citado um modelo de arquitetura que possui um grupo de gerenciadores autônomos que controlam apenas uma parte do sistema, ou ainda, apenas um aspecto da computação autônoma que deve ser considerado (por exemplo, um GA para cuidar dos servidores Web ou um GA

responsável pelo aspecto de auto-cura), sendo que esses GAs passam informação para um GA central, hierarquicamente acima, que tem uma visão do sistema como um todo. Por exemplo, suponha um sistema formado por um grupo de servidores Web, um grupo de SGBDs e um grupo de servidores de aplicação. Pode-se ter um GA para cada grupo, que coleta as informações e envia para um gerenciador central, que toma as decisões e repassa para os GAs inferiores executarem. A Figura 7 exibe um modelo de arquitetura centralizado, a Figura 8 exibe um modelo de arquitetura distribuído e a Figura 9 exibe um modelo de arquitetura hierárquico. A Tabela 2 mostra um resumo dos modelos de arquitetura existentes.

Figura 7 – Modelo de arquitetura centralizado

Figura 8 – Modelo de arquitetura distribuída

2.3.4 Computação Autônoma e Sistemas Distribuídos

É interessante observar como a CA pode ser usada para resolver problemas de auto-gerenciamento de SDs. De acordo com (TANENBAUM; STEEN, 2006), SDs devem possuir

Figura 9 – Modelo de arquitetura hierárquica

Tabela 2 – Modelos de arquitetura de um Sistema Distribuído Autônomo

Arquitetura	Características	Vantagens	Desvantagens
Centralizada	Decisões executadas de forma ordenada, um único gerenciador tem a visão e controle de todo o sistema.	Baixo risco de conflitos, pode fornecer soluções ótimas ou sub-ótimas.	Ponto único de falha, falta de escalabilidade e risco de se tornar o gargalo do sistema.
Distribuída	Muitos elementos compartilham informações que podem ser usadas para o auto-gerenciamento. Nenhum elemento tem a visão e/ou controle total do sistema.	Maior escalabilidade e não possui um ponto único de falha.	Sobrecarga de comunicação, maior probabilidade de conflitos e dificuldade em garantir uma solução ótima ou sub-ótima.
Hierárquica	É uma combinação da centralizada e distribuída. Vários gerenciadores autônomos se reportam a um gerenciador central	Combina as vantagens da arquitetura centralizada e da distribuída.	Pode sofrer com sobrecarga de comunicação, além do custo computacional de se manter vários gerenciadores.

Tabela 3 – Resumo das técnicas usadas para problemas de sistemas distribuídos

Problema dos SDs	Aspecto da CA
Segurança	Autoproteção.
Escalabilidade	Autoconfiguração/auto-otimização.
Tratamento de falhas	Autocura.
Vazão	Autoconfiguração/auto-otimização.
Balanceamento de carga	Autoconfiguração/auto-otimização.
QoS	Autocura/autoconfiguração/auto-otimização/autoproteção.

a capacidade de serem auto-gerenciáveis. Para isso, deve-se organizar os SDs como sistemas de controle de alto nível, por meio de laços de controle retroalimentados, permitindo uma adaptação automática do SD, de acordo com a necessidade que ocorra em tempo de execução. Essa habilidade, é justamente fornecida pela CA.

Segundo (TANENBAUM; STEEN, 2006), existem diferentes visões sobre sistemas auto-gerenciáveis, mas o que eles possuem em comum (implicitamente ou explicitamente) é que devem ser retroalimentados por um ou mais laços de controle, tais sistemas são conhecidos como Sistemas Baseados em Laços de Controle Retroalimentados. Laços de controle tem sido aplicados em diversos campos da engenharia, e seu embasamento matemático também tem sido aplicado aos sistemas computacionais. Para sistemas auto-gerenciáveis, os aspectos arquiteturais são os mais importantes inicialmente.

O núcleo do SD é composto pelos componentes que deverão ser gerenciados. Esses componentes possuem parâmetros configuráveis que podem ser modificados. Entretanto, alguns parâmetros que não podem ser controlados pelo laço de controle também podem influenciar no desempenho do SD. Por exemplo, a carga de trabalho injetada no SD ou ainda, o atraso de rede da Internet, são parâmetros que estão fora do controle do SD ou do laço de controle, sendo assim, tais parâmetros são chamados de variáveis externas.

Além do núcleo do SD, o laço de controle é composto por três componentes: um módulo de monitoramento, que executa a tarefa de medir e/ou estimar as métricas relevantes do SD, um módulo de análise, que deve decidir, baseado em valores de referência, se o sistema deve ou não ser modificado e por fim, um módulo executor, que efetivamente modifica o núcleo do SD, visando adapta-lo a situação de execução atual.

Também é interessante notar como a CA pode ser usada para solucionar os principais problemas dos SDs. Como já foi mencionado no capítulo sobre SDs, alguns dos principais problemas relacionados aos SDs são: heterogeneidade, segurança, escalabilidade, tratamento de falhas, concorrência e transparência. Cada um desses problemas pode ser solucionado por um ou mais dos aspectos da CA. O capítulo de trabalhos relacionados fornecerá uma série de exemplos de como isso tem sido feito atualmente. A Tabela 3 fornece alguns exemplos, relacionando os problemas de SDs aos aspectos da CA que podem ser usados para solucioná-los.

Em termos de **segurança**, uma série de trabalhos tem usado o aspecto de autoproteção para análise e detecção de intrusos. Eles serão citados no capítulo de trabalhos relacionados.

A **escalabilidade** pode ser resolvida usando-se os aspectos de autoconfiguração e auto-otimização. Um exemplo de uso da **autoconfiguração** é o serviço da EC2. Usando PEs (por exemplo, uso da CPU), o número de MVs pode aumentar ou diminuir automaticamente. Ademais, a **auto-otimização** pode ser usada em conjunto, visando buscar a melhor configuração possível.

O **tratamento de falhas** pode ser resolvido usando-se o aspecto de autocura. Alguns exemplos serão citados no capítulo de trabalhos relacionados.

A **vazão** pode ser melhorada usando-se técnicas de autoconfiguração e auto-otimização. Um exemplo clássico seria um escalonador/balanceador de carga autônomo. Se a quantidade de servidores for fixa, pode-se empregar a auto-otimização. Caso seja variável, pode-se usar a auto-otimização em conjunto com a autoconfiguração, para decidir as quantidades de servidores adequadas em cada situação. Esse também é um exemplo de como resolver um problema de **balanceamento de carga**.

Os atributos de QoS são variados, sendo assim, cada atributo poderia ser atendido por um aspecto diferente da CA. Como exemplo, pode-se usar a autoproteção para aumentar a disponibilidade do sistema (protegendo contra ataques) ou a auto-otimização focando-se em minimizar o tempo médio de execução das requisições.

2.4 Pesquisa Operacional

Segundo ([KELLERER; PFERSCHY; PISINGER, 2004](#)), cada aspecto da vida humana é crucialmente determinado pelo resultado de decisões. Enquanto decisões pessoais podem ser baseadas em emoções e gostos pessoais, o complexo ambiente profissional do século 21 requer um processo de decisão que possa ser formalizado e validado independentemente das pessoas envolvidas. Dessa forma, uma formalização quantitativa de todos os fatores influenciando uma decisão e também do resultado de cada decisão deve ser buscado.

A PO pode ser definida como o desenvolvimento de métodos científicos de sistemas complexos, com a finalidade de prever e comparar estratégias ou decisões alternativas. O objetivo é dar suporte à definição de políticas e determinação de ações de forma científica. De maneira sucinta, pode-se dizer que a PO é um enfoque científico sobre a tomada de decisões ([ARENALES et al., 2007](#)).

De acordo com ([ARENALES et al., 2007](#)), fazer ciência é a capacidade de observar e descrever fenômenos naturais, sociais, econômicos, entre outros, tendo a matemática uma importância fundamental na descrição desses fenômenos. A partir da observação dos fenômenos, processos ou sistemas, buscam-se leis que os regem. Essas leis, se passíveis de serem descritas por relações matemáticas, dão origem aos modelos matemáticos. O termo modelo é usado

como objeto abstrato, que procura imitar as principais características de um objeto real para fins de representação do objeto real. Sendo assim, uma versão simplificada do sistema ou problema real é criada, visando captar os elementos essenciais do sistema, para que as soluções desenvolvidas para esse modelo matemático possam ser aplicadas ou adaptadas para o sistema real. A Figura 10 ilustra um processo simplificado da abordagem de solução de um problema usando a modelagem matemática. O processo simplificado de modelagem matemática possui as seguintes fases (ARENALLES *et al.*, 2007):

- **Formulação (modelagem):** define as variáveis e as relações matemáticas para descrever o comportamento relevante do sistema ou do problema real.
- **Dedução (análise):** aplica técnicas matemáticas e tecnologia para resolver o modelo matemático e visualizar quais conclusões ele sugere.
- **Interpretação (inferência):** argumenta que as conclusões retiradas do modelo têm significado suficiente para inferir conclusões ou decisões do problema real.
- **Avaliação (julgamento):** as conclusões ou decisões inferidas são avaliadas de acordo com sua adequação. Caso não sejam adequadas, a definição do problema e sua modelagem matemática passam por uma revisão e, então, o ciclo é repetido.

Figura 10 – O Processo de modelagem usando Pesquisa Operacional

Fonte: Adaptado de (ARENALLES *et al.*, 2007)

2.4.1 Otimização

Otimização é um processo que, dado um conjunto de objetivos e restrições, visa a encontrar uma ou mais soluções que maximizem (ou minimizem) os objetivos e, ao mesmo tempo, respeitem as restrições impostas. Problemas de otimização existem nos mais variados domínios, sendo frequentemente resultantes do mapeamento de uma situação do mundo real para um modelo matemático. Por exemplo, uma fábrica pode empregar otimização para descobrir

a melhor estratégia de produção e venda, buscando obter lucro máximo. Ou então uma equipe de administradores pode empregar otimização para decidir a melhor forma de aplicar recursos (GIUSTI, 2010).

No caso mais simples, o problema de otimização é modelado como um conjunto de **Variáveis de Decisão** (VD) e uma **Função-Objetivo** (FO). As **variáveis de decisão** são parâmetros relacionados ao problema em questão, por exemplo: número de peças fabricadas ou montante investido em um fundo. A FO é uma função que associa as VD a um valor real. Uma solução é um vetor de valores associados às VD que descreve uma possível estratégia de resolução do problema de otimização, sendo uma solução ótima aquela para qual a função objetivo assume o valor máximo ou mínimo possível, dependendo do problema em questão. Soluções que não são ótimas são chamadas sub-ótimas (GIUSTI, 2010). Um exemplo de modelo para esse tipo de problema de otimização é exibido na Figura 11.

Figura 11 – Função a ser otimizada.

$$\begin{aligned} (\min) \quad & f(x, y) = x^2 + y^2 \\ & x, y \in \mathbb{R} \\ & f : \mathbb{R}^2 \rightarrow \mathbb{R} \end{aligned}$$

Fonte: (GIUSTI, 2010)

A leitura desse modelo é a seguinte: deseja-se encontrar um par de valores reais (x, y) tal que o valor da função f seja o menor possível para esse par. Em outras palavras, deseja-se minimizar a função f (GIUSTI, 2010).

Duas possíveis soluções para o problema em questão são, por exemplo, $S_1 = (2, 2)$ e $S_2 = (0, 0)$. Observe que $f(S_1) = 8$ e $f(S_2) = 0$. Como trata-se de um problema de minimização, então a solução S_2 é melhor que a solução S_1 . É possível verificar também que não existe uma solução melhor do que S_2 , observando a Figura 12. Sendo assim, pode-se dizer que S_2 é a solução ótima.

2.4.2 Formulação de Problemas Clássicos de Pesquisa Operacional

Conforme (ARENALES *et al.*, 2007), linguagens algébricas, criadas a partir da década de 1980, tiveram um grande impacto prático para a resolução de problemas de otimização. Essas linguagens de alto nível permitem que o usuário escreva modelos genéricos de programação linear e não-linear em um formato parecido com o da notação algébrica. Existem diversas linguagens populares, tais como: GAMS, AMPL, MPL, AIMMS e assim por diante. A ideia, portanto, é que dado um determinado problema de otimização, busque-se identificar, se ele pode ser definido com uma aplicação de um problema clássico de PO. Existem uma série de linguagens, ferramentas e algoritmos para a resolução de problemas clássicos de PO, sendo assim, se for possível classificar o problema que deverá ser solucionado dentro de algum problema

Figura 12 – Espaço de decisão do problema

Fonte: (GIUSTI, 2010)

clássico, tem-se a possibilidade do uso de uma solução pronta, o que permite evitar erros e economizar tempo, levando-se em último caso, a uma economia de recursos financeiros.

A quantidade de problemas clássicos conhecidos é enorme, sendo que existem livros inteiros dedicados a determinadas classes de problemas de PO. Em (KELLERER; PFERSCHY; PISINGER, 2004) são abordadas variações do Problema da Mochila (*Knapsack Problem*), que está dentro da categoria de Otimização Combinatória (OC), que por sua vez faz parte da Otimização Discreta.

2.4.3 Problema da Mochila

Para explicar os conceitos por trás dos problemas da mochila, primeiramente será usado uma linguagem informal, para facilitar o entendimento. Após isso, o problema será formalizado, por meio de um modelo matemático. Isso ajuda a identificar quais são as variáveis de decisão, qual a função que deverá ser otimizada, bem como as funções de restrição. Em problemas do mundo real, geralmente inicia-se por uma descrição textual e em alto nível desses três aspectos: VD, FO e restrições, para depois iniciar o processo de modelagem matemática. Uma vez que o problema tenha sido formalizado matematicamente, deve-se então implementar uma ou mais soluções para o problema em questão.

Suponha um vendedor ambulante que deseja encher sua mochila com itens que venderá posteriormente. Ele possui uma mochila e vários itens que pode colocar dentro dela. Cada item fornece um lucro ao ser vendido, que é medido por um número positivo p_j e possui um peso w_j . Por razões óbvias, o vendedor tem um limite de peso que pode carregar, digamos o limite c . Sendo assim, para cada item n disponível que o vendedor possui, ele deve decidir se coloca

dentro da mochila ou não, visando maximizar o lucro total, porém sem ultrapassar o limite de peso que pode carregar. De acordo com (KELLERER; PFERSCHY; PISINGER, 2004), esse é o caso mais simples de problema da mochila, conhecido como **Problema da Mochila Binário** (PMB), ou problema da mochila 0-1. A formalização é a seguinte: possui-se um conjunto N de itens, que contem n itens j com lucro p_j e peso w_j e a mochila possui uma capacidade c . Sendo assim, o objetivo é escolher um subconjunto de N , cujo benefício total seja o máximo possível, desde que o peso não ultrapasse a capacidade máxima c . A Equação 2.1 mostra o modelo matemático do problema.

De acordo com (KELLERER; PFERSCHY; PISINGER, 2004), **fortes evidências** indicam que não existe um algoritmo que consegue executar em tempo polinomial e garante a solução ótima para o Problema da Mochila e suas variantes. Em termos de complexidade assintótica de algoritmos, isso o enquadra na classe de problemas NP-Difícil (*NP-Hard*).

$$\begin{aligned} \text{Maximizar} \quad & \sum_{j=1}^n p_j x_j \\ \text{Sujeito a:} \quad & \sum_{j=1}^n w_j x_j \leq c, \\ & x_j \in \{0, 1\}, \quad j = 1, \dots, n. \end{aligned} \tag{2.1}$$

2.5 Avaliação de Desempenho

Usuários de computadores, administradores e projetistas demonstram interesse em avaliação de desempenho, uma vez que o objetivo da técnica é obter resultados que possibilitem atingir ou prover o maior desempenho possível com o menor custo. A avaliação de desempenho é necessária em todos os estágios do ciclo de vida de um *software*, incluindo o projeto, manufatura, vendas, uso, atualização e assim por diante. Ela também é necessária quando o projetista de *software* deseja comparar um número de alternativas para o projeto e escolher a melhor entre elas (JAIN, 1991). Segundo (JAIN, 1991) os passos necessários para se realizar uma avaliação de desempenho sistemática são:

- **Definir os objetivos e o sistema:** O primeiro passo em qualquer avaliação de desempenho é definir os objetivos do estudo e definir o que constitui o sistema, definindo suas fronteiras. Levando-se em conta o mesmo conjunto de *hardware* e *softwares*, a definição do sistema pode variar dependendo dos objetivos que se quer alcançar com o estudo.
- **Listar serviços e saídas:** Quando um usuário solicita qualquer um desses serviços, existe uma série de saídas possíveis. Algumas dessas saídas são desejáveis e outras não. Por exemplo, um SGBD pode responder a uma consulta corretamente, incorretamente (devido a atualizações inconsistentes), ou nenhum dos dois (devido a *deadlocks* ou problemas

do gênero). Uma lista de serviços e possíveis saídas serão úteis posteriormente, para selecionar as métricas e cargas de trabalho corretas.

- **Selecionar Métricas:** O próximo passo é selecionar os critérios para comparar o desempenho. Esses critérios são chamados métricas. Em geral, as métricas são relacionadas com velocidade, precisão e disponibilidade dos serviços.
- **Listar Parâmetros:** O próximo passo em projetos de avaliação de desempenho é criar uma lista de todos os parâmetros que afetam a performance. A lista pode ser dividida em parâmetros do sistema e parâmetros de carga de trabalho. A lista de parâmetros pode não estar completa. Isto é, após o primeiro passo da análise, pode-se descobrir que existem parâmetros adicionais que podem afetar o desempenho. Pode-se então adicionar esses parâmetros na lista, mantendo-a sempre o mais coerente possível. Isso permite ao analista e aos tomadores de decisão discutir o impacto de vários parâmetros e determinar quais dados precisam ser coletados antes ou durante a análise.
- **Selecionar Fatores para Estudo:** A lista de parâmetros pode ser dividida em duas partes: aqueles que serão variados durante a avaliação e aqueles que permanecerão constantes. Os parâmetros que serão variados são denominados fatores e seus valores são denominados níveis. Em geral, a lista de fatores, e seus possíveis níveis são maiores do que os recursos disponíveis permitem.
- **Selecionar a Técnica de Avaliação:** As três abordagens para avaliação de desempenho são: modelagem com solução analítica ou por simulação e aferições em sistemas reais. A seleção da técnica correta depende do tempo e recursos disponíveis para resolver o problema e o nível desejado de precisão.
- **Selecionar a Carga de Trabalho:** A carga de trabalho consiste de uma lista de requisições de serviços ao sistema. Por exemplo, a carga de trabalho para comparar diferentes SGBDs pode se constituir de uma série de consultas. Dependendo da técnica de avaliação escolhida, a carga de trabalho pode ser expressa de diferentes formas.
- **Planejamento de Experimentos:** Uma vez que se tenha uma lista de fatores e seus níveis, é necessário decidir a sequência de experimentos que forneça o máximo de informação com o mínimo de esforço.
- **Analisar e Interpretar Dados:** É importante reconhecer que os resultados das medidas e simulações são quantidades aleatórias em que os resultados poderão ser diferentes cada vez que a experiência for repetida.
- **Apresentar Resultados:** O passo final para todos os projetos de desempenho é comunicar os resultados para outros membros do time de tomadores de decisão. É importante que os resultados sejam apresentados de uma forma que seja fácil de entender. Isso geralmente

necessita que os resultados sejam apresentados em forma de gráficos e sem jargões estatísticos. Os gráficos devem ser apropriadamente escalados.

2.5.1 *Planejamento de Experimentos*

O objetivo de um planejamento de experimentos adequado é obter o máximo de informação com o mínimo de experimentos. Isso economiza um trabalho considerável que teria sido gasto com a coleta dos dados. A análise adequada de experimentos também ajuda a separar os efeitos de vários fatores que podem afetar o desempenho. Além disso, permite determinar se um fator tem um efeito significativo ou se a diferença observada é simplesmente devido a variações aleatórias causadas por erros de medição e parâmetros que não foram controlados (JAIN, 1991).

Os seguintes termos são comumente utilizados no planejamento de experimentos: **variável de resposta**, **fatores**, **níveis**, **fatores primários**, **fatores secundários**, **replicação**, **unidade de experimento** e **interação**. Existem inúmeras variedades de planejamentos de experimentos. Os três modelos mais utilizados são: planejamento fatorial simples, planejamento fatorial completo e planejamento fatorial parcial. Suas vantagens e desvantagens serão explicadas a seguir (JAIN, 1991):

- **Planejamento fatorial simples:** Em um planejamento fatorial simples, começa-se com uma configuração típica e varia-se um fator de cada vez para ver como esse fator afeta o desempenho. Esse tipo de planejamento é o mais fácil de utilizar, porém não é estatisticamente eficiente.
- **Planejamento fatorial completo:** Um planejamento fatorial completo utiliza todas as combinações possíveis em todos os níveis de todos os fatores. A vantagem de um planejamento fatorial completo é que todas as combinações possíveis e cargas de trabalho são examinadas.
- **Planejamento fatorial parcial:** Algumas vezes, o número de experimentos necessários para um planejamento fatorial completo é muito grande. Isso pode acontecer se o número de fatores ou níveis é muito grande. Nesses casos, pode-se usar apenas uma fração do planejamento fatorial completo. Para cada vantagem existe uma desvantagem correspondente. Fatoriais parciais podem economizar tempo e despesas em relação ao fatorial completo.

2.5.2 *Metodologias para Avaliação de Desempenho*

Existem diversas metodologias e procedimentos distintos para executar a ADSC. Para ajudar a entender e classificar algumas dessas metodologias, elas foram categorizadas em diferentes tipos, por exemplo, análise observacional e análise experimental. A Tabela 4 exibe algumas das metodologias existentes (GREGG, 2014).

Tabela 4 – Metodologias para avaliação de desempenho.

Metodologia	Categoria
Lista de checagem padrão	Análise observacional e análise experimental.
Análise de cima para baixo	Análise observacional.
<i>Benchmarking</i>	Análise experimental.
Análise do estado de <i>threads</i>	Análise observacional
Complexidade assintótica	Análise matemática

De maneira resumida, pode-se definir essas metodologias da seguinte maneira (GREGG, 2014):

- **Lista de checagem padrão:** Verificar uma lista de checagem padrão é uma metodologia usada por analistas de suporte quando é necessário avaliar e configurar um sistema, usualmente em um curto intervalo de tempo. Um cenário típico é quando um novo servidor ou aplicação é implantada, e o analista de suporte passa algumas horas configurando o sistema de acordo com a carga de trabalho e alguns objetivos de desempenho padrão. Uma lista de checagem padrão pode ser composta de uma dúzia de comandos de verificação simples e ajustes de parâmetros configuráveis. Apesar dessas listas serem um bom ponto de partida, elas precisam ser constantemente atualizadas para não se tornarem obsoletas.
- **Análise de cima para baixo:** Esta metodologia começa com a análise do problema em um nível elevado, em seguida, estreitando o foco baseando-se em descobertas anteriores, descartando áreas que parecem desinteressantes, e aprofundando em áreas que são interessantes. O processo pode continuar até camadas mais baixas da pilha de camadas de *software*, chegando até o *hardware*, caso seja necessário, para achar causa de um determinado problema.
- **Benchmarking:** Um *benchmark* testa o desempenho de um sistema computacional de maneira controlada, permitindo que escolhas sejam comparadas e limites de desempenho sejam compreendidos – antes que eles ocorram no sistema computacional em produção. Esses limites podem ser recursos do sistema, limites de *software* em um ambiente virtualizado (CN), ou limites na aplicação alvo.
- **Análise do estado de *threads*:** A análise do estado das *threads*, tem como objetivo identificar em alto nível, aonde as *threads* da aplicação estão gastando seu tempo, o que pode solucionar alguns problemas imediatamente e também direcionar a investigação de outros.

Em sistemas empresariais, executar *benchmarks* durante as provas de conceito são considerados um importante exercício antes de investir em *hardware* caro e pode ser um processo que dura várias semanas. Isso inclui o tempo de receber os equipamentos, instalá-los em uma estrutura de rack, instalar o cabeamento, e somente então instalar os SOs e começar a testá-lo em produção (GREGG, 2014).

Tabela 5 – Complexidades assintóticas e exemplos.

Notação	Exemplos
$O(1)$	Teste booleano.
$O(\log_n)$	Busca binária em um vetor ordenado.
$O(n)$	Busca linear em uma lista ligada.
$O(n \log_n)$	Algoritmo de ordenação MergeSort.
$O(n^2)$	Algoritmo de ordenação método da bolha.
$O(2^n)$	Cálculo de números fatoriais; crescimento exponencial.
$O(n!)$	Busca exaustiva do problema do caixeiro viajante.

Em ambientes de CN, recursos de infraestrutura estão disponíveis sob demanda, sem a necessidade de um grande investimento inicial em *hardware*. Mesmo assim, deve-se realizar alguns estudos para definir qual a melhor linguagem de programação a ser adotada, com qual SGBD, qual servidor Web, qual gerenciador de carga de trabalho e assim por diante. Algumas dessas escolhas são difíceis de modificar posteriormente. Pode-se usar *benchmarks* para investigar quão bem essas escolhas atendem aos objetivos de desempenho quando a infraestrutura precisar ser aumentada. A computação em nuvem também faz com que *benchmarks* em sistemas de larga escala sejam fáceis, um ambiente de larga escala pode ser criado em minutos, executar uma série de *benchmarks* e então ser destruído, a um custo bem pequeno (GREGG, 2014).

2.5.3 Complexidade Assintótica de Algoritmos

A complexidade assintótica de algoritmos, mais especificamente a notação do **grande O**, é usualmente considerada um tópico da ciência da computação, e é usada para analisar a complexidade de algoritmos e prever como os algoritmos se comportarão, de acordo com o crescimento nos seus dados de entrada. Isso ajuda os programadores a selecionar algoritmos mais eficientes quando estão desenvolvendo suas aplicações. A Tabela 5 apresenta alguns exemplos de tarefas e suas respectivas complexidades assintóticas.

A notação permite que os programadores estimarem o aumento no desempenho (do inglês, *speedup*) de diferentes algoritmos, determinando quais áreas de código levarão aos maiores ganhos de desempenho (GREGG, 2014). A Figura 13 exibe um exemplo de comportamento de diferentes notações.

A Figura 13 mostra as linhas de tendência para as diferentes complexidades assintóticas conhecidas. No eixo horizontal, está o tamanho da entrada que o algoritmo receberá (por exemplo, se for um algoritmo de ordenação, n representará a quantidade de elementos que devem ser ordenados). No eixo vertical está o tempo de execução do algoritmo. Sendo assim, nota-se que o tempo de execução pode fazer com que determinados algoritmos, com complexidade assintótica maior, tornem-se impraticáveis, dependendo do tamanho da entrada. Isso é ainda mais crítico no caso dos sistemas de tempo real, que possuem um limite de tempo máximo para a execução de suas operações.

Figura 13 – Comportamento de algoritmos em relação ao tamanho da entrada.

2.5.4 Otimização off-line e Otimização online

Atualmente, a maioria dos *softwares* possui uma série de parâmetros que podem ser configurados de duas maneiras diferentes: **estática** e **dinâmica**. Parâmetros que devem ser configurados de maneira estática são aqueles em que o *software* não pode estar em execução quando o parâmetro é modificado. Por exemplo, suponha que um determinado parâmetro estático de um servidor Web deve ser modificado. Se o servidor Web estiver em execução, o processo deve ser encerrado, o parâmetro modificado (por meio de um arquivo de configuração, por um comando executado em linha de comando ou usando alguma ferramenta gráfica de configuração) e o processo reiniciado. Por outro lado, um parâmetro dinâmico pode ser modificado enquanto o processo (servidor Web) está em execução.

Um servidor Web mundialmente conhecido e amplamente usado é o **Apache Httpd Server**, que possui em sua documentação uma seção denominada “*Apache Performance Tuning*”. Nela, ele classifica os parâmetros relacionados ao desempenho em duas categorias: em tempo de compilação (**estática**) e em tempo de execução (**dinâmica**). A Tabela 6 mostra alguns parâmetros estáticos e uma breve descrição. A Tabela 7 mostra alguns parâmetros dinâmicos e uma breve descrição. Para maiores informações, consultar a documentação.

Nesta tese, todas as otimizações feitas enquanto o SDA não está em execução, serão chamadas de **otimização off-line**. Isso inclui a definição de parâmetros estáticos, escolha de componentes, modificação de códigos fonte e assim por diante. Diversas metodologias diferentes podem ser empregadas para a otimização *off-line*. A lista de checagem padrão, *micro-benchmarks*, otimização do código fonte usando *profiling* e assim por diante.

As otimizações feitas enquanto o SDA está em execução serão chamadas de **otimização online**. Elas incluem a modificação de parâmetros dinâmicos (por exemplo, número de *threads*), a modificação de recursos computacionais (por exemplo, adição e/ou remoção de MVs), além do emprego de laços de controle retro-alimentados, usando as mais variadas técnicas e algoritmos para isso. Os aspectos de auto-otimização e autoconfiguração da CA podem ser usados para

otimização online.

De maneira similar aos atributos citados nas Tabelas 6 e 7, outras aplicações também possuem parâmetros estáticos e dinâmicos. O MySQL server (SGBD), o Apache Tomcat (container de aplicações) e a *Java Virtual Machine* (JVM) (Máquina Virtual Java) são alguns exemplos. Além disso, caso o SDA esteja executando em MVs, também é possível modificar parâmetros estáticos (n. de vCPUs, memória RAM, etc.).

Tabela 6 – Parâmetros estáticos do Apache Httpd Server.

Parâmetro	Descrição
Módulo de multi-processamento	<p>Worker: múltiplos processos filhos com múltiplas <i>threads</i> cada um.</p> <p>Event: É similar a anterior, mas é projetada para permitir que mais requisições sejam atendidas simultaneamente, transferindo algum trabalho de processamento para as <i>threads</i> de apoio, liberando as <i>threads</i> principais para atender novas requisições.</p> <p>Prefork: Usa vários processos filho com uma <i>thread</i> cada. Cada processo lida com uma conexão de cada vez. Em alguns SOs, <i>prefork</i> é comparável em velocidade ao módulo <i>Worker</i>, mas usa mais memória principal. O fato de ter poucas <i>threads</i> facilita a depuração em sistemas que não tem um bom suporte a <i>threads</i>.</p>
Carregar módulo	De fato, a característica que mais afeta o desempenho de um servidor Web é a memória RAM. Se o servidor Web tiver que usar memória virtual, o tempo de resposta das requisições cairá para um patamar inaceitável. Sendo assim, carregar somente os módulos necessários para a sua aplicação afeta o desempenho significativamente.

Tabela 7 – Parâmetros dinâmicos do Apache Httpd Server

Parâmetro	Descrição
N. max. de servidores ociosos	Define o número máximo desejado de processos filhos ociosos. Um processo ocioso é aquele que não está atendendo nenhuma requisição. Se houver mais processos ociosos do que o valor definido, então o processo pai matará os filhos que estão sobrando.
N. min. de servidores ociosos	De maneira semelhante ao anterior, porém define a quantidade mínima de processos ociosos.
N. max. de <i>threads</i> ociosas	Define o número máximo desejado de <i>threads</i> ociosas por processo.
N. min. de <i>threads</i> ociosas	De maneira semelhante ao anterior, porém define a quantidade mínima de <i>threads</i> ociosas por processo.
N. max. de req. simultâneas	Define o limite de requisições sendo atendidas simultaneamente. Qualquer tentativa de conexão que tente ultrapassar o limite estabelecido, será colocada na fila de espera.

2.6 Considerações Finais

Uma característica importante a ser destacada, é que o PU pode ser usado para criação de sistemas de pequeno até grande porte, feitos por equipes que podem variar de uma a centenas de pessoas. Além disso, deve-se usar o bom senso para decidir quais diagramas UML, descrições textuais e demais documentos deverão ser criados em cada projeto de *software* específico. Por exemplo, quando a equipe responsável pelo fluxo de trabalho projeto é a mesma responsável pelo fluxo de trabalho implementação, não é necessário criar pseudocódigos explicando passo a passo o que cada algoritmo deve fazer.

Neste capítulo foram apresentados os principais conceitos relacionados aos SDs, bem como seus desafios e os benefícios de seu uso. Houve um foco em *Web services*, pois os estudos de casos realizados abordam problemas relacionados à seleção de *Web services* baseada em QoS. Foram mencionados os conceitos fundamentais relacionados a CA, bem como uma relação entre os aspectos da CA e problemas inerentes aos SDs. Ademais, os modelos de arquiteturas de sistemas autônômicos com suas vantagens e limitações também foram apresentados. Por fim, alguns exemplos de como os aspectos da CA podem ser usados para resolver problemas de SDs foram mencionados.

Os conceitos relativos a PO, o roteiro a ser seguido para se formalizar um problema matematicamente e o um exemplo clássico de PO, o problema da mochila foram apresentados. A ideia é que, uma vez entendida as aplicações e modelagens existentes nos problemas clássicos de PO, seja possível entender como adaptar problemas referentes a sistemas computacionais, usando PO para modela-los matematicamente e os algoritmos apropriados para soluçona-los.

Por fim, foi abordada a ADSC, mostrando algumas de suas motivações, etapas e metodologias existentes. Os conceitos de otimização *off-line* e otimização *online* também foram mencionados e são importantes para o entendimento do PDS2DA e dos estudos de caso desenvolvidos.

TRABALHOS RELACIONADOS

Este capítulo visa apresentar alguns trabalhos relacionados, permitindo um melhor entendimento do tema estudado e de como se dá o relacionamento entre as diferentes áreas de estudo desta tese: processo unificado, sistemas distribuídos, computação autônômica, pesquisa operacional e avaliação de desempenho de sistemas computacionais. De fato, após analisar os trabalhos relacionados, ficará mais claro a ligação entre essas áreas e algumas motivações em uni-las em um **Processo de Desenvolvimento de *Software* focado em Sistemas Distribuídos Autônômicos**.

3.1 Sistemas Distribuídos e Computação Autônômica

Uma parte significativa dos trabalhos focados em CA tratam de SDs. Isso serve para comprovar que a combinação de SDs e CA faz sentido e permite aos SDs atenderem seus principais requisitos.

Na CN, a CA vem sendo aplicada com sucesso. No trabalho de ([ARDAGNA et al., 2012](#)), é proposto um *framework* para a alocação de recursos de forma autônômica em ambientes de aplicações multicamadas virtualizado, visando economia de energia.

Em ([PALMA et al., 2012](#)), o alvo são aplicações distribuídas em um grupo de computadores. Por meio do projeto, implementação, condução de experimentos e ADSC, um protótipo contendo um mecanismo com a capacidade de autoproteção foi desenvolvido.

Na área de WSs, o trabalho de ([NALLUR; BAHSOON, 2013](#)) propõe um mecanismo descentralizado de auto-adaptação, para WSs que estão implantados em ambientes de CN. Eles abordam a CWSbQ usando os seguintes atributos de QoS: latência, disponibilidade e preço.

A CA também já foi usada em SGBDs. Em ([ALOMARI; MENASCE, 2013](#)) é proposto um controlador autônômico que permite que o SGBD tenha as capacidades de autoproteção e auto-otimização. Por meio da implementação de um protótipo e da condução de experimentos e

ADSC, este trabalho mostra que incluindo o controlador autônomo e o módulo de monitoramento de desempenho, o SGBD conseguiu um maior grau de autonomia.

No trabalho de (CHEN *et al.*, 2015), por meio de um laço de controle e modelos orientados a objetivos (*Goal models*) foi proposta uma abordagem que permite a auto-otimização da composição de WSs, baseando-se nos requisitos de negócios.

3.2 Sistemas Distribuídos e Pesquisa Operacional

Os SDs e a PO vem sendo estudados de maneira conjunta em diversas aplicações diferentes. Nesse ponto, deve-se entender a PO como uma ferramenta para solucionar problemas que são inerentes aos SDs. Da mesma maneira que a CA atende alguns requisitos dos SDs, a PO também o faz.

O trabalho de (MENASCÉ; BARBARÁ; DODGE, 2001) modelou um site de *e-commerce* composto de: servidores web, servidores de aplicação e SGBDs, usando um modelo de redes de filas. Para a solução do modelo, usou um algoritmo chamado subida de encosta (*hill climbing*), que é um algoritmo da área de programação matemática, mais especificamente, um algoritmo de busca local.

Em (CANFORA *et al.*, 2005), o problema de CWSbQ foi definido com um problema de programação linear inteira (otimização linear inteira). Foi feita uma comparação entre um algoritmo de PLI e um AG.

No trabalho de (CHECIU *et al.*, 2011), foi criado um modelo matemático não-linear, usando teoria de filas e a técnica conhecida como filtro de Kalman, possibilitando a criação de composições de WSs de forma autônoma.

O problema de CWSbQ também foi investigado no trabalho de (PRADO *et al.*, 2013). Ele foi modelado como um problema de **otimização combinatória sem restrições** e resolvido por um algoritmo de busca exaustiva e algumas heurísticas.

Recentemente, no trabalho de (EWING; MENASCE, 2014), foi proposto um meta-controlador autônomo, para uma arquitetura AOS, denominada SASSY (*A Framework for Self-Architecting Service-Oriented Systems*), esse meta-controlador resolve dois problemas de complexidade **NP-Difícil**. Primeiro, decidir qual é a melhor arquitetura (quais serão os serviços abstratos que participarão do serviço composto, qual o fluxo de dados entre os serviços, qual a ordem de execução dos serviços) e em seguida, quais WSs concretos participarão da composição (para cada WS abstrato, podem haver dezenas de WSs concretos candidatos). Para solucionar esses problemas, foram criadas duas formulações matemáticas para o problema, uma definindo-o como um problema de otimização combinatória sem restrições e outra definindo-o como um problema de otimização combinatória tendo o custo dos WSs selecionados como uma restrição. Em ambos os casos, uma das soluções propostas é o algoritmo subida de encosta, que como já

foi citado, é um algoritmo de programação matemática.

Na dissertação de mestrado de (BORRO, 2014), foi abordado o problema de gerenciamento energético em grades móveis, garantindo o cumprimento de requisitos de QoS das aplicações dos usuários. O problema foi modelado como um problema de PLI, sendo que dois algoritmos foram implementados e comparados usando ADSC.

3.3 Sistemas Distribuídos e Avaliação de Desempenho

As técnicas de ADSC tem sido usadas em diversos trabalhos relacionados a SDs, comparando quantitativamente diferentes soluções propostas. Como foi citado no capítulo de ADSC, existem três técnicas diferentes de avaliação de desempenho, são elas: modelagem com solução analítica, modelagem com solução por simulação e aferição. No caso dos sistemas computacionais, o que se tem observado é que cria-se um modelo matemático para o sistema e costuma-se resolvê-lo por meio de simulação ou aferição. A aferição pode tanto ser no sistema real ou em um protótipo do sistema real.

Um sistema adaptativo e inteligente para a negociação automática de SLAs foi proposto em (ZULKERNINE; MARTIN, 2011). Por meio da implementação de um protótipo e um conjunto extensivo de experimentos, essa abordagem foi validada e teve seus resultados avaliados quantitativamente, por meio da ADSC.

A CWSbQ tem sido alvo da avaliação de desempenho em diversos trabalhos. Em (PRADO *et al.*, 2012a), foi criado um protótipo de um seletor de WSs. Esse protótipo era composto de: um SGBD contendo os atributos de QoS dos WSs candidatos, um servidor de aplicações que continha o WS seletor e uma aplicação cliente, que enviava requisições para o WS seletor. Por meio da ADSC, foi possível comparar diferentes algoritmos para a solução desse problema (busca exaustiva, AGs, heurísticas), tanto em relação ao tempo de execução dos algoritmos, como a qualidade (QoS total da composição) das soluções fornecidas. O mesmo foi feito em (PRADO *et al.*, 2012b), (PRADO *et al.*, 2013), (PRADO, 2012), (PRADO; SANTANA; SANTANA, 2015) e (SCHIMIDT; PRADO; NEVES, 2014) porém com outros algoritmos e/ou parâmetros ou diferentes contextos.

No trabalho de (CHEN *et al.*, 2012), é proposto um mecanismo de autoconfiguração de MVs em um ambiente de CN. Para validar o mecanismo, foi criado um protótipo de um ambiente de nuvem (uma versão reduzida, rodando em um grupo de computadores com algumas máquinas virtuais) e foram definidas três cargas de trabalho diferentes: processamento intensivo, uso de memória principal intensivo e uso de rede intensivo. Por meio de aferições nesse protótipo, foi possível comparar diferentes técnicas de migração de MVs.

3.4 Algoritmos e Técnicas para Implementação dos Aspectos da Computação Autônômica

No artigo de (KHALID *et al.*, 2009), são citadas diversas técnicas e algoritmos para implementação dos aspectos da CA. Cada subseção apresenta os algoritmos e técnicas relacionados a um dos aspectos da CA.

3.4.1 Algoritmos e Técnicas para Autoconfiguração

Os algoritmos e técnicas mencionados são: **troca a quente** (*Hot Swapping*), **Aprendizado de Máquina** (AM), **Raciocínio Baseado em Casos** (RBC) e **agrupamento de dados** (*Data Clustering*). Outra técnica conhecida é denominada **Políticas Estáticas** (PEs).

A **troca a quente** é uma técnica para inserir código de monitoramento e diagnóstico em código fonte em execução usando inter-posicionamento e substituição. Inter-posicionamento é a inserção de um novo componente e substituição é a troca de um componente ativo por um com a mesma funcionalidade, porém implementação diferente. O sistema adapta-se perfeitamente a essas mudanças nos componentes. A troca a quente suporta muitas características da CA, tais como: gestão ótima de recursos computacionais, monitoramento do sistema e configuração dinâmica e extensível.

As **técnicas de AM** tem sido usadas para prover capacidades de autoconfiguração em sistemas de armazenamento de arquivos. Foram usadas árvores de decisão para a classificação automática de arquivos, permitindo que tais sistemas de arquivos e armazenamento possam fazer configurações corretas, resultando em um provisionamento eficiente e alocação de recursos necessários.

A técnica de **RBC** é usada para permitir a autoconfiguração em sistemas autônômicos. A autoconfiguração baseada em RBC ocorre durante a fase de Planejamento do ciclo de vida do gerenciador autônômico, recomendando uma nova configuração, baseada em casos anteriores. A configuração recomendada, é então aplicada ao sistema autônômico monitorado.

O **agrupamento de dados**, é um algoritmo de **aprendizado não-supervisionado**. Ele é usado para identificar classes de configuração e determina o grau de similaridade entre essas classes usando uma métrica de média convexa. Os dados são agrupados de acordo com suas similaridades estruturais, portanto, infere boas definições de registro. Depois de encontrar classes de configurações, restrições de correção são estabelecidas.

Por fim, as **PEs** podem ser vistas como valores limite pré-determinados, sempre possuindo um valor para alguma variável de controle e uma ação a ser tomada, caso aquele valor seja atingido. Por exemplo, pode-se determinar que quando a bateria de um notebook chegar a 10%, deve-se entrar no modo de economia de energia. Ou ainda, quando a taxa de utilização de uma MV chegar a 90%, deve-se inicializar uma cópia dessa MV e fazer o devido escalonamento

das requisições e o balanceamento da carga de trabalho.

3.4.2 Algoritmos e Técnicas para Auto-otimização

Os algoritmos e técnicas mencionados são: **teoria de controle (control theory)** e **aprendizado ativo (active learning)**. Além dessas, também são técnicas conhecidas: **Busca Local (BL)**, **redes de filas** e AGs.

A **teoria de controle** tem sido usada para prover auto-otimização em sistemas autônomos. A teoria de controle define a tarefa de auto-gerenciamento como um laço de controle fechado que monitora e controla os recursos computacionais. O uso de um laço de controle linear foi usado para prover auto-otimização no trabalho de (GHANDI *et al.*, 2001). No *Lotus Notes*, o desempenho é avaliado pelo monitoramento do número máximo de usuários no servidor. O *feedback* é fornecido para um controlador externo que usa modelos estatísticos e valores anteriores para gerar uma saída para o controle. Essa forma de auto-otimização é avaliada usando teoria de controle e avaliação empírica no sistema gerenciado.

No trabalho de (SHIVAM; BABU; CHASE, 2006) foi usada uma abordagem de **aprendizado ativo** para prover auto-otimização ao sistema autônomo gerenciado. Esta abordagem baseia-se na construção de modelos de previsão estatística com base no histórico anterior de utilizadores de computação.

De acordo com (EWING; MENASCE, 2014), os algoritmos de BL (também conhecidos como busca direta, na comunidade de PO) iniciam com uma ou mais soluções (chamadas de soluções visitadas) e, em seguida, avaliam soluções semelhantes, chamadas soluções vizinhas. Em um esforço para encontrar soluções melhores, o algoritmo visita um ou mais vizinhos que foram bem avaliados e gera novas soluções vizinhas, gerando novos vizinhos que poderão ser visitados. Esse procedimento continua até que um ótimo local seja encontrado ou algum critério de parada seja atingido. A maioria dos algoritmos de busca local, após encontrar um ótimo local, pode reiniciar a busca por meio de novos pontos aleatoriamente dispersos, visando encontrar uma solução melhor.

Segundo (AWAD; MENASCE, 2014), as **redes de filas** são uma abordagem de modelagem em que os sistemas computacionais são modelados como uma rede de filas. Cada fila representa clientes (por exemplo, usuários finais ou requisições de processamento de dados) e dispositivos (por exemplo, CPU, disco). As redes de filas possuem dois tipos de parâmetros: (i) intensidade da carga de trabalho, que pode ser mensurada pelo número de requisições no sistema ou pela média da taxa de chegada das requisições; e (ii) as demandas dos serviços, que são o tempo médio total necessário para atender uma requisição de uma classe particular para todos os dispositivos do sistema. O tempo de espera na fila não é incluso nesse tempo de demanda do serviço. Redes de Filas complexas representam várias filas, sendo que um cliente pode entrar em outra fila após ter sido atendido em uma fila anterior, dentro da rede de filas.

Os AGs já foram explicados anteriormente e tem sido usados para prover auto-otimização em alguns trabalhos relacionados. Um exemplo é o artigo de (JIANG; LU; ZHANG, 2011), que usa um AG para prover autoconfiguração e auto-otimização em um ambiente de computação em nuvem.

3.4.3 Algoritmos e Técnicas para Autocura

Os algoritmos e técnicas mencionadas são: **RBC**, **sondagem ativa e redes bayesianas** (*Active Probing and Bayesian Networks*) e **árvores de decisão**. Também são técnicas conhecidas: **políticas estáticas** e **detecção adaptativa de anomalias**.

A auto-cura é o aspecto da CA que envolve a **detecção de problemas, diagnóstico e reparo**. A RBC executa todas essas atividades, sendo que a detecção de problemas é feita durante a fase de Análise, a solução para o problema é encontrada na fase de Planejamento e o reparo do problema é feito na fase de Execução. No trabalho de (ANGLANO; MONTANI, 2007) é demonstrada como a técnica de CBR é usada para prover auto-cura.

A **sondagem ativa** é uma técnica de diagnóstico de problemas que permite que sondas sejam selecionadas e enviadas sob demanda. Quando os resultados das sondas são recebidos, o estado atual do sistema é atualizando usando inferência probabilística. Esse processo continua até que o problema seja detectado. Ele usa uma matriz de dependência e rede bayesiana para representar as sondas. Ambas as formas de sondagem, periódicas e ativas são usadas no trabalho de (RISH *et al.*, 2004) para detecção e diagnóstico de problemas.

O uso de **árvores de decisão** tem sido aplicado para obter auto-cura, quando o intuito é fazer a reinicialização de componentes ou "*rollback*" em programas, visando reparar algum problema previamente detectado.

No trabalho de (FELLER; RILLING; MORIN, 2012), são usadas políticas estáticas para prover auto-cura em um sistema gerenciador de MVs. É feito um monitoramento dos gerenciadores autônomicos locais e quando algum falha, essa falha é detectada e um novo gerenciador é colocado para substituí-lo.

Uma técnica conhecida como **detecção adaptativa de anomalias** foi usada no trabalho de (PANNU; LIU; FU, 2012). Realizando um monitoramento das requisições atendidas por um ambiente de computação em nuvem, o algoritmo tenta prever se determinado comportamento representa uma falha. Caso essa falha seja confirmada por um operador do ambiente em nuvem, o algoritmo aprende que sua previsão estava correta. Além disso, falhas que não foram detectadas pelo algoritmo, mas foram reportadas pelos operadores do ambiente em nuvem, são aprendidas pelo algoritmo, para quem possam ser detectadas posteriormente, caso voltem a ocorrer.

3.4.4 Algoritmos e Técnicas para Autoproteção

Os algoritmos e técnicas mencionadas são: **modelo de componentes** e **alertas de auto-certificação**. Também são técnicas conhecidas: **políticas estáticas**, algoritmos de **função-utilidade**, **regras semânticas** e **técnicas baseadas em validação de modelos**.

Os **modelos de componentes** podem ser usados para permitir a autoproteção em um recurso monitorado. No trabalho de (CLAUDEL *et al.*, 2006), foi desenvolvido o sistema Jade que apoia a construção de programas autônomos usando o modelo de componentes. Ele implementa autoproteção identificando operações ilegais e isolando o componente comprometido do sistema, visando evitar maiores danos.

Os **alertas de auto-certificação** podem ser usados para evitar a propagação de vermes (*worms*) por meio de uma rede, como foi demonstrado no artigo de (COSTA *et al.*, 2005), com a criação do antivírus chamado Vigilante. O Vigilante é um antivírus baseado na colaboração entre seus hospedeiros, sendo que quando um *worm* é detectado, sua assinatura é transmitida para os demais hospedeiros da rede, para que estes possam criar um filtro que bloqueia automaticamente o *worm* detectado.

Um conjunto de **políticas estáticas** foi usado no trabalho de (PALMA *et al.*, 2012). A arquitetura do sistema monitorado, bem como as comunicações permitidas entre os elementos são armazenadas em uma base de conhecimento. Por meio de um monitoramento constante, qualquer tentativa de comunicação que não esteja expressamente permitida na base de conhecimento, ativa uma política de isolamento, fazendo com que o usuário que está tentando realizar essa ação seja proibido de fazer qualquer tentativa de comunicação futura.

Um algoritmo de **função-utilidade multi-objetivo** foi usado no trabalho de (ALOMARI; MENASCE, 2013), para garantir auto-otimização e autoproteção em um sistema composto de várias instâncias de SGBDs. O algoritmo visa manter um equilíbrio entre o nível de segurança do sistema e o QoS fornecido pelo mesmo.

As **regras semânticas** são outra forma de implementar autoproteção em um sistema autônomo. No trabalho de (CHAARI; FAKHFAKH, 2012) é proposto um *framework* que auxilia na definição de regras semânticas que podem ser usadas para implementar qualquer um dos aspectos da CA, incluindo autoproteção.

Em (CHEN; ABDELWAHED; ERRADI, 2014) é apresentada uma técnica baseada em **validação de modelos**, visando prover a capacidade de autoproteção para ambientes de internet das coisas. Ela é composta de três partes, são elas: estimacão em tempo real e uso de controles de segurança de linha de base, visando prever e eliminar possíveis ataques; análise de dados para prever e classificar ataques; e o uso de um algoritmo de otimização multi-objetivo, visando a escolha da melhor solução possível, quando uma ameaça for detectada.

Tabela 8 – Resumo das técnicas usadas por aspecto da computação autônoma

Aspecto	Técnicas usadas
autoconfiguração	Troca a quente, aprendizado de máquina, raciocínio baseado em casos e agrupamento de dados.
auto-otimização	Teoria de controle, aprendizado ativo, busca local, redes de filas e AGs.
autocura	Raciocínio baseado em casos, sondagem ativa e redes <i>bayesianas</i> , árvores de decisão, políticas estáticas e detecção adaptativa de anomalias.
autoproteção	Modelo de componentes, alertas de auto-certificação, políticas estáticas, algoritmos de função-utilidade, regras semânticas e técnicas baseadas em validação de modelos.

3.4.5 Resumo dos Algoritmos e Técnicas

É interessante notar, que alguns algoritmos e técnicas têm sido usados para implementar mais de um aspecto diferente da CA. Por exemplo, o RBC foi usado tanto para implementar o aspecto de autoconfiguração, quanto o aspecto de autocura. No momento de definição de qual aspecto da CA será implementado, deve-se estudar os trabalhos relacionados, identificando quais algoritmos foram usados e quais seus benefícios e limitações. Dessa forma, é possível ter uma ideia de qual a melhor solução para o SDA, antes de começar implementá-lo. A Tabela 8 mostra um resumo dos algoritmos e técnicas citados nesta seção.

3.5 Categorização dos Trabalhos Relacionados a Computação Autônoma

A ideia desta seção é determinar o que um grupo de trabalhos relacionados possui em comum, permitindo entender quais contribuições já foram feitas nas diferentes áreas da computação autônoma, bem como possíveis melhorias que podem ser feitas. Dessa forma, quatro aspectos foram levados em consideração, quando a revisão da literatura foi realizada.

O primeiro aspecto é o **Foco** do trabalho sendo estudado. O foco pode ser definido como em qual área o trabalho se encontra, podendo ser, por exemplo: Ws, CN, redes de sensores, SDs em geral, etc.

O segundo aspecto é a **Contribuição** do trabalho. Nesse caso, a contribuição do trabalho é resumida em uma frase, por exemplo: “*framework* visando economia de energia”.

O terceiro aspecto é a **Validação** do trabalho. Esse aspecto especifica como o que foi proposto no trabalho foi validado, são eles: “**Simulação (S)**”, “**Aferição em sistema real (A)**”, “**Protótipo (P)**”, “**Protótipo e avaliação de desempenho (P&A)**” e “**Não executou (NE)**”.

O quarto aspecto é a **Categoria** em que o trabalho se encontra, por exemplo: “*framework*,

middleware ou ferramenta de programação (FMP)”, “aplicação de computação autônoma em um sistema específico (A)”, “modelo de arquitetura de sistemas autônomos (M)”, “revisão da literatura (R)” ou “não diretamente relacionado a computação autônoma (NDR)”.

É interessante frisar que não foi encontrado nada parecido dentre os trabalhos relacionados estudados, sendo assim, acredita-se que seja um ponto de partida importante e interessante para uma revisão da literatura mais aprofundada da área.

3.5.1 Frameworks, Middlewares e Ferramentas de Programação

Um *framework de software* pode ser definido como uma abstração, em que é fornecido um conjunto genérico de funcionalidades (operações, métodos, etc.) que pode ser selecionado e modificado por um usuário, visando adaptar-se as necessidades de uma aplicação específica.

Um *middleware* é um *software* que provê serviços para aplicações além daqueles fornecidos pelo sistema operacional. É um termo geralmente usado para aplicações que permitem a comunicação e gerenciamento de dados em SDs.

Uma **ferramenta de programação** (do inglês, *software development tool* ou *programming tool*) é um programa de computador que os desenvolvedores de *software* usam para criar, depurar, manter ou de outra maneira, dar suporte, a outros programas e aplicações.

Uma série de trabalhos relacionados a computação autônoma tem visado o desenvolvimento de *frameworks* ou *middlewares* para facilitar a criação e/ou manutenção de sistemas autônomos.

Um *framework* visando facilitar o gerenciamento de diferentes laços de controle foi proposto no artigo de (AL-SHISHTAWY *et al.*, 2008). O *framework* foi denominado Sistema Gerenciador de Componentes Distribuídos (ou em inglês, *Distributed Component Managment System* – DCMS) e deve ser usado em aplicações implantadas em ambientes de grades computacionais. Ele é um *framework* baseado em eventos que provê capacidades de sensoriamento e atuação em ambientes distribuídos, separando os laços de acordo com o aspecto da computação autônoma em questão. Por exemplo, para um sistema com a capacidade de autocura e autoproteção, deve-se implementar dois laços de controle separados.

No trabalho de (BHAKTI; ABDULLAH; JUNG, 2010), foi proposto um *framework* para a integração entre computação autônoma e OAS. O GA foi implementado usando o laço de controle MAPE-C, sendo que no módulo Planejador, foi usada a técnica conhecida como RBC. Por meio de um protótipo e um estudo de caso, o *framework* foi validado.

Os Sistemas Baseados em SOA (SBS) foram alvo do *framework* criado no trabalho de (CALINESCU *et al.*, 2011). Para lidar com o gerenciamento de atributos de QoS dos SBS, foi desenvolvido um *framework* denominado QoS MOS (*QoS Management and Optimization of*

Service-based Systems). Neste artigo, os autores fazem uma interessante comparação com os trabalhos relacionados até então. É fornecido, para cada trabalho relacionado citado, os seguintes aspectos: domínio do problema, atributos de QoS considerados, forma de avaliação dos atributos de QoS, método de otimização ou adaptação do QoS global e como foi feita a validação. No caso do QoSMOS, o domínio do problema é: seleção de WSs, alocação de recursos computacionais para os WSs e parametrização dos WSs. Os atributos de QoS considerados são: desempenho e confiabilidade. A forma de avaliar o QoS é por meio da solução analítica de modelos de Markov. A técnica de otimização ou adaptação é uma solução exata obtida por um modelo probabilístico. A forma de validação foi a execução de experimentos e simulações, baseadas em exemplos genéricos e por meio de um estudo de caso. A arquitetura proposta pelo QoSMOS segue os módulos propostos no laço de controle MAPE-C.

Os ambientes de nuvem privadas foram alvo do *framework* proposto no artigo de (FELLER; RILLING; MORIN, 2012), denominado Snooze (*A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds*). Por meio do projeto, implementação e execução de experimentos, com consequente avaliação de desempenho, foi demonstrado que o *framework* possui as capacidades de ser autoconfigurável e autocurável. Foi usado um grupo de computadores, com uma estrutura de laços de controle hierárquica (cada Gerente de Grupo gerenciava uma quantidade de MVs e todos os gerentes de grupo eram gerenciados pelo Líder de grupo) e uma série de aplicações de *benchmark*, visando avaliar o tempo de execução das cargas de trabalho fornecidas. Foram criadas dois tipos de MVs diferentes, uma contendo um *benchmark* que representa uma aplicação *cpu-bound*, o *NAS Parallel Benchmark* (NPB) e uma outra MV contendo as aplicações: Linux, Apache Web server, MySQL, PHP LAMP, que representa uma aplicação distribuída tradicional.

3.5.2 Aplicação da Computação Autônômica em Sistema Específico

Um protótipo foi implementado no trabalho de (PANNU; LIU; FU, 2012), que criou um mecanismo com capacidade de autocura para ambientes de CN. Foram executados experimentos em um ambiente de CN dentro do campus universitário, composto de 362 nós, comparando o desempenho de diferentes algoritmos para a detecção e recuperação de falhas.

O trabalho de (LASIERRA *et al.*, 2012), criou uma ontologia para o gerenciamento doméstico de pacientes, denominada HOTMES (*Home Ontology for inTegrated Managment in homE-based Scenarios*). Por meio do uso de um laço de controle MAPE-K, um protótipo em Java e dois estudos de caso, foi comprovada a eficiência da ontologia proposta.

Um sistema bastante conhecido no gerenciamento de MVs em ambientes de CN é o *Eucalyptus Cloud Platform*. No artigo de (SONG; LI; LIU, 2012), foi proposto um sistema denominado *IdleCached*, que executa em uma camada superior e integrado ao Eucalyptus. Por meio do sistema desenvolvido, denominado *TorqueCloud*, foi possível fazer a migração de tarefas entre as MVs e lidar com a escalabilidade de maneira autônômica. Dessa forma, usando

um algoritmo de escalonamento, juntamente com o *TorqueCloud* e o *Eucalyptus*, foi possível avaliar, por meio de experimentos, um acréscimo no atendimento as demandas dos usuários, uma minimização nos custos dos serviços e também no consumo de energia.

Um sistema para gerenciar autonomicamente a evolução de WSs compostos para longo prazo, chamada Ev-LCS (*Evolution of Long-term Composed Services*) foi proposto no trabalho de (LIU *et al.*, 2013). Por meio de um modelo formal, visando mostrar a semântica do sistema e posteriormente por um protótipo e pela execução de uma série de experimentos, essa abordagem foi validada.

3.5.3 Modelos de Arquitetura de Sistemas Autônomicos

No trabalho de (JIANG; LU; ZHANG, 2011), foi proposta uma arquitetura com capacidades de auto-otimização e autoconfiguração, para ambientes de CN. Para resolver o problema de definir aonde cada WS deve ser implantado, dentro do ambiente de CN, foi criado um modelo matemático, que define o problema como sendo um problema de otimização combinatória. O módulo gerenciador autônomico é composto dos seguintes submódulos: Analisador e Previsor de carga de trabalho; Módulo de Mensuração de QoS; Módulo Otimizador e Escalonador; e Módulo de configuração de recursos.

A CN foi alvo do trabalho de (BUYYA; CALHEIROS; LI, 2012). O artigo cita as limitações atuais relacionadas ao gerenciamento de ambientes de CN, bem como de que maneira a CA pode ser integrada nesses ambientes e os possíveis benefícios que ela trará. Por meio de uma arquitetura que integra a CA em um ambiente de CN, com o uso de uma aplicação de *benchmark* e alguns experimentos, foi possível avaliar o desempenho dessa abordagem.

O monitoramento de SLAs foi o foco do trabalho de (BERTOLINO; CALABRO; ANGELIS, 2013). Por meio de uma arquitetura de monitoramento autônomico, capaz de sintetizar regras de SLA “*on-the-fly*”, é possível acompanhar a evolução do ambiente de CN que está sendo monitorado. Para provar sua viabilidade e aplicabilidade, um estudo de caso foi definido, mostrando seus benefícios, limitações e possíveis trabalhos futuros.

3.6 Análise Crítica dos Trabalhos Relacionados

A ideia dessa seção é fornecer uma visão geral, porém mais contextualizada e detalhada dos trabalhos relacionados. De fato, nas referências consultadas que estão categorizadas como revisão da literatura, apresenta-se uma visão limitada dos trabalhos relacionados.

Nos artigos de revisão da literatura, é comum duas abordagens: (i) algoritmos e técnicas usados para implementar algum aspecto específico da CA e (ii) revisão de conceitos, apresentação de modelos teóricos e/ou já implementados de arquiteturas de SDAs.

Sendo assim, serão apresentadas as quantidades de trabalhos relacionados por ano, por

foco, pela forma de validação da proposta presente no artigo, pela categoria de trabalho e quais aspectos da CA ele aborda. Isso também permite identificar uma série de lacunas nos trabalhos relacionados, por exemplo: (i) falta de validação da proposta; (ii) limitação em termos de linguagem de programação e/ou modelo de CD usada; (iii) falta de integração com as demais fases de desenvolvimento do SDA; (iv) falta de uma avaliação quantitativa dos algoritmos propostos e assim por diante.

Levando-se em conta os trabalhos citados até aqui, acredita-se que nenhum possui o objetivo claro de ser um processo de desenvolvimento de *software* focado em SDAs.

Uma proposta inicial aparece no artigo denominado “*CObAPAS: Combinatorial Optimization based Approach for Autonomic Systems*”, de (PRADO *et al.*, 2013). Neste artigo, foi proposto que é possível transformar o problema que o gerenciador autônomo deve solucionar em um problema de otimização combinatória. Foi usado um seletor de WSs, baseado em atributos de QoS, como estudo de caso. Apesar de ser um trabalho em estágio inicial, com muitas lacunas que devem ser preenchidas e poucas respostas fornecidas, trata-se de um esforço necessário e um passo inicial para a metodologia proposta nesta tese.

No trabalho de (SOLOMON *et al.*, 2010), também é proposta uma metodologia, focada no desenvolvimento de SDAs para ambientes de CN. Ela fornece uma sequência de passos que deve ser cumprida, porém não implementa o SDA proposto, muito menos faz algum tipo de avaliação de desempenho para comprovar a viabilidade da metodologia. Por fim, os passos fornecidos são explicados em alto nível, sendo necessário um grande conhecimento da área de CA para que seja possível transformar tais passos em um SDA real.

O desenvolvimento de SAs envolvem uma série de problemas, como foi citado no trabalho de (AHUJA; DANGEY, 2014). O primeiro é como determinar e especificar qual o problema em questão. Pelo fato da CA ser uma área relativamente recente, fica difícil definir o problema, por exemplo, quais partes do sistema devem ser gerenciadas de maneira autônoma. A definição do problema inclui quais dados devem ser coletados e também como deverão ser usados para gerenciar o sistema de maneira autônoma. O segundo problema é como projetar o sistema autônomo. O problema de auto-gerenciamento pode ser resolvido usando-se técnicas matemáticas (otimização combinatória, otimização linear, sistemas de fila e otimização) ou ainda técnicas baseadas em IA (AGs, RNAs, etc.). Qual técnica é a melhor opção para determinado sistema autônomo pode ser uma pergunta difícil de se responder. O terceiro problema é quais tecnologias serão usadas na implementação do sistema. Por fim, como testar o sistema e realizar a manutenção também é algo que deve ser considerado, quando se desenvolve um novo sistema autônomo ou transforma-se um sistema tradicional em um sistema autônomo.

Observando-se a Tabela 9, pode-se notar que 140 trabalhos foram citados nesta tese, porém, um total de 220 trabalhos foram consultados no decorrer deste projeto de doutorado. Uma observação interessante é que 80 trabalhos foram publicados entre 2011 e 2016, o que representa cerca de 57% das referências usadas. É sempre interessante acompanhar as últimas tendências

e contribuições fornecidas pela academia, pois algumas técnicas ou tendências que eram tidas como promissoras a alguns anos atrás, podem ter se provado ineficientes ou ultrapassadas. As referências mais antigas, são em sua maioria de livros, que concentram alguns conceitos básicos que não são influenciados com o passar dos anos (por exemplo, o que é um sistema distribuído, o que é pesquisa operacional, etc.).

Tabela 9 – Qtd. de referências, por ano.

Ano	Quantidade
1991	1
2000	1
2001	5
2003	7
2004	3
2005	8
2006	5
2007	7
2008	4
2009	12
2010	7
2011	12
2012	22
2013	15
2014	23
2015	8
Total	140

Tabela 10 – Qtd de referências, por foco.

Foco	Quantidade
Computação Autônômica	15
Computação em <i>Cluster</i>	1
Computação em Grade	3
Computação em Nuvem	32
CN + WS	2
Internet das Coisas	1
Pesquisa Operacional	2
Qualidade de Serviço	1
Redes	2
Sensores	1
SGBDs	2
SLA	1
SOA	7
Sistemas de Tempo-Real	1
VANETs	2
Web Sites	1
Web Services	41

A Tabela 10 apresenta os trabalhos citados nesta tese, organizados pelo seu Foco. As três áreas que tiveram mais foco, foram, nessa ordem: WSs, CN e CA. Vale ressaltar que WSs foi bastante citado, não apenas pelo uso da CA, mas também por ser uma das áreas não diretamente relacionadas, com trabalhos que forneceram o embasamento teórico para a criação de dois dos três estudos de caso presentes nesta tese.

A forma de validação usada pelos trabalhos é apresentada na Tabela 11. Nota-se uma dominância pela Prototipagem combinada com avaliação de desempenho. Isso é interessante por dois motivos: primeiro, a prototipagem serve como prova de conceito para diferentes categorias de trabalhos, são eles: aplicações da CA em sistemas específicos, *frameworks*, *middlewares* e ferramentas de *software*; e modelos de arquitetura. Além disso, a avaliação de desempenho permite uma análise quantitativa da contribuição fornecida pelo trabalho relacionado, ao invés de uma análise meramente qualitativa. Alguns trabalhos não executaram nenhuma validação, isso é aceitável em artigos de revisão da literatura, porém quando se propõe alguma contribuição (seja um novo algoritmo, um novo modelo de arquitetura, etc.) é interessante (ou até mesmo necessário) que uma forma de validação seja fornecida, visando demonstrar a viabilidade e a qualidade fornecida pela contribuição.

As diferentes categorias de trabalho são exibidas na Tabela 12. Nota-se um equilíbrio entre a quantidade de trabalhos que são aplicações da CA em sistemas específicos e trabalhos que são da categoria de *frameworks*, *middlewares* e ferramentas de *software*. A quantidade de trabalhos não diretamente relacionados também é significativa, tendo em vista que algumas referências não tem relação com a CA, porém são relevantes para esta tese de doutorado (por exemplo, avaliação de desempenho, pesquisa operacional, etc.). Foram listados dezoito trabalhos categorizados como revisão da literatura, posteriormente serão comentados suas qualidades e defeitos.

Tabela 11 – Qtd. de referências, por validação.

Validação	Quantidade
Aferição	7
Modelo matemático	2
Não executou	34
Protótipo	20
Protótipo e ADSC	56
P e A / Aferição	1
Simulação	14
Simulação / P e ADSC	2

Tabela 12 – Qtd de referências, por categoria.

Categoria	Quantidade
Aplicação da CA	35
<i>Framework</i> ou <i>Middleware</i>	35
Modelo de arquitetura	15
Não diretamente relacionado	35
Revisão da literatura	18

Os aspectos da CA abordados em cada trabalho são mostrados na Tabela 13. O aspecto mais citado é **Nenhum Aspecto Implementado**. Isso pode parecer estranho, porém é devido a dois fatos: os trabalhos que não são diretamente relacionados com a CA e os trabalhos que foram classificados como revisão da literatura (pois citam trabalhos relacionados a CA, porém não implementam nenhum aspecto da CA como contribuição). Sendo assim, os aspectos mais abordados são, respectivamente: autoconfiguração combinada com auto-otimização (CO & O) com 26 trabalhos citados e autoconfiguração (CO) com 15 trabalhos citados. Apenas 5 trabalhos citados foram considerados genéricos, ou seja, fornecem contribuições que podem ser aplicadas a qualquer aspecto da CA. De acordo com as categorias de trabalhos mencionadas, há pouquíssimos trabalhos que focam na criação de alguma metodologia para o desenvolvimento de sistemas computacionais autônomicos (apenas um trabalho, dentre as 220 referências consultadas). Isso faz falta, pois desenvolver sistemas autônomicos é uma tarefa que está longe de ser trivial.

3.7 Considerações finais

Este capítulo, apresentou como as diferentes áreas do conhecimento do PDS2DA têm interagido nos trabalhos relacionados. Além disso, apresentou diferentes algoritmos e técnicas usadas para implementas os aspectos da CA. Por fim, foi feita uma categorização dos trabalhos relacionados, bem como uma análise crítica. Isso é importante, pois é algo que não foi encontrado nos trabalhos que fizeram uma revisão da literatura.

Tabela 13 – Quantidade de referências, por aspecto da CA

Aspecto(s)	Quantidade
Autocura	7
Autocura e autoproteção	1
Autoconfiguração	15
Autoconfiguração e autocura	3
Autoconfiguração e auto-otimização	26
Autoconfiguração, auto-otimização e autocura	1
Genérico	5
Nenhum aspecto implementado	55
Auto-otimização	18
Autoproteção e auto-otimização	2

PROCESSO DE DESENVOLVIMENTO DE *SOFTWARE* FOCADO EM SDAS

4.1 Considerações iniciais

Este capítulo visa mostrar as motivações, desafios, conceitos relacionados (estado desejável, estado indesejável, restrições, função-objetivo, espaço de busca, otimização *off-line* e otimização *online*), interação com as diferentes áreas do conhecimento (PU, SD, CA, PO e ADSC), papéis dos desenvolvedores de SDAs (desenvolvedor de sistemas, administrador de sistemas, administrador de recursos computacionais e desenvolvedor/administrador de sistemas), fases e fluxos de trabalho relacionados ao **Processo de Desenvolvimento de *Software* focado em Sistemas Distribuídos Autônomicos** (PDS2DA). Os capítulos anteriores forneceram a fundamentação teórica necessária, bem como uma visão geral e consequente categorização dos trabalhos relacionados.

4.2 Motivação e Objetivos

As principais motivações para a criação do PDS2DA são: **(i)** os benefícios proporcionados pelo PU; **(ii)** as lacunas encontradas nos trabalhos relacionados; **(iii)** a crescente complexidade dos SDs; **(iv)** a completude presente no PDS2DA e **(v)** a flexibilidade proporcionada.

O PU reúne o **conhecimento** e a **experiência** prática de décadas de Engenharia de *Software*. Ele tem sido amplamente usado na indústria e no meio acadêmico, para criação de sistemas de diversas categorias e tamanhos. Além disso, ele é um *framework* genérico de processo, que pode ser especializado para uma ampla gama de sistemas, áreas de aplicação e assim por diante. Sendo assim, ele **pode ser especializado** para conter as áreas de conhecimento relevantes para o PDS2DA, são elas: SDs, CA, PO e ADSC.

Como foi mencionado no Capítulo 3, existe uma lacuna em termos de processo de

desenvolvimento de *software* focado em SDAs. A grande maioria dos trabalhos, simplesmente apresenta um sistema tradicional, depois modifica-o, visando implementar algum aspecto da CA e apresenta seus resultados. Isso pode auxiliar de certa forma, mostrando qual algoritmo ou técnica pode ser usada para implementar determinado aspecto da CA em um SD. O laço de controle MAPE-C, também fornece algumas ideias iniciais de como implementar a arquitetura do SDA, mas apenas isso.

Como foi citado em (HARBAOUI *et al.*, 2010), o gerenciamento de sistemas distribuídos modernos está ficando cada vez mais **complexo** e **caro**. A CA costuma lidar com esses problemas, por meio de seus aspectos de auto-gerenciamento. Segundo (ROSA *et al.*, 2013), os *softwares* atuais (por exemplo: Apache HTTPd Server, Apache Tomcat, MySQL, VMs) oferecem diferentes recursos para configurar seu comportamento, incluindo módulos carregáveis e vários parâmetros configuráveis. Essas facilidades podem ser usadas adaptar o comportamento desses componentes do sistema mesmo em tempo de execução em resposta a mudanças no ambiente de execução. Por exemplo, a carga de trabalho do sistema ou os recursos computacionais disponíveis podem mudar enquanto o sistema está em execução. Apesar da alocação dinâmica de recursos possa ser usada para responder a essas mudanças, adaptações nos componentes do sistema também são uma poderosa ferramenta.

O PDS2DA aborda diversos aspectos da **criação** e **manutenção** do SDA. Usando-se o PU como base, ele começa na definição dos requisitos, passando pela análise, projeto, implementação, testes, integração e manutenção do SDA. De fato, como já foi mencionado no Capítulo 3, os trabalhos relacionados abordam apenas a otimização online do SDA ou propõe algum modelo de arquitetura para o SDA.

Por fim, por ter como base o PU, o PDS2DA é **flexível**. Algumas atividades e artefatos gerados são opcionais. A quantidade de diagramas UML, descrições textuais e assim por diante, depende do tamanho do SDA, do tamanho e diversidade da equipe de desenvolvimento e das decisões de projetos relativos a cada SDA. Por exemplo, na otimização *off-line*, pode-se formalizar o problema usando PO ou somente definir um conjunto de experimentos e realizar uma ADSC.

Sendo assim, o **objetivo** do PDS2DA é permitir o desenvolvimento sistemático de SDAs, usando-se técnicas estatísticas para comparar quantitativamente as diferentes soluções possíveis, em cada etapa da criação e manutenção do SDA.

4.3 Conceitos Relacionados

Primeiramente, é preciso definir alguns conceitos que serão usados no PDS2DA, possibilitando que o usuário saiba o que deve ser feito em cada etapa, que serão explicadas posteriormente.

Em termos gerais, a ideia por trás da CA é fazer com que os sistemas se auto-gerenciem,

dispensando a necessidade de um operador manual do sistema, que deve atuar em cada ocasião em que o sistema se encontra em um **estado indesejável**. Sendo assim, o primeiro passo é definir o que seria um **estado desejável** e um **estado indesejável** do sistema.

Um **estado desejável**, é quando o sistema se encontra em condições aceitáveis de funcionamento, não sendo necessário que o gerenciador autônomo tome alguma providência ou, no caso de um sistema tradicional, que o operador do sistema tome alguma providência.

Um **estado indesejável**, é quando o sistema não se encontra em condições aceitáveis de funcionamento, sendo assim, o gerenciador autônomo deve tomar alguma providência ou, no caso de um sistema tradicional, o operador do sistema deve tomar alguma providência. Essa “providência” é vista como uma sequência de ações que fazem com que o sistema volte a estar em um estado desejável.

Nesse momento, a PO entra com o intuito de fornecer uma formalização para esses conceitos de estado desejável e estado indesejável. Um estado desejável é quando nenhuma das restrições definidas foram violadas, já um estado indesejável é quando uma ou mais das restrições definidas foram violadas.

Por exemplo, suponha um servidor Web em que foi incorporado um gerenciador autônomo visando prover a capacidade de **auto-otimização**. Nesse servidor Web, foi definida uma restrição de tempo médio de resposta das requisições de no máximo dois segundos. Sendo assim, sempre que o tempo médio de resposta for igual ou menor do que dois segundos, o sistema estará em um **estado desejável**. Caso o tempo médio de resposta seja maior do que dois segundos, o sistema se encontrará em um **estado indesejável**.

Pode-se aplicar esses conceitos não somente para a auto-otimização, mas também para qualquer aspecto da CA. Suponha um sistema que faz a composição de WSs baseada em QoS e tem a capacidade de **auto-cura**. Após feita a seleção de quais WSs concretos participarão da composição, pode ser que um ou mais dos WSs concretos selecionados esteja indisponível. Sendo assim, pode-se definir que quando todos os WSs participantes da composição estão disponíveis, o sistema está em um **estado desejável**. Caso um ou mais dos WSs concretos selecionados estejam indisponíveis, o sistema encontra-se em um **estado indesejável**. Sendo assim, o gerenciador autônomo deve agir, selecionando outro WS concreto para substituir o que se encontra indisponível no momento de execução.

Outro conceito “emprestado” da PO são os **pontos no espaço de busca** e o **espaço de busca** propriamente dito. Os pontos no espaço de busca, no caso de SDAs, são definidos como uma combinação dos parâmetros do sistema, da carga de trabalho atual e da carga de trabalho prevista (caso seja usada uma técnica de previsão de carga de trabalho) que formam um estado do sistema, seja ele desejável ou indesejável.

Por exemplo, suponha um SDA que gerencia MVs, formando um sistema composto basicamente de três aplicações: um servidor Web, um servidor de aplicação e um SGBD.

Sendo assim, quando o sistema é inicializado, têm-se três MVs em execução: uma executando um servidor Web, outra executando um servidor de aplicação e outra executando um SGBD. Considere-se também, que o sistema permite a inicialização de mais sete MVs idênticas, sendo que pode-se determinar se elas executarão uma cópia do servidor Web, do servidor de aplicação ou do SGBD. Dessa maneira, o que o gerenciador autônomo deve decidir é quais as quantidades de MVs dedicadas a cada componente, variando a quantidade entre três (no mínimo uma para cada componente) até dez MVs. Sendo assim, cada combinação possível, juntamente com a carga de trabalho atual formariam um ponto no espaço de busca. Por exemplo, uma MV para cada componente seria um ponto no espaço de busca. Duas MVs para cada componente seria outro ponto no espaço de busca e assim por diante. A carga de trabalho influencia, pelo fato das restrições que foram definidas. Por exemplo, para uma determinada carga de trabalho, pode ser que nenhuma restrição seja violada e uma MV para cada componente seja suficiente, estando assim, o sistema em um estado desejável. Se dobrarmos ou triplicarmos essa carga de trabalho atual, uma MV para cada componente torna-se insuficiente (pois viola-se alguma restrição definida), sendo assim, o sistema encontra-se em estado indesejável. O conjunto de todos os pontos no espaço de busca é denominado espaço de busca. Usando esse mesmo exemplo do gerenciador de MVs, pode-se definir outros conceitos importantes, as **variáveis de decisão** e as **variáveis externas**.

As **variáveis de decisão**, são aquelas que estão sob o controle do gerenciador autônomo ou do operador manual do sistema. No exemplo, a quantidade de MVs em execução e qual tipo de aplicação que cada MV executará (servidor Web, servidor de aplicação ou SGBD) são características que podem ser definidas pelo gerenciador autônomo. A carga de trabalho, por outro lado, não pode ser definida pelo gerenciador autônomo, tratando-se, portanto, de uma **variável externa**. É importante ressaltar que os pontos no espaço de busca são formados pela combinação dos valores das variáveis de decisão e também das variáveis externas.

O **ótimo global**, no caso dos sistemas autônomos e do PDS2DA, trata-se da combinação de variáveis de decisão que fornece o melhor estado possível do sistema. Para fins de simplificação, suponha um sistema com auto-otimização, sendo que deve-se minimizar o tempo médio de resposta. Imagine que só existem três configurações possíveis: ouro, prata e bronze. Na configuração ouro, o tempo médio de resposta do sistema é um segundo, na prata são três segundos e na bronze são sete segundos. Sendo assim, fica evidente que a configuração ouro é o ótimo global, pois é ela que fornece o melhor estado possível do sistema.

Por fim, o último conceito importante é a **função-objetivo** ou função que deve ser otimizada. Como foi explicado no capítulo de PO, trata-se da função que deverá ser minimizada ou maximizada, sempre buscando o ótimo global quando possível.

Para definir quais RFs e RNFs serão considerados no modelo matemático que será criado, seja na otimização *off-line*, no serviço prestado pelo SDA ou no problema solucionado pelo GA, deve-se seguir o seguinte fluxo de trabalho.

Primeiramente, deve-se considerar que os RFs serão **mapeados em componentes do sistema**, que executarão uma ou mais das funcionalidades necessárias. Em termos de WSs, um RF ou conjunto de RFs relacionados serão mapeados em um Web service abstrato.

Por exemplo, suponha o Web service abstrato: "Buscar Passagens". Ele deve permitir que o usuário escolha a cidade de partida e chegada, se deseja comprar uma passagem apenas de ida ou de ida e volta, qual classe gostaria de viajar e assim por diante. Todos esses RFs são mapeados para o WS abstrato "**Buscar Passagens**".

Posteriormente, esses componentes (ou WS abstratos) serão **mapeados em variáveis de decisão** no modelo matemático que será criado. Ou seja, como já foi explicado, cada WS abstrato possui uma série de WS concretos que implementam suas funcionalidades, sendo assim, pode-se escolher qual será usado, baseando-se em algum critério, neste caso, nos atributos de QoS custo e TME. A Figura 14 mostra os mapeamentos envolvidos.

Figura 14 – Mapeamento dos Requisitos Funcionais

Além disso, devem-se considerar quais RNFs serão incluídos no modelo matemático. Os RNFs diretamente relacionados com o problema em questão, serão **mapeados em atributos de QoS**. Por exemplo, custo, TME, confidencialidade, disponibilidade e assim por diante. Posteriormente, esses atributos de QoS **serão mapeados em cláusulas de SLA**. Por exemplo, pode-se definir uma cláusula de SLA dizendo que o orçamento de cada plano de composição que será criado, não deve ultrapassar uma determinada faixa (número absoluto ou relativo), definida pelo usuário. Por fim, essas cláusulas de SLA serão **mapeadas em restrições** ou na própria **função-objetivo** do modelo matemático. A Figura 15 mostra os mapeamentos envolvidos.

Figura 15 – Mapeamento dos Requisitos Não-Funcionais

Deve-se considerar apenas os atributos de QoS/SLA diretamente ligadas ao problema que o modelo matemático irá formalizar. Por exemplo, RNFs relacionados a tecnologia usada (por exemplo, qual SGBD deve ser usado) não deve ser inserida no modelo matemático.

Neste momento, a consulta aos trabalhos relacionados é de **vital** importância. A PO tem sido aplicada em diversos problemas relacionados a SDs e muitos problemas atuais podem ser modelados usando algum problema padrão da PO ou adaptando-se o problema atual para algum problema padrão. Além disso, padronizando o modelo matemático em alguma forma clássica, permite usar diversas ferramentas/bibliotecas/algoritmos já implementados e amplamente testados no meio acadêmico e na indústria.

4.4 Papéis presentes no PDS2DA

O PDS2DA possui os seguintes papéis (além dos presentes no PU): desenvolvedor de sistema, administrador de sistema, desenvolvedor/administrador de sistema e usuário de sistema. Isso ocorre principalmente pelo fato de alguns *softwares* não possuírem o código fonte aberto ou mesmo pelo desenvolvedor não possuir capacidade e/ou tempo para realizar modificações no código fonte de alguns componentes do sistema. As características de cada papel são:

- **Desenvolvedores de sistemas:** são aqueles que tem acesso ao código fonte, portanto podem modificar os programas, de acordo com suas necessidades. Sendo assim, podem melhorar o desempenho dos sistemas, por exemplo, usando algoritmos mais eficientes.
- **Administradores de sistemas:** são aqueles que são responsáveis pela implantação, configuração e manutenção dos sistemas, mas não possuem acesso ao código fonte. Sendo assim, podem escolher quais componentes farão parte do sistema, podem configura-los, criar mecanismos externos de monitoramento e configuração automática, realizar avaliações de desempenho e até criar seus próprios *softwares* (por exemplo, seu gerenciador autônomo), sem modificar o código fonte dos *softwares* usados (por exemplo, servidores Web, SGBDs, servidores de aplicação, etc.).
- **Desenvolvedor/administrador de sistemas:** em alguns casos, pode ocorrer que os papéis de desenvolvedor e administrador de sistemas sejam exercidos pelo mesmo profissional. Isso permite um alto grau de modificação e configuração dos sistemas, podendo-se usar diversas metodologias e ferramentas para auxiliar na modificação interna (código fonte) e externa (parâmetros configuráveis estática e dinamicamente) dos sistemas computacionais.
- **Usuário de sistemas:** Em alguns casos, o sistema sendo desenvolvido se integra com subsistemas externos. Por exemplo, um sistema de vendas pode consultar um WS externo, para verificar as dívidas relativas ao CPF de um determinado cliente. Nesse caso, o desenvolvedor é apenas um usuário daquele subsistema, não tendo acesso ao seu código fonte e não podendo modificar seus parâmetros de configuração.

Sendo assim, para cada componente do sistema, deve-se avaliar qual o papel do desenvolvedor. Por exemplo, suponha um *software* de vendas online denominado **VendasPira**. Ele é

Tabela 14 – Papéis e metodologias de ADSC.

Papel	Metodologias
Desenvolvedor de sistemas	Análise do estado das <i>threads</i> , <i>CPU profiling</i> , caracterização da carga de trabalho, metodologia USE, análise de cima para baixo e método científico.
Administrador de sistemas	<i>Benchmarking</i> , estatísticas de base, lista de checagem padrão e caracterização da carga de trabalho.
Desenvolvedor/administrador de sistemas	Neste caso, todas as metodologias citadas para os papéis de desenvolvedor de sistemas e administrador de sistemas podem ser usadas.

composto por: um subsistema de vendas, feito usando a linguagem de programação Java, um SGBD com os dados dos produtos disponíveis e um WS externo, fornecido pela *MasterCard* para validação dos dados de cartão de crédito dos clientes. Dessa forma, pode-se assumir que o código fonte do subsistema de vendas está disponível ao desenvolvedor do VendasPira, sendo assim, para esse componente, o papel dele é: **desenvolvedor de sistema**. Já o SGBD, será configurado, mas não terá seu código fonte modificado (ou porque ele não está disponível ou por decisão do desenvolvedor do VendasPira), sendo assim, para o SGBD, o papel do desenvolvedor é: **administrador de sistema**. Por fim, o desenvolvedor do VendasPira não tem acesso ao código fonte e nem pode configurar o WS externo da *MasterCard*, sendo assim, ele é um **usuário do sistema**.

Para cada papel possível, diferentes metodologias de avaliação de desempenho estão disponíveis. A Tabela 14 mostra algumas metodologias, sendo que todas são citadas no livro de (GREGG, 2014).

4.5 Fases do PDS2DA

Como já foi citado, o PDS2DA é uma especialização do PU, sendo assim, grande parte do seu conteúdo é semelhante. As fases são as mesmas: **iniciação**, **elaboração**, **construção** e **transição**. Além disso, os fluxos de trabalho também são os mesmos: **requisitos**, **análise**, **projeto**, **implementação** e **testes**. Entretanto, o PDS2DA possui algumas **atividades e artefatos adicionais**, justamente visando integrar as diferentes áreas de conhecimento que o compõe.

4.5.1 Iniciação

Nesta fase, tudo que já foi mencionado no capítulo sobre PU também ocorre no PDS2DA. Entretanto, devido ao enfoque em SDAs, mais alguns aspectos devem ser considerados.

Nesse momento, deve-se focar mais em o que precisa ser feito, ao invés de como será feito. Pensando em termos de SDAs, deve-se definir qual ou **quais aspectos da CA serão implementados** no SDA. Isso dependerá do escopo do SDA e de quais necessidades serão

Tabela 15 – Atividades e artefatos gerados na Fase de Iniciação.

Atividades	Artefatos
Listar principais requisitos funcionais e não-funcionais.	Descrição dos requisitos funcionais e não-funcionais.
Definir qual aspecto da CA será abordado.	Esboço da arquitetura base do SDA.
Definir se o SDA usará PO para implementar o serviço que ele provê.	Estimativas de custos, cronograma, etc.
Definir se o gerenciador autônomo usará PO.	

atendidas usando-se a CA. Por exemplo, pode-se querer transformar um SD que faz a CWSbQ em um SDA, com capacidade de tolerância a falhas, recuperando-se automaticamente quando algum WS invocado em tempo de execução estiver indisponível.

Ao estudar CA, nota-se que o aspecto que lida com a recuperação automática dos SDAs é a autocura. Sendo assim, sabe-se que nesse dado SDA, deverá ser implementado e/ou usado algum algoritmo/framework/ferramenta de *software* visando a autocura.

Além disso, deve-se definir se o SDA usará a PO para criar um modelo matemático que será solucionado com algum algoritmo, visando atender aos **requisitos funcionais** e/ou **requisitos não-funcionais** do SDA. Nesse caso, além do conhecimento do escopo do SDA, é preciso levar em consideração a PO. Esse conjunto de RFs representam o serviço que SDA irá prover. Por exemplo, o serviço de um servidor Web é atender requisições HTTP, o serviço de um SGBD e atender consultas feitas em SQL e assim por diante. É importante ressaltar que o uso da PO é opcional.

Por fim, deve-se definir se o gerenciador autônomo usará a PO para criar um modelo matemático do problema que será resolvido por ele. Também se ressalta que o uso da PO no GA não é obrigatória. A Tabela 15 mostra um resumo do que deverá ser realizado nesta fase. Nesta fase, o fluxo de trabalho principal são os **requisitos**.

4.5.2 Elaboração

Da mesma maneira, nesta fase tudo que já foi mencionado no Capítulo 2 sobre o PU também deve ser feito no PDS2DA. Em adição, algumas outras atividades devem ser conduzidas nesta fase.

Deve-se definir se será feita somente a **otimização online** ou também a **otimização off-line** do SDA. Se a otimização *off-line* for feita, deve-se formalizar o problema usando PO ou então realizar o planejamento de experimentos, escolhendo os fatores, níveis e variáveis de resposta de interesse. Por exemplo, pode-se definir que existem cinco opções de SGBDs para o SDA e que se deseja o mais rápido deles. Sendo assim, o fator seria o SGBD, os níveis seriam os diferentes SGBDs disponíveis (MySQL, Oracle, etc.) e a variável de resposta seria o tempo médio

de resposta dos SGBDs. Neste exemplo, apenas um fator foi selecionado. Entretanto, podem haver quantos fatores se julgar necessário, respeitando-se as restrições de tempo e financeiras. Além disso, as metodologias de avaliação de desempenho citadas anteriormente, relativas aos diferentes papéis presentes no PDS2DA também podem ser empregadas.

Outra atividade importante nesta fase, é a **identificação do papel do desenvolvedor** ou grupo de desenvolvedores do SDA. As diferentes funções e possibilidades de cada papel, já foram explicadas anteriormente. Entretanto, nesta fase, é importante explicitar o papel do desenvolvedor do SDA em termos dos componentes do SDA. Isso ocorre, porque a maioria dos SDs modernos são compostos de outros SDs, que acabam virando subsistemas do SD em questão. Por exemplo, um SD pode ser composto de um servidor Web, um servidor de aplicação e um SGBD. Sendo assim, cada um desses elementos torna-se um subsistema do SD citado.

Dessa forma, por exemplo, o servidor Web pode ter o código fonte aberto, permitindo que o desenvolvedor do SDA faça modificações que julgar necessárias, mas o SGBD seja de código fechado, impossibilitando que o desenvolvedor do SDA modifique ou mesmo tenha acesso ao seu código fonte. Sendo assim, para cada componente ou subsistema, deve-se avaliar se o desenvolvedor do SDA irá ter o papel de desenvolvedor de sistema, administrador de sistema, desenvolvedor/administrador de sistema ou usuário do sistema.

Nesta fase, é crucial um estudo dos trabalhos relacionados, para se ter conhecimento de como a PO pode ser aplicada ao SDA (quais são os algoritmos disponíveis, *frameworks* e assim por diante), quais algoritmos e/ou *frameworks* foram usados para implementar o aspecto da CA que foi selecionado na fase anterior e quais técnicas de avaliação de desempenho podem ser usadas no SDA, pensando tanto na otimização *off-line* como na otimização online.

Como foi mencionado no capítulo sobre o PU, na fase de elaboração, são executadas algumas tarefas relacionadas a Análise, Projeto e até mesmo Implementação. Sendo assim, deve-se considerar como a CA se inclui dentro desses fluxos de trabalho.

No Capítulo 2, foi mencionado que é possível implementar a CA por meio de diferentes arquiteturas, sendo as principais: **centralizada, distribuída e hierárquica**. Deve-se também definir se o SDA usará uma das três abordagens de auto-gerenciamento possíveis: **(i)** proativa, **(ii)** reativa ou **(iii)** proativa e reativa.

Por fim, a maioria dos RFs e RNFs devem ter sido definidos ao final desta fase. Mesmo que isso demande um certo tempo, é melhor do que começar a implementação e ser necessária sua modificação no decorrer do projeto. Nesta fase, os fluxos de trabalho **requisitos, análise e projeto** são os principais. Entretanto, alguma implementação também é realizada. A Tabela 16 mostra as atividades e artefatos gerados nesta fase.

Tabela 16 – Atividades e artefatos gerados na Fase de Elaboração

Atividades	Artefatos
Definir se será realizada otimização off-line.	Formalização do problema de otimização off-line ou planejamento de experimentos.
Definir quais algoritmos/técnicas/frameworks serão usados para otimização online.	Formalização do serviço que o SDA provê e/ou do gerenciador autonômico.
Estudo dos trabalhos relacionados.	Lista com as vantagens/desvantagens de cada algoritmo/técnica para o aspecto da CA escolhido.
Definição da arquitetura do gerenciador autonômico.	Diagrama da arquitetura do gerenciador autonômico.
Definição da abordagem usada pelo gerenciador autonômico (proativa ou reativa).	Lista com o papel do desenvolvedor em relação a cada componente do SDA.

4.5.3 Construção

Deve-se levar em conta a CA em relação ao SDA como um todo. A CA não deve ser vista como apenas um componente isolado do SDA. Para que o GA possa executar suas tarefas, ele deve ser capaz de interagir com os demais componentes do SDA, usando seus **sensores** e **atuadores**.

Todos os módulos do laço de controle MAPE-C devem ser implementados nesta fase. Entretanto, como em todo *software* em desenvolvimento, deve-se evitar o trabalho desnecessário e tentar aproveitar ao máximo os algoritmos, *frameworks* e ferramentas de *software* disponíveis, desde que sejam capazes de realizar as tarefas necessárias.

No Capítulo 3, são citadas diversas contribuições capazes de auxiliar o desenvolvedor do SDA nesse sentido. Ademais, deve-se lembrar que muitos *softwares* disponíveis (SGBDs, Servidores Web, etc.) possuem recursos próprios de monitoramento, que podem ser usados pelo gerenciador autonômico, evitando que o desenvolvedor do SDA precise implementar tais funções.

Por fim, como foi citado no Capítulo 2 na seção sobre ADSC, a mais rápida e mais eficaz forma de aumentar o desempenho de um sistema computacional, é eliminando o trabalho desnecessário, principalmente nas camadas mais altas da arquitetura de *software*. No fim dessa fase, a maior parte do SDA deve ter sido implementada com sucesso. Deve-se começar a planejar como será feita a ADSC do GA, para comparar suas soluções, caso existam mais de uma.

O significado desse melhor desempenho, está obviamente ligado ao aspecto da CA que foi implementado e qual era o objetivo visado ao implementá-lo. Por exemplo, se foi criado um Servidor Web autonômico com auto-otimização, visando reduzir o tempo médio de resposta das suas requisições, então isso será avaliado. Se foi selecionado o aspecto de autoproteção, visando aumentar a disponibilidade do Servidor Web autonômico, então essa disponibilidade deverá ser maior do que a do Servidor Web tradicional e assim por diante.

Nesta fase, os fluxos de trabalho principais são: **projeto** e **implementação**. A Tabela 17 mostra quais são os fluxos de trabalho principais, as atividades relacionadas e os artefatos gerados ao final desta fase.

Tabela 17 – Atividades e artefatos gerados na Fase de Construção

Atividades	Artefatos
Implementação do SDA.	Arquivos de código fonte, arquivos de configuração, tabelas de bancos de dados, etc.
Otimização <i>off-line</i>	Scripts de configuração.
Distribuição dos componentes nos nós computacionais.	Arquivos executáveis.
Testes unitários	Configuração inicial dos componentes.
	Diagramas de implementação/implantação.

4.5.4 Transição

Nesta fase, os testes de integração e de sistema devem ser finalizados. Possíveis defeitos encontrados no SDA devem ser corrigidos e testados novamente. A ADSC deve ser empregada, com o objetivo de garantir que o desempenho do SDA é satisfatório em relação aos possíveis cenários atuais e de preferência em relação a alguns cenários futuros previstos.

Como foi citado no Capítulo 2 na seção sobre ADSC, técnicas como planejamento de capacidade, estão em crescente declínio. Isso se deve a uma série de fatores, os principais são: (i) a dificuldade na criação e validação de modelos dos SDs, tendo em vista que estão cada vez mais complexos, heterogêneos, dinâmicos e distribuídos; (ii) o baixo custo do aluguel temporário de recursos computacionais na nuvem, que podem ser usados apenas para a execução dos experimentos ou *benchmarks* definidos e (iii) a falta de controle sobre os componentes do SDA.

Em relação ao terceiro item, faz-se necessária uma explicação mais detalhada. O uso de serviços fornecidos por outras organizações está cada vez maior. Isso ocorre em diferentes camadas da arquitetura do *software*, desde uma simples busca usando um navegador Web (respondida por um SaaS) até o uso de recursos computacionais (fornecido por um serviço de IaaS). Sendo assim, o usuário do SDA é apenas mais um usuário (ou mais uma requisição feita a um SaaS) e não o gerenciador, ou responsável por aquele serviço. Sendo assim, informações necessárias para o uso de planejamento de capacidade não estão disponíveis para o usuário do SDA. Tempo médio de execução, vazão, taxa de chegada de requisições e assim por diante, não são informações disponíveis para o usuário do SDA.

Sendo assim, nesta fase, recomenda-se seguir uma abordagem mais **experimental**, como citada no Capítulo 2 na seção sobre ADSC, baseada em planejamento de experimentos, cálculo da influência dos fatores e assim por diante. Nesta fase, o principal fluxo de trabalho são os

testes. A Tabela 18 mostra quais são os fluxos de trabalho principais, as atividades relacionadas e os artefatos gerados ao final desta fase.

Tabela 18 – Atividades e artefatos gerados na Fase de Transição

Atividades	Artefatos
Testes de integração.	Casos de teste.
Testes de sistema	Procedimentos de teste.
Deteção e correção de falhas.	Criação e/ou uso de ferramentas automatizadas de testes.
ADSC com cenários de teste com o diferentes soluções para o gerenciador autônomo (caso exista mais de uma)	Comparação, usando ADSC de diferentes soluções para o gerenciador autônomo (caso exista mais de uma).
	Diagramas de implementação/implantação.

4.6 Fluxos de Trabalho

Por se tratar de uma especialização do PU, o PDS2DA possui os mesmos fluxos de trabalho. Entretanto, algumas atividades e artefatos gerados no PDS2DA não estão presentes no PU. Isso se deve, principalmente, devido a inclusão da PO, CA e ADSC, não presentes no PU.

4.6.1 Fluxo de Trabalho: Requisitos

Como foi citado no capítulo do PU, o intuito desse fluxo de trabalho é guiar o desenvolvimento do sistema correto. Sendo assim, deve-se pensar tanto em termos de RFs, como em RNFs.

Os formalismos e detalhes técnicos devem ser deixados de lados, para que seja possível um consenso entre todas as partes interessadas no sistema (clientes, usuários, desenvolvedores, etc.). Em termos da CA, deve-se pensar no mapeamento entre RFs, que futuramente serão considerados como variáveis de decisão. Além disso, os RNFs, serão mapeados futuramente como restrições que deverão ser cumpridas. Lembrando que os RNFs podem ser específicos para um determinado **componente**, ou focar no SDA como um **todo**.

Os fluxos de trabalho de análise e projeto, explicarão esse mapeamento em maiores de detalhes, mas durante os requisitos, basta saber que um ou mais RFs serão, primeiramente, mapeados em um componente do sistema e posteriormente em variáveis de decisão, tanto na otimização *off-line*, como na otimização *online*. Os RNFs, serão mapeados como **restrições** que deverão ser cumpridas pelo GA, durante a otimização *online*.

4.6.2 Fluxo de Trabalho: análise

Como foi citado no capítulo do PU, neste fluxo de trabalho, os requisitos que foram capturados no fluxo anterior, devem ser agora analisados por meio de refinamento e estruturação.

A principal diferença em relação ao fluxo de requisitos, é que neste fluxo é usado uma linguagem formal, ou seja, a linguagem dos desenvolvedores do *software*.

Os requisitos devem ser estruturados em classes de análise e pacotes, fornecendo uma visão interna do sistema. Em termos da CA, deve-se definir se a otimização *off-line* será empregada. Caso ela seja, deve-se escolher se a PO será usada ou se será feita apenas ADSC. Isso fornece os dados necessários para no próximo fluxo, o de projeto, definir-se qual algoritmo da PO será usado para solucionar-la ou como empregar a ADSC.

Além disso, deve-se definir as **variáveis de decisão** e **restrições** que serão usadas pelo GA. Com as variáveis de decisão, qualquer algoritmo ou técnica que será usado posteriormente, saberá quais parâmetros e/ou recursos computacionais poderá modificar em tempo de execução. Com as restrições, é possível definir quando o SDA se encontra em um **estado desejável** ou **estado indesejável**. Se o gerenciador autônomo usar a PO para formalizar seu problema, neste fluxo deverá ser definida a **função-objetivo**.

É importante ressaltar que a otimização *online* é de responsabilidade do GA. Deve-se definir também qual arquitetura do GA será escolhida. No capítulo da CA, as principais citadas são: centralizada, distribuída e hierárquica.

Por fim, deve-se pensar como o laço MAPE-C será implementado, levando-se em consideração todos os aspectos citados no capítulo da CA.

4.6.3 Fluxo de Trabalho: projeto

Como foi mencionado no capítulo do PU, neste fluxo o sistema é moldado e encontra-se sua forma (incluindo sua arquitetura) que atende todos os requisitos (funcionais, não-funcionais e demais restrições). Uma entrada essencial para esse fluxo são os diagramas criados no fluxo de análise.

Em termos de CA, deve-se detalhar como cada componente do laço MAPE-K será definido. Quais **dados** deverão ser monitorados, quais serão os componentes **sensores** e **atuadores**, qual a frequência e forma de coleta dos dados e assim por diante. A interação entre os diferentes componentes da CA também deverá ser detalhada. Além disso, deve-se detalhar quais técnicas/algoritmos/*frameworks*/ferramentas de *software* serão empregadas no módulo de análise, no módulo de planejamento e na base de conhecimento.

A arquitetura dos componentes da CA também deverá ser detalhada. Caso a PO tenha sido usada para formalizar o problema que o GA deve resolver, o algoritmo que será usado deverá ser definido e especificado em termos de classes de projeto, interfaces e assim por diante.

Por fim, vale lembrar que é durante este fluxo de trabalho que o modelo de implantação é criado. Em termos da CA, deve-se definir como cada módulo do laço MAPE-K será distribuído, como será a interação com os demais componentes do sistema e assim por diante. Para isso,

deve-se levar em conta alguns fatores: tamanho do SDA, abordagem empregada pelo GA (por exemplo, reativa), qual aspecto da CA está sendo implementada e assim por diante. Por exemplo, se a CA está sendo usada visando aumentar a disponibilidade do SDA, provavelmente uma arquitetura centralizada não deverá ser usada (por ser um ponto único de falha).

4.6.4 Fluxo de Trabalho: implementação

Como foi citado no capítulo do PU, é neste fluxo de trabalho que ocorre a criação dos códigos fonte, *scripts*, binários, executáveis e assim por diante. Em termos da CA, todos os módulos do laço MAPE-C devem ser implementados e testados. A distribuição do SDA deve ocorrer por meio do mapeamento dos componentes executáveis em nós computacionais, baseando-se no modelo de implantação.

De fato, o GA é um subsistema do SDA, devendo seguir o padrão de fraco acoplamento e alta coesão. Deve-se levar em conta que a quantidade de RGs pode mudar em tempo de execução, sendo assim, a implementação do GA deve levar em conta essa flexibilidade.

Ademais, pode ser usada ADSC para garantir que o GA não é um gargalo para o SDA. Se for identificado como gargalo, as técnicas de otimização *off-line* deverão ser usadas para solucionar o problema.

4.6.5 Fluxo de Trabalho: testes

Como foi citado no capítulo do PU, neste fluxo são verificados os resultados das implementações de cada entrega, tanto das intermediárias como as finais, bem como a versão final do sistema. Em relação a CA, deve-se garantir que o SDA, comporta-se de maneira correta em diferentes cenários de execução. Ele deve ser testado em cenários em que o GA não precisa realizar nenhuma ação e também em cenários onde ele realiza uma ou mais ações.

Sendo assim, devem ser feitos testes com o subsistema autônomo ativado e desativado. Apesar de uma determinada sobrecarga no SDA devido ao gerenciador autônomo ser esperada, deve-se garantir que ela não seja significativa o suficiente para prejudicar seu desempenho.

Como já foi mencionado anteriormente, esse ganho de desempenho está relacionado ao aspecto da CA implementado, bem como as restrições que foram definidas.

4.7 Considerações finais

Este capítulo detalhou os conceitos do PDS2DA, suas fases e fluxos de trabalho. Além disso, mostrou como as diferentes áreas do conhecimento se integram ao PU e formam o PDS2DA. No próximo capítulo, ele será aplicado a três estudos de caso, visando detalhar seu uso e também servir como uma prova de conceito.

ESTUDOS DE CASO

Este capítulo visa mostrar como o PDS2DA pode ser aplicado, usando como exemplo três estudos de caso, sendo que o tema central será a seleção de WSs baseada em QoS. Sendo assim, a ideia é definir os principais RFs e RNFs e usar o PDS2DA para criar três protótipos que servirão como **prova de conceito** e também para demonstrar como empregar o PDS2DA na construção de um SDA.

Serão abordadas tanto a otimização *off-line*, quanto a otimização *online*. Portanto, três problemas **NP-Difícil** serão solucionados neste capítulo, são eles: **(i)** seleção da arquitetura de base que atenda aos RFs e RNFs (problema da mochila com múltiplas escolhas, na otimização *off-line*); **(ii)** seleção *online* dos WSs baseada em QoS que farão parte do plano de composição (problema da mochila com múltiplas escolhas, o "serviço" que o SDA provê) e **(iii)** escalonamento autônomo das requisições pelo SDA (problema da atribuição generalizado, a forma do GA prover auto-otimização).

5.1 Seleção de Web services baseada em QoS

Esse é o serviço que o SDA proverá. Ou seja, o SDA deve implementar funcionalidades que permitam que ele realize a seleção dos WSs baseada em atributos de QoS. Como já foi mencionado no capítulo sobre SDs, a composição de WSs pode ser dividida em dois aspectos: seleção dos WSs baseada em QoS e criação do plano de composição. Focou-se na seleção de WSs baseada em QoS.

Primeiramente o problema de seleção de WSs baseada em QoS será definida de maneira informal, posteriormente, o **modelo matemático** criado será apresentado. Sendo assim, suponha-se que existam **tarefas** que devam ser realizadas pelo Web service composto que deverá ser criado. Cada tarefa, pode ser definida, como por exemplo: buscar passagens aéreas, reservar um quarto de hotel, alugar um carro e assim por diante. Levando-se em conta as recomendações

do PU e dos WSs, essas tarefas devem ser fracamente acopladas, ou seja, aplicações diferentes devem ser responsáveis por tarefas diferentes. De fato, para cada uma dessas tarefas, um WS deve ser apto a realiza-la. Sendo assim, cada tarefa é denominada um **Web service abstrato**. Cada WS que realiza uma tarefa é denominado um **Web service concreto**.

Dado um determinado conjunto de WSs abstratos, suponha-se m , existem n WSs concretos para cada WS abstrato. Sendo assim, deve-se criar um **plano de composição**, ou seja, escolher um conjunto de WSs concretos que atenda a todos os WSs abstratos, baseando-se nos atributos de QoS dos WSs concretos e em alguma restrição imposta aos WSs abstratos. Cada WS concreto possui um **custo** e também um **Tempo Médio de Execução (TME)**. Se houver um **orçamento máximo permitido**, deve-se criar um plano de composição cujo custo total seja menor ou igual ao orçamento definido. A Figura 16 mostra esses conceitos.

Figura 16 – WSs abstratos, WSs concretos, atributos de QoS e restrição.

5.2 Aplicação do PDS2DA

Uma das vantagens do PU é que ele pode ser aplicado a sistemas de diversos tamanhos, desde os mais reduzidos, feitos por apenas alguns desenvolvedores, até sistemas enormes, que levam meses ou até anos para serem concluídos, e envolvem dezenas de desenvolvedores.

Ademais, o PU é um *framework* genérico de processo de desenvolvimento, podendo ser adaptado de acordo com as necessidades de cada *software* que será desenvolvido.

Sendo assim, levando-se em conta que os objetivos principais dos estudos de caso é ser uma **prova de conceito** e um **guia** de como usar o PDS2DA para desenvolver um SDA e também o tamanho do protótipo ser deveras reduzido, a aplicação do PDS2DA constará, para cada estudo de caso, de: **(i)** definição dos requisitos funcionais e requisitos não-funcionais; **(ii)** definição informal do problema; **(iii)** criação do modelo matemático; **(iv)** implementação das soluções e **(v)** avaliação de desempenho.

Tabela 19 – Requisitos Funcionais do Estudo de Caso 1.

Requisito	Descrição
RF 1.1	Para cada WS abstrato, deve-se selecionar um WS concreto.
RF 1.2	Os atributos de QoS considerados são: custo e TME.
RF 1.3	Deve-se minimizar o TME do plano de composição.
RF 1.4	Deve-se cumprir a restrição de orçamento definida.
RF 1.5	Deve-se criar uma lista de todos os planos de composição que cumprem a restrição de orçamento.
RF 1.6	Deve-se buscar o ótimo global ou o mais próximo possível.

Tabela 20 – Requisitos Não-Funcionais do Estudo de Caso 1.

Requisito	Descrição
RNF 1.1	A lista mencionada no RF 1.5 deve ser salva em armazenamento secundário, para consulta posterior.
RNF 1.2	Idem para o RF 1.6.

5.3 Estudo de Caso 1: Otimização *off-line*

Como foi mencionado no Capítulo 4, pode-se optar por realizar a otimização *off-line* de duas formas: (i) formalizando-a usando PO, resolvendo o problema de PO e fazendo ADSC ou (ii) usar uma ou mais metodologias de avaliação de desempenho para testar diferentes componentes/configurações para o SDA. Sendo assim, optou-se pela primeira opção.

5.3.1 Definição dos Requisitos

Constatou-se que os dois atributos de QoS selecionados possuem comportamentos diferentes. O custo de um WS concreto é atualizado **mensalmente** e seu TME é atualizado a cada hora. Usando-se as técnicas de levantamento de requisitos presentes no PU, determinou-se o seguinte conjunto de RFs e RNFs. A Tabela 19 exibe os RFs e a Tabela 20 exibe os RNFs.

5.3.2 Componentes, papéis e arquitetura

Baseando-se nos RFs e RNFs, definiu-se que seriam usados dois componentes: uma aplicação usando a linguagem de programação Java, que implementaria os algoritmos usados para resolver o modelo matemático proposto e um SGBD, onde seria possível consultar os dados relativos ao QoS dos WSs concretos, bem como possibilitar o cumprimento dos requisitos não-funcionais.

Dessa forma, para a aplicação Java, definiu-se que o papel do desenvolvedor seria: **desenvolvedor/administrador** do sistema. Já para o SGBD, foi definido que o desenvolvedor seria: **usuário do sistema**. Ademais, foi selecionado o SGBD MySQL.

Para a arquitetura, os algoritmos seriam executados em uma aplicação Java *desktop* e o

SGBD estaria presente nesta mesma máquina. A aplicação faria uma consulta SQL ao SGBD, que retornaria os dados sobre os atributos de QoS dos WSs, elaboraria o plano de composição e salvaria os resultados novamente no banco de dados.

Por se tratar de um grupo de requisitos pequeno, não julgou-se necessário a criação de diagramas ou demais explicações a respeito.

5.3.3 Modelagem

Esta seção visa explicar como aplicar a PO para realizar a otimização *off-line*. Apesar de ser específica para o SDA em questão, a ideia é que os mesmos conceitos sejam aplicados na criação de outros SDAs.

Como foi citado no capítulo de PO, as fases para criação do modelo matemático que representará o problema são: **modelagem, análise, inferência e julgamento**. Sendo assim, os RFs e RNFs definidos serão considerados na criação do modelo matemático deste estudo de caso.

Para realizar seleção dos WSs *off-line*, optou-se por criar um modelo matemático que leva em consideração os atributos de QoS mencionados e a restrição orçamentária.

A escolha (ou criação) do modelo matemático depende de duas coisas: (i) do conhecimento do desenvolvedor sobre o problema que deverá ser modelado e (ii) do conhecimento do desenvolvedor sobre PO. Sendo assim, após consultar dezenas de artigos relacionados a seleção de WSs baseada em QoS, levando-se em conta os atributos de QoS definidos e a restrição, chegou-se a conclusão que o problema se encaixa em uma variação dos problemas da mochila, o **problema da mochila com múltiplas escolhas** (*Multiple-Choice Knapsack Problem*).

Os atributos de QoS foram gerados aleatoriamente, sendo esta a abordagem padrão na grande maioria dos trabalhos relacionados a seleção de WSs baseada em QoS (CANFORA; ESPOSITO, 2004), (CANFORA *et al.*, 2005), (FANJIANG *et al.*, 2010), (ARDAGNA; PERNICI, 2005), (ZHANG *et al.*, 2010). Isso ocorre porque estimar os valores de atributos de QoS não são o foco do trabalho, mas sim, como realizar a seleção dos WSs baseando-se em tais atributos. Por outro lado, alguns trabalhos focaram em técnicas para estimar os atributos de QoS, como pode ser visto nos trabalhos de (PEDERSEN *et al.*, 2011), (ZHENG; ZHANG; LYU, 2014) e (CASALICCHIO; SILVESTRI, 2012).

De fato, no trabalho de (SHAO *et al.*, 2012) são citadas algumas técnicas conhecidas para a predição do tempo médio de resposta de WSs são elas: **Média Móvel** (*Moving Average*), **Média Móvel Ponderada Exponencialmente** (*Exponentially Weighted Moving Average*), **Média Móvel Ponderada Exponencialmente Auto-adaptável** (*Self-Adaptive Exponentially Weighted Moving Average*) e assim por diante. Na criação dos módulos de monitoramento e análise, diferentes técnicas de predição de QoS podem ser comparadas usando ADSC, para verificar qual delas se aproxima mais do que foi realmente mensurado. Porém, nesta tese o enfoque foi no

módulo planejador, portanto, como a maioria dos trabalhos de seleção de WSs baseada em QoS, definiu-se que os valores foram gerados aleatoriamente entre os seguintes intervalos: **(i) TME:** 2 e 200 e **(ii) Custo:** 5 e 500.

Sendo assim, o **custo máximo** que um conjunto de m WSs abstratos pode assumir é $m * 500$. Dessa forma, foram definidas as seguintes faixas de orçamento, em relação ao custo máximo: **25%**, **50%** e **75%**. Por exemplo, um conjunto com 2 WS abstratos, e um orçamento de 25%, teria um orçamento de $(2 * 500) * 0,25 = 250$. Os TMEs dos WSs concretos foram normalizados entre zero e um. Sendo zero o TME mais alto e um o TME mais baixo. Isso facilita na comparação de diferentes algoritmos, bem como a visualização de quão perto a solução chegou de um determinado **ótimo global hipotético**. Por exemplo, suponha um conjunto de WSs abstratos com 2 elementos. Se o plano de composição escolher o WS concreto mais rápido para o WS_{a1} e o WS concreto mais rápido para o WS_{a2} , o TME do plano de composição será $(1 + 1) = 2$. Ele é dito hipotético, porque não há garantias que ele seja válido, pois o custo desse plano de composição pode ultrapassar o orçamento definido.

$$\begin{aligned}
 &\text{Maximizar} && \sum_{i=1}^m \sum_{j \in N_i} t_{ij} x_{ij} \\
 &\text{Sujeito a:} && \sum_{i=1}^m \sum_{j \in N_i} c_{ij} x_{ij} \leq \Omega, \\
 &&& \sum_{j \in N_i} x_{ij} = 1, \quad i = 1, \dots, m \\
 &&& x_{ij} \in \{0, 1\}, \quad i = 1, \dots, m, \quad j \in N_i
 \end{aligned} \tag{5.1}$$

A leitura desse modelo é a seguinte: considere m classes mutuamente disjuntas $N_1, \dots, N_m$ de itens para serem selecionados para uma mochila de capacidade Ω . Cada item $j \in N_i$ tem um benefício t_{ij} e um custo c_{ij} , e o problema é escolher exatamente um item de cada classe de forma que a soma do benefício seja maximizada sem exceder a capacidade Ω na soma de todos os custos. Se introduzirmos uma variável binária x_{ij} que vale um se e somente se o item j for selecionado para a classe N_i , o problema é formulado como mostra a Equação 5.1. Em termos da seleção de WSs baseada em QoS. As m classes são os WSs abstratos, os itens são os WSs concretos, o benefício é o TME normalizado, o custo é o custo para usar cada WS concreto e a capacidade é o orçamento definido. A variável t_{ij} representa o TME normalizado do WS concreto j atribuído ao WS abstrato i . A variável c_{ij} representa o custo de atribuir o WS concreto j ao WS abstrato i . A variável x_{ij} vale um se e somente se o WS concreto j for atribuído ao WS abstrato i . A restrição da segunda linha garante que exatamente um WS concreto deve ser selecionado para cada WS abstrato.

5.4 Análise

Após criado o modelo matemático, deve-se usar algoritmos/*frameworks*/bibliotecas/etc. para solucionar o modelo criado.

De fato, pode-se usar diversos algoritmos diferentes para solucionar o mesmo modelo matemático. No trabalho de (PRADO, 2012) foram implementados e avaliados dez algoritmos diferentes para a seleção de WSs baseada em atributos de QoS.

Sendo assim, optou-se por dois algoritmos: **Busca Exaustiva** (BE) e Heurística Gulosa (HG).

A BE é um algoritmo bem reconhecido nos trabalhos relacionados, sendo que seu funcionamento consiste de testar todas as combinações possíveis de variáveis, garantindo assim, o ótimo global. Ou seja, dando um conjunto de WSs abstratos, ele testa todas as combinações possíveis de WSs concretos, escolhendo o melhor plano de composição possível.

Apesar de fornecer sempre a melhor solução (ou ótimo global), ele possui complexidade assintótica **exponencial**, ou seja, se houver em média n WSs concretos para cada um dos k WSs abstratos, ele demorará $O(n^k)$ unidades de tempo para executar.

Sendo assim, a grande maioria dos trabalhos relacionados simplesmente descarta a BE e sequer chega a implementá-la. Entretanto, como foi constatado em (PRADO, 2012), para determinados tamanhos de espaço de busca, ela pode ser usada, sendo de fato, o melhor algoritmo possível.

Combinando isso ao fato de que o custo dos WSs concretos muda apenas mensalmente, optou-se por executar o algoritmo BE, visando **reduzir** o tamanho do espaço de busca percorrido quando o SDA está em execução. Ou seja, a BE testa todos os planos de composição possíveis, selecionando apenas aqueles que não ultrapassam o orçamento definido. Dessa forma, quando o SDA está em execução, apenas um subconjunto dos pontos no espaço de busca são verificados (aqueles que cumpriram a restrição de orçamento).

Ademais, sem um *deadline* de poucos segundos para executar, durante a otimização *off-line* é possível encontrar o ótimo global de todos os conjuntos de WSs abstratos. Sendo assim, sempre que o SDA for reiniciado, ele poderá reiniciar com os planos de composição que representam o ótimo global.

Como os TMEs são atualizados de hora em hora, a solução ótima muda com o tempo. Mesmo assim, economiza-se um tempo considerável, tentando encontrar um plano de composição percorrendo apenas os pontos no espaço de busca que cumpriram a restrição de orçamento.

A HG funciona de maneira diferente. Ela inicia gerando um plano de composição usando o WS concreto mais rápido para cada WS abstrato. Caso o custo total ultrapasse o orçamento, ela seleciona o segundo WS concreto mais rápido para cada WS abstrato e assim por diante. Se após percorrer os n WS concretos ela ainda não tiver encontrado um plano de composição que

Tabela 21 – Fatores e níveis escolhidos.

Fator	Níveis
Número de WSs abstratos	2, 3, 4 e 5.
Número de WSs concretos por WS abstrato	25, 50 e 100.
Porcentagens do orçamento (escolhidos/máximo)	25%, 50% e 75%.
Algoritmos	BE e HG.

cumpra o orçamento, ela simplesmente seleciona o WS concreto com o menor custo para cada WS abstrato e retorna esse plano de composição. Sendo assim, a complexidade assintótica é linear, ou seja, $O(n)$, sendo n a quantidade média de WSs concretos por WS abstrato. Apesar de parecer um algoritmo simples, conforme os experimentos executados, ela provou-se uma solução interessante.

5.4.1 Inferência

Para decidir qual a melhor solução, ou mais especificamente, em quais situações cada solução possui um desempenho mais apropriado, foi realizado um planejamento de experimentos e avaliação de desempenho.

Os fatores e níveis escolhidos são apresentados na Tabela 21. Sendo assim, foram executados um total de 72 experimentos na otimização *off-line*.

Como foi citado anteriormente, um dos objetivos da BE foi selecionar apenas os pontos no espaço de busca que atendem as restrições de orçamento. Sendo assim, quando o SDA está em execução e usa o algoritmo BE, ele percorre somente uma parcela dos pontos do espaço de busca. Em alguns casos, a proporção entre pontos no espaço de busca que cumprem a restrição e total de pontos no espaço de busca é realmente pequena (chegando a ser 1,80% em um caso). A Figura 17 exibe essas proporções quando existem dois WSs abstratos. No eixo vertical temos a proporção em porcentagem e no eixo horizontal os tamanhos de espaço de busca. O primeiro número é a quantidade de WSs abstratos, o segundo a quantidade de WSs concretos por WS abstrato e o terceiro o orçamento definido.

Observando-se a Figura 17 percebe-se que quando o orçamento foi 250, a proporção foi em média 12% do tamanho original do espaço de busca. Levando-se em conta que o SDA em execução pode executar dezenas de requisições em um curto período de tempo, tem-se uma melhora significativa em seu desempenho. Os resultados para 3 WSs abstratos são exibidos na Figura 18, para 4 WSs abstratos na Figura 19 e para 5 WSs abstratos na Figura 20.

Além disso, o algoritmo BE conseguiu encontrar o ótimo global para todos os casos definidos. A HG também conseguiu bons resultados, apesar de não garantir o ótimo global como a BE. A Figura 21 mostra a comparação entre o QoS obtido pela BE e pela HG, com o conjunto de 2 WSs abstratos. A Figura 22 mostra com o conjunto de 3 WSs abstratos, a Figura 23 com 4 WSs abstratos e a Figura 24 com 5 WSs abstratos. As colunas azuis representam a BE e as

Figura 17 – Proporção entre pontos que cumprem a restrição e total de pontos, para dois WSs abstratos.

Figura 18 – Proporção entre pontos que cumprem a restrição e total de pontos, para três WSs abstratos.

Figura 19 – Proporção entre pontos que cumprem a restrição e total de pontos, para quatro WSs abstratos.

Figura 20 – Proporção entre pontos que cumprem a restrição e total de pontos, para cinco WSs abstratos.

vermelhas representam a HG.

Figura 21 – QoS obtido pela BE e HG, com 2 WSs abstratos.

Figura 22 – QoS obtido pela BE e HG, com 3 WSs abstratos.

Em média, levando-se em conta todos os experimentos realizados, a HG obteve uma relação de HG/BE de **91%** em termos da qualidade da solução encontrada. Além disso, das 36 combinações de quantidade de WSs abstratos, quantidade de WSs concretos e orçamentos, em **21** a HG encontrou o **ótimo global**.

A Tabela 22 mostra as proporções entre o QoS dos planos de composição obtidos pelos algoritmos BE e HG, com dois WSs abstratos no fluxo. A coluna **tabela** mostra a quantidade

Figura 23 – QoS obtido pela BE e HG, com 4 WSs abstratos.

Figura 24 – QoS obtido pela BE e HG, com 5 WSs abstratos.

Tabela 22 – Resultados obtidos pelo BE e HG, com dois WSs abstratos no fluxo de composição.

Tabela	O	PEB	CBE	CHG	HG	BE	HG/BE
2_25	250	625	92	247	0,5647	1,7471	0,32
2_25	500	625	373	434	1,8000	1,8941	0,95
2_25	750	625	639	639	1,9118	1,9118	1,00
2_50	250	2.500	55	55	1,9661	1,9661	1,00
2_50	500	2.500	401	55	1,9661	1,9831	0,99
2_50	750	2.500	653	653	1,9944	1,9944	1,00
2_100	250	10.000	141	172	1,8222	1,8944	0,96
2_100	500	10.000	474	474	1,9444	1,9444	1,00
2_100	750	10.000	474	474	1,9444	1,9444	1,00

de WSs abstratos e a quantidade de WSs concretos por WS abstrato. A coluna **O** mostra o orçamento disponível. A **PEB** apresenta a quantidade total de pontos no espaço de busca, a coluna **CBE** mostra o custo total do plano selecionado pelo BE, a coluna **CHG** mostra o custo total do plano selecionado pelo HG, a coluna **HG** mostra o QoS do plano de composição obtido pelo HG, a coluna **BE** mostra o QoS do plano de composição obtido pelo BE e a coluna **HG/BE** mostra a proporção entre o QoS do HG e BE. De fato, quando os dois algoritmos obtiveram o mesmo QoS, a razão HG/BE vale um.

Observando-se a Tabela 22 nota-se que no resultado obtido para a tabela "2_25" e orçamento 250, o algoritmo HG obteve a pior proporção: 0,32. Na Tabela 23 o pior resultado ocorre na tabela "3_100" e orçamento 375, com resultado 0,58. Na Tabela 24 isso ocorre na tabela "4_25" e orçamento 500, valendo 0,68. Por fim, na Tabela 25 isso ocorre na tabela "5_100" e orçamento 625, com valor 0,28.

Sendo assim, como pode-se notar em todos esses casos com pior desempenho do algoritmo HG, o orçamento foi o menor possível (25% do custo máximo possível). Esse orçamento reduzido implica em três coisas: (i) a proporção de pontos no espaço de busca que cumprem a restrição é consideravelmente pequena, como foi mostrado nos gráficos mostrando a proporção entre pontos selecionados/total de pontos no espaço de busca, (ii) nesses casos, o algoritmo BE costuma executar dentro do *deadline* estabelecido e (iii) o QoS obtido pelo algoritmo HG é consideravelmente pior do que o obtido pelo BE.

De fato, pode-se inferir que quanto menor o orçamento, menor a proporção de pontos válidos no espaço de busca, fazendo com que o algoritmo BE seja cada vez uma opção melhor do que o algoritmo HG. De maneira análoga, quanto maior o orçamento, maior a quantidade de pontos no espaço de busca que são válidos, portanto, mais lento e inviável torna-se o algoritmo BE.

Também foi calculado a **influência dos fatores**, na degradação da solução obtida pela HG. Dessa forma, pode-se descobrir como cada fator influencia nos seus resultados. A Tabela 26 apresenta os fatores e níveis considerados, bem como a variável de resposta. A letra **A** representa

Tabela 23 – Resultados obtidos pelo BE e HG, com três WSs abstratos no fluxo de composição.

Tabela	O	PEB	CBE	CHG	HG	BE	HG/BE
3_25	375	15.625	362	281	2,4111	2,6667	0,90
3_25	750	15.625	743	509	2,7222	2,9000	0,94
3_25	1125	15.625	780	780	2,9333	2,9333	1,00
3_50	375	125.000	313	357	2,6722	2,8778	0,93
3_50	750	125.000	690	722	2,8667	2,9611	0,97
3_50	1125	125.000	988	988	2,9833	2,9833	1,00
3_100	375	1.000.000	352	306	1,6983	2,9441	0,58
3_100	750	1.000.000	542	542	2,9777	2,9777	1,00
3_100	1125	1.000.000	542	542	2,9777	2,9777	1,00

Tabela 24 – Resultados obtidos pelo BE e HG, com quatro WSs abstratos no fluxo de composição.

Tabela	O	PEB	CBE	CHG	HG	BE	HG/BE
4_25	500	390.625	486	142	2,4049	3,5337	0,68
4_25	1000	390.625	811	811	3,8708	3,8708	1,00
4_25	1500	390.625	811	811	3,8708	3,8708	1,00
4_50	500	6.250.000	488	59	2,6889	3,9222	0,69
4_50	1000	6.250.000	760	760	3,9500	3,9500	1,00
4_50	1500	6.250.000	760	760	3,9500	3,9500	1,00
4_100	500	100.000.000	491	491	3,9778	3,9778	1,00
4_100	1000	100.000.000	491	491	3,9778	3,9778	1,00
4_100	1500	100.000.000	491	491	3,9778	3,9778	1,00

Tabela 25 – Resultados obtidos pelo BE e HG, com cinco WSs abstratos no fluxo de composição.

Tabela	O	PEB	CBE	CHG	HG	BE	HG/BE
5_25	625	9.765.625	624	90	3,1508	4,5200	0,70
5_25	1250	9.765.625	1173	1169	4,3128	4,8100	0,90
5_25	1875	9.765.625	1766	1766	4,8380	4,8380	1,00
5_50	625	312.500.000	444	555	4,7303	4,8820	0,97
5_50	1250	312.500.000	1170	1192	4,9157	4,9157	1,00
5_50	1875	312.500.000	1170	1192	4,9157	4,9157	1,00
5_100	625	10 bilhões	531	610	1,3444	4,8444	0,28
5_100	1250	10 bilhões	1138	1138	4,9111	4,9111	1,00
5_100	1875	10 bilhões	1138	1138	4,9111	4,9111	1,00

o tamanho do espaço de busca. A letra **B** representa a porcentagem do orçamento selecionada e o QoS representa a diferença entre o QoS obtido pela BE e pela HG. Na Figura 25, pode-se visualizar os valores obtidos. A letra **AB** representa a interação entre os fatores **A** e **B**.

Tabela 26 – Fatores, níveis e variável de resposta.

Experimento	A - Tamanho	B - Orçamento(%)	QoS
1	2_25	25	1,182
2	2_25	75	0
3	5_100	25	3,5
4	5_100	75	0

Figura 25 – Análise da influência dos fatores.

Por meio dessa análise, percebe-se que quando o orçamento é pequeno, a qualidade da resposta obtida pela HG cai drasticamente. De fato, analisando todos os experimentos, nota-se que os dois piores resultados da relação HG/BE foram respectivamente: 5_100_625 com 0,28 e 2_25_250 com 0,32. Em ambos os casos, o orçamento selecionado foi 25%. Além disso, com essa porcentagem de orçamento, a proporção entre pontos que atendem o orçamento e pontos totais do espaço de busca é a menor de todas, como foi visualizado, por exemplo, na Figura 20, 5_25_625 em que a proporção foi a menor de todas, 1,80%. Ou seja, quanto menor o orçamento, mais provável que a BE obtenha um resultado bem melhor que a HG e em tempo hábil.

5.4.2 Julgamento

A otimização *off-line*, ajudou a otimizar o desempenho do SDA em execução de três formas: (i) encontrando o ótimo global, que deve ser usado no momento em que o SDA é inicializado; (ii) eliminando os pontos do espaço de busca que não cumprem a restrição de orçamento e (iii) fornecendo um entendimento melhor do comportamento dos algoritmos, permitindo saber em quais situações usa-los. O modelo matemático criado atende aos RFs e

Tabela 27 – Requisitos Funcionais do Estudo de Caso 2.

Requisito	Descrição
RF 2.1	Para cada WS abstrato, deve-se selecionar um WS concreto.
RF 2.2	Os atributos de QoS considerados são: custo e TME.
RF 2.3	Deve-se minimizar o TME do plano de composição.
RF 2.4	Deve-se cumprir a restrição de orçamento definida.
RF 2.5	Pode-se usar a lista de planos de composição selecionados, de acordo com o RF 1.5.
RF 2.6	Deve-se encontrar o ótimo global ou o mais próximo possível.

Tabela 28 – Requisitos Não-Funcionais do Estudo de Caso 2.

Requisito	Descrição
RNF 2.1	O TME da seleção do plano de composição não deve ultrapassar dois segundos.
RNF 2.2	O SDA deve estar apto a lidar com múltiplas requisições de maneira concorrente.
RNF 2.3	O SDA deve ser acessível pela Web.

RNFs definidos. Entretanto, tais RFs e RNFs podem mudar, como é de se esperar em qualquer software que permanece em uso por determinado tempo.

Caso isso aconteça, o modelo matemático deve ser atualizado, bem como suas soluções e planejamento de experimentos e avaliação de desempenho. Por exemplo, pode-se querer adicionar um atributo de QoS adicional a função-objetivo. Ou ainda, pode-se querer adicionar uma restrição, ou ambos.

5.5 Estudo de Caso 2: O Sistema Distribuído Autônomo em Execução

Como foi citado no capítulo do PDS2DA, a PO pode ser empregada para formalizar o "serviço"(ou conjunto de funcionalidades) que o SDA deve prover. Sendo assim, optou-se novamente por usar a PO, dessa vez no estudo de caso 2, que trata do SDA em execução.

5.5.1 Definição dos Requisitos

Novamente, o estudo de caso inicia-se com a definição dos RFs e dos RNFs. De fato, a situação é semelhante a ocorrida na otimização *off-line*. A Tabela 27 mostra os RFs e a Tabela 28 mostra os RNFs.

5.5.2 Componentes, papéis e arquitetura

Neste estudo de caso, foi definido que a arquitetura seria composta por: cliente, SGBD e servidor de aplicação. As funcionalidades de cada um são:

- **Cliente:** é a máquina que faz requisições de seleção de WSs baseada em QoS. Neste exemplo, usou-se o **Apache HttpComponents**, que segundo a própria Apache é: "um conjunto de ferramentas de componentes Java de baixo nível, focado em HTTP e protocolos associados". Em resumo, ele permite usar a linguagem Java para criar requisições usando o protocolo HTTP e seus métodos (PUT, GET, POST, etc.) (??). Usando as classes para criar aplicações *multi-thread* do Java, é possível enviar várias requisições simultaneamente ao servidor de aplicação.
- **SGBD:** é aonde estão armazenados os dados relativos aos atributos de QoS dos WSs concretos. Além disso, também estão armazenados os WSs concretos que foram selecionados pela BE na otimização *off-line*, por terem cumprido a restrição de orçamento. Foi usada a versão *open source* do MySQL.
- **Servidor de aplicação:** Para cumprir o RNF 2.3, foi selecionado o *container* de aplicações Apache Tomcat. Por meio do método GET do protocolo HTTP, é possível ao cliente solicitar a seleção dos WSs concretos. Além disso, o Apache Tomcat lida com o gerenciamento de múltiplas requisições HTTP.

A Figura 26 exhibe a arquitetura definida. Nela, as três máquinas estão conectadas por meio de um *switch gigabit Ethernet*. Apesar de todas as requisições virem de uma única máquina, nada impede de que outros testes sejam feitos, em que as requisições chegam de diversas máquinas diferentes, espalhadas pela Web.

Figura 26 – Arquitetura usada no Estudo de Caso 2.

Quando o *servlet* contendo o seletor de WSs baseado em QoS inicia no Apache Tomcat, ele chama o método *init()* que faz uma conexão com o SGBD e recupera as tabelas com os dados sobre os atributos de QoS dos WSs concretos e com os dados sobre os WSs concretos selecionados pela BE na otimização *off-line*. Após concluir isso, ele está pronto para atender as requisições do *cliente*. Ao carregar as tabelas na memória principal, é evitado que a cada requisição o *servlet* tenha que buscar os dados de QoS no SGBD, uma operação custosa do ponto de vista da *aplicação* (abrir e fechar conexões com o SGBD) e também do ponto de vista de *hardware* (ler várias vezes dados do disco rígido da máquina que contém o SGBD).

Em termos de papéis, foi definido que para o *servlet* que executa no servidor de aplicação, o desenvolvedor é: **desenvolvedor/administrador de sistemas**. Para o *cliente* o desenvolvedor também é: **desenvolvedor/administrador de sistemas**, pois deve modificar o código fonte para criar os diferentes cenários de testes que deseja. Já para o SGBD e o servidor de aplicação, é apenas **usuário do sistema**.

5.5.3 Modelagem

De fato, o "serviço" que o SDA provê quando está *online* é a seleção de WSs baseada em QoS, de maneira semelhante a realizada no estudo de caso 1. Sendo assim, o modelo matemático usado é o mesmo apresentado na Equação 5.1.

Entretanto, algumas diferenças são consideradas. Primeiro, ao invés de selecionar todos os planos de composição que cumprem o orçamento, deve-se selecionar apenas o ótimo global ou o mais próximo que se puder chegar dele. Segundo, como o SDA está em execução, de acordo com o RNF 2.1, o tempo médio de execução do algoritmo não deve ultrapassar dois segundos. Ademais, os RNFs 2.2 e 2.3 também devem ser considerados.

5.5.4 Análise

Os algoritmos usados são os mesmos apresentados no Estudo de Caso 1. A única diferença, é que no caso do algoritmo BE, ao invés de percorrer todo o espaço de busca, ele percorre um subconjunto, a saber, aquele formado pelos planos de composição selecionados durante a otimização *off-line*.

Isso permite que o algoritmo BE execute de maneira consideravelmente mais rápida, o que é uma característica crucial, devido ao *deadline* estabelecido.

5.5.5 Inferência

A comparação das diferentes soluções propostas foi feita por meio de planejamento de experimentos e avaliação de desempenho. A variável de resposta selecionada foi o tempo médio de execução do algoritmo de seleção (BE ou HG). Os fatores e níveis definidos estão na Tabela

Tabela 29 – Fatores e níveis escolhidos.

Fator	Níveis
Número de WSs abstratos	2, 3, 4 e 5.
Número de WSs concretos por WS abstrato	25, 50 e 100.
Porcentagens do orçamento (escolhidos/máximo)	25%, 50% e 75%.
Algoritmos	BE e HG.

29. O QoS obtido pelos algoritmos é são os mesmos da estudo de caso 1, pois usou-se os mesmos valores de atributos de QoS armazenados.

Primeiramente, serão analisados os experimentos que executaram o algoritmo BE. A Figura 27 apresenta os resultados obtidos pelo algoritmo BE, com um fluxo de composição contendo dois WSs abstratos. O **eixo vertical** mostra o tempo médio de execução em milissegundos, o **eixo horizontal** mostra as diferentes combinações entre tamanho de espaço de busca e orçamento definido. Por exemplo "2_25_250", significa um fluxo com dois WSs abstratos, vinte e cinco WSs concretos por WS abstrato e um orçamento total de duzentos e cinquenta.

Figura 27 – Tempo médio de execução do algoritmo BE.

As colunas representam as médias, sendo que cada experimento foi executado dez vezes. As linhas dentro das colunas representam o intervalo de confiança, com um grau de confiança de 95%. O intervalo de confiança foi explicado no capítulo sobre avaliação de desempenho.

Sendo assim, fica claro que o algoritmo BE pode ser usado, pois seu TME é de poucos milissegundos, sendo que o *deadline* permitido é de dois segundos. Ademais, ele garante o ótimo global.

A Figura 28 apresenta os resultados obtidos pelo algoritmo BE, com um fluxo de composição contendo três WSs abstratos.

Novamente, o algoritmo BE conseguiu cumprir o *deadline* determinado. Vale lembrar que isso deve-se ao fato do tamanho do espaço de busca reduzido, graças a otimização *off-line*.

Figura 28 – Tempo médio de execução do algoritmo BE.

A Figura 29 apresenta os resultados obtidos pelo algoritmo BE, com um fluxo de composição contendo quatro WSs abstratos.

Figura 29 – Tempo médio de execução do algoritmo BE.

Nota-se que as colunas referentes aos experimentos "4_100_1000" e "4_100_1500" não estão presentes no gráfico. Isso ocorre porque, mesmo percorrendo um espaço de busca reduzido, a quantidade de pontos é muito grande para que o algoritmo seja viável. No caso do experimento "4_100_1000", existe cerca de 52 milhões de planos de composição possíveis. Para o experimento "4_100_1500" existem mais de 96 milhões.

Uma série de problemas ocorreriam caso esses experimentos fossem executados. Primeiro, o tempo de **inicialização** do servidor aumentaria de maneira a torna-la inviável. Segundo, como ocorreu durante o experimento, a memória *heap* necessária seria teria sua capacidade ultrapassada e seria lançada uma exceção do tipo *JavaOutOfMemory exception*. Mesmo desconsiderando esses dois problemas (suponha-se que encontrou-se uma solução para eles) o tempo

médio de execução ultrapassaria o *deadline* definido.

A Figura 30 apresenta os resultados obtidos pelo algoritmo BE, com fluxo de composição contendo cinco WSs abstratos.

Figura 30 – Tempo médio de execução do algoritmo BE.

Neste caso, apenas os experimentos "5_25_625" e "5_25_1250" conseguiram obter um tempo de execução viável. Como foi citado, o algoritmo BE possui uma complexidade assintótica exponencial, o que torna seu uso limitado quando o tamanho do espaço de busca cresce.

O algoritmo HG, por se tratar de uma heurística, não percorre todos os pontos do espaço de busca. Isso faz com que seu TME seja consideravelmente mais rápido que o algoritmo BE, porém não garante o ótimo global. A Figura 31 mostra os resultados obtidos pelo algoritmo HG, com fluxo de composição de dois WSs abstratos.

Figura 31 – Tempo médio de execução do algoritmo HG.

Comparando com a Figura 27, percebe-se que os TMEs do HG e BE são bem parecidos nesses casos. Isso ocorre por dois motivos: (i) uma parcela desse tempo é **fixa** para todos os experimentos, é o tempo referente a rede, bem como o tempo para acessar o **servlet** via protocolo

HTTP e receber sua resposta e (ii) os tamanhos de espaço de busca são pequenos. Nesses casos, fica claro que o BE é a melhor opção, pois garante o ótimo global.

Figura 32 – Tempo médio de execução do algoritmo HG.

A Figura 32 mostra os resultados obtidos pelo HG, com fluxo de composição com 3 WSs abstratos. No experimento "3_100_1125", percebe-se a diferença entre o HG e o BE. O algoritmo BE é cerca de dez vezes mais lento que o HG. De fato, a complexidade assintótica exponencial do BE começa a ser notada.

Figura 33 – Tempo médio de execução do algoritmo HG.

A Figura 33 mostra os resultados obtidos pelo HG, com fluxo de composição de 4 WSs abstratos. Nestes casos, já existe uma diferença significativa entre os dois algoritmos. Primeiro, o BE não foi capaz de executar com os tamanhos "4_100_1000" e "4_100_1500". Já o HG, manteve seu TME em um patamar aceitável. Nos tamanhos "4_25_500", "4_25_1000", "4_25_1500" e "4_50_500" o BE variou seu TME entre 16,7 e 39,9 milissegundos. Nesses casos, por garantir o ótimo global, parece ser o melhor entre os dois algoritmos.

5.5.6 Julgamento

O modelo matemático criado atendeu aos RFs e RNFs definidos para este estudo de caso. A otimização *off-line* permitiu que o algoritmo BE percorresse um sub-espço de busca, possibilitando seu uso em situações que não seriam possíveis se ele tivesse que percorrer todo espaço de busca. Mesmo assim, em determinado momento, devido a sua complexidade assintótica exponencial, ele tornou-se uma opção inviável.

Baseando-se nos dados obtidos pelo estudo de caso 1 e estudo de caso 2, notou-se que o orçamento tem um impacto significativo no desempenho do algoritmo BE, sendo que quanto menor o orçamento, maiores as probabilidades do algoritmo BE ser uma opção viável.

Apesar de terem sido comparados dois algoritmos, nada impede que uma série de algoritmos para solução do modelo matemático sejam implementados. De fato, isso foi feito no trabalho de (PRADO, 2012), comparando o desempenho de dez algoritmos diferentes. A tendência dos trabalhos relacionados, de simplesmente descartarem o algoritmo BE devido a sua complexidade assintótica, ou ainda, de determinar um único algoritmo que seja a melhor opção em todas as situações, mostrou-se inadequada. É razoável assumir que não existe um único algoritmo que seja superior em todas as situações, devendo os esforços serem empregados em descobrir quais algoritmos são melhores em quais situações e como criar um SDA que use o algoritmo ideal para cada situação, ao invés de restringir-se a um único algoritmo.

5.6 Estudo de Caso 3: Gerenciadores Autônômicos para o Sistema Distribuído Autônômico

Neste estudo de caso, pretende-se mostrar como é possível formalizar o aspecto de **auto-otimização** do SDA usando a PO para definir o problema que o GA resolverá.

Além disso, usando a ADSC serão comparadas duas soluções diferentes para auto-otimização. Apesar de nesse estudo de caso o foco ser o módulo Planejador, do laço de controle MAPE-C do SDA, pode-se usar a ADSC para comparar diferentes soluções para: módulo de monitoramento, módulo de análise, módulo de planejamento, módulo de execução, base de conhecimento e modelo arquitetural do gerenciador autônômico.

Para implementar o aspecto de auto-otimização, foi definido que será criado um **Escalonador Autônômico de Requisições** (EAR). O objetivo do EAR é minimizar o TME das requisições recebidas, distribuindo-as entre os servidores, sem ultrapassar as capacidades individuais de cada servidor.

Dessa forma, o EAR ataca dois dos principais problemas dos SDs: **balanceamento de carga** e **QoS** (especificamente TME). Ademais, levando-se em conta que o EAR está inserido no contexto de selecionar WSs concretos para criar um WS composto, mais um problema dos SDs é atacado: a **heterogeneidade** (solucionada pelo uso de WSs).

Nesse estudo de caso, considera-se que o número de servidores, durante a execução do SDA é fixa. Se o aspecto de **auto-configuração** também fosse levado em conta, poderia-se ter um número **variável de servidores**. Isso é fácil de obter-se atualmente, com serviços de IaaS como o EC2. Apenas com essa característica adicional, mais dois dos principais problemas dos SDs poderiam ser atacados: **escalabilidade e tolerância a falhas**.

Como já foi mencionado, serviços de IaaS como o *Amazon Web Services - Elastic Compute Cloud* (EC2) possuem recursos para o **escalonamento horizontal** de recursos computacionais, como mostrado na Figura ???. Sendo assim, pode-se assumir (precipitadamente) que qualquer SD, desde que esteja sendo executado em um serviço de IaaS como o EC2, pode ser facilmente e praticamente automaticamente transformado em um SDA com os aspectos de autoconfiguração e auto-otimização.

De fato, por meio de uma aplicação denominada *Amazon CloudWatch*, os recursos computacionais presentes no serviço contratado pelo EC2 são monitorados em tempo real, com métricas definidas por cada instância de MV e também métricas que agregam os dados do sistema como um todo.

Entretanto, as métricas disponíveis são relacionadas a dados do "hardware" e não das aplicações que executam nas MVs. Algumas métricas são: **utilização da CPU, E/S de disco, E/S de rede** e assim por diante. Apesar de serem métricas úteis, elas dizem pouco ou mesmo nada sobre o comportamento das aplicações que estão em execução nas MVs. Por exemplo, métricas como: vazão, TME e disponibilidade de uma aplicação executando em uma MV, não podem ser medidas por utilização da CPU ou E/S de rede. Elas precisam ser coletadas da aplicação em si e não da MV em que ela está executando.

O *Amazon CloudWatch* possui uma API para permitir que o administrador do IaaS inclua métricas personalizadas, bem como defina gatilhos necessários para tais métricas. Sendo assim, permanece a necessidade do desenvolvedor da aplicação/administrador do IaaS definir **quais métricas coletar, como coletar, com que frequência**, quais **gatilhos** definir e assim por diante.

Existe também um serviço chamado *Application Load Balancer*, que permite que as instâncias do EC2 façam o balanceamento de carga baseando-se em métricas das aplicações que executam nas MVs.

Além disso, usando a tecnologia de *Auto-Scaling*, mais especificamente a política de escalabilidade *Dynamic Scaling*, pode-se definir diferentes gatilhos, baseando-se em diferentes métricas de monitoramento. Ainda assim, trata-se de uma **política estática** para implementar autoconfiguração e auto-otimização.

Por tanto, mesmo se todos esses recursos forem integrados: criação de métricas personalizadas, balanceamento de carga baseado nas aplicações que executam nas MVs e política de escalabilidade dinâmica, a única técnica possível de ser usada para autoconfiguração e

Tabela 30 – Requisitos Funcionais do Estudo de Caso 3.

Requisito	Descrição
RF 3.1	Cada requisição deve ser atribuída a exatamente um servidor.
RF 3.2	Deve-se minimizar o TME das requisições de cada lote.
RF 3.3	As capacidades individuais de cada servidor devem ser respeitadas.

Tabela 31 – Requisitos Não-Funcionais do Estudo de Caso 3.

Requisito	Descrição
RNF 3.1	Deve-se atualizar periodicamente o TME de cada servidor.
RNF 3.2	Deve-se atualizar periodicamente a capacidade atual de cada servidor.
RNF 3.3	Para este estudo de caso, as requisições são homogêneas.

auto-otimização são as **políticas estáticas**.

Como foi mencionado no capítulo de CA, existem diversas técnicas com uso comprovado para implementar esses aspectos, por exemplo: troca a quente, raciocínio baseado em casos, teoria de controle, aprendizado ativo, algoritmos de busca local, algoritmos genéticos, redes de filas, algoritmos de função-utilidade, redes neurais artificiais e assim por diante.

Limitar as possibilidades a apenas políticas estáticas é uma abordagem cientificamente inaceitável. Como foi citado no capítulo do PDS2DA, a ADSC deve ser usada para comparar quantitativamente diferentes soluções para o módulo planejador do SDA. Ademais, dependendo dos RFs e RNFs do SDA, pode ser simplesmente impossível implementar autoconfiguração e auto-otimização usando-se apenas políticas estáticas.

Sendo assim, segue a necessidade e utilidade do uso do PDS2DA para criar SDAs de uma forma mais completa e flexível.

5.6.1 Definição dos Requisitos

Para construir-se um SDA completo, todos os módulos do laço MAPE-C devem ser considerados. Além disso, deve-se considerar qual modelo de arquitetura do SDA será empregada: centralizada, distribuída ou hierárquica.

Dos módulos do MAPE-C, o mais importante e o que permite o uso de PO de maneira mais evidente é o **módulo Planejador**. É ele que define qual ação será tomada, para que o SDA esteja em um **estado desejável** de execução ou mude de um estado **indesejável** para um estado **desejável**.

Os RNFs 3.1 e 3.2 são de responsabilidade do módulo de **Monitoramento**. Sendo assim, eles não serão abordados neste estudo caso. Como já foi citado no Estudo de Caso 1, alguns trabalhos tem focado na estimação de valores reais de atributos de QoS, mas este não é o caso desta tese de doutorado.

O módulo de **Análise** também não foi implementado neste estudo de caso. Entretanto,

sua solução pode ser definida, baseando-se na característica de escalonamento de requisições em lote. Pode-se usar uma técnica **Reativa** para o módulo de Análise. Ou seja, sempre que um lote de requisições chegar, o EAR deve agir e fazer o escalonamento das requisições. Isso pode ser feito tanto periodicamente (a cada u unidades de tempo), ou por valores absolutos (a cada n requisições na fila) ou ainda uma combinação de ambos (a cada u unidades de tempo ou n requisições na fila, o que ocorrer primeiro).

O módulo **Planejador** foi o implementado neste estudo de caso. De fato, foram implementados dois algoritmos diferentes para o EAR, que serão explicados posteriormente.

O módulo de **Execução** não foi implementado, porém sua implementação seria trivial. Dado um plano de escalonamento definido pelo EAR, ele apenas encaminharia as requisições para cada servidor, conforme definido pelo EAR.

A **base de conhecimento** não foi implementada, porém acredita-se que um SGBD atenderia essas necessidades de maneira satisfatória.

5.6.2 Componentes, papéis e arquitetura

Neste estudo de caso, focou-se apenas no módulo planejador do laço MAPE-C. Sendo assim, esse foi o único componente implementado neste estudo de caso. Para solucionar o modelo matemático criado, foram implementados dois algoritmos, a saber: **escalonamento circular** e **escalonamento limite**.

O módulo planejador foi o escolhido, pois é nele que se insere o modelo matemático, que servirá de base para tomar as decisões de como o EAR escalonará as requisições.

5.6.3 Modelagem

A Figura 34 mostra o funcionamento do EAR. Informalmente, as requisições são do tipo "*bag-of-tasks*", ou seja, elas chegam em lote, o EAR as distribui entre os diferentes servidores, elas são finalizadas e um novo lote de requisições chega e o processo se repete, enquanto o SDA estiver em execução.

Sendo assim, tem-se n requisições e m servidores. Cada servidor tem uma capacidade máxima associada, Cap_i . Além disso, cada servidor possui um TME associado, TME_i . Alocar uma requisição a um servidor possui um custo, neste estudo de caso, o custo será o quanto aquela requisição ocupa da capacidade do servidor. Sendo assim, c_{ij} é o custo de atribuir a requisição j ao servidor i . Se houverem requisições de diferentes tamanhos, os custos associados serão diferentes. Neste estudo, definiu-se que todas as requisições são iguais, portanto, c_{ij} é sempre um.

Formalizando esse problema com PO, tem-se uma aplicação do Problema de Atribuição Generalizado, explicado no capítulo de PO. Entretanto, a formulação foi adaptada para ser um

Figura 34 – O funcionamento do Escalonador Autônomo de Requisições.

problema de minimização, levando-se em conta o TME médio das requisições atendidas. A Equação 5.2 mostra o modelo matemático criado.

$$\begin{aligned}
 &\text{Minimizar} \quad \sum_{i=1}^m \sum_{j=1}^n \frac{t_{ij}}{n} x_{ij} \\
 &\text{Sujeito a:} \quad \sum_{j=1}^n c_{ij} x_{ij} \leq \Gamma_i, \quad i \in M = \{1, \dots, m\}, \\
 &\quad \sum_{i=1}^m x_{ij} = 1, \quad j \in N = \{1, \dots, n\}, \\
 &\quad x_{ij} \in \{0, 1\}, \quad i \in M, j \in N
 \end{aligned} \tag{5.2}$$

De maneira formal, considere-se n requisições e m servidores, com:

- t_{ij} = TME da requisição j associada ao servidor i .
- c_{ij} = custo da requisição j associada ao servidor i .
- cap_i = capacidade do servidor i .

Atribua uma requisição a exatamente um servidor, de maneira que o TME médio das requisições seja minimizado, não ultrapassando a capacidade individual de cada servidor. A

variável binária x_{ij} vale um se e somente se a requisição j for atribuída ao servidor i , caso contrário, seu valor é zero.

5.6.4 Análise

Neste estudo de caso, focou-se no módulo Planejador do laço de controle MAPE-C, pois é ele que decide como as requisições serão distribuídas entre os servidores. Em um SDA real, entretanto, todos os módulos do laço MAPE-C devem ser implementados.

De fato, como já foi mencionado anteriormente, pode-se usar a ADSC para avaliar diferentes soluções para cada módulo do MAPE-C e também para base de conhecimento e a arquitetura do GA.

Foram criadas duas **políticas estáticas**, são elas: **escalonamento circular** e **escalonamento limite**.

No **escalonamento circular**, cada requisição é atribuída a um servidor e o processo se repete até que todas as requisições tenham sido atribuídas. Por exemplo, suponha quatro requisições e dois servidores. A requisição um é atribuída ao servidor um, a requisição dois é atribuída ao servidor dois, a requisição três é atribuída ao servidor um e a requisição quatro é atribuída ao servidor dois.

No **escalonamento limite**, os servidores são ordenados de acordo com seu TME. Sendo assim, o servidor mais rápido recebe requisições até que sua capacidade seja ocupada. Após isso, o segundo servidor mais rápido recebe requisições até o fim de sua capacidade e assim por diante.

5.6.5 Inferência

Neste estudo de caso, variou-se a quantidade de servidores, a capacidade dos servidores, o número de requisições e o TME dos servidores. Sendo assim, no total, foram executados 48 experimentos. A sigla "**E**" representa o número do experimento, a sigla "**NS**" representa o número de servidores, a sigla "**NR**" representa o número de requisições, a sigla "**C_i**" representa a capacidade do servidor i e a sigla "**T_i**" representa o TME do servidor i . A **variável de resposta** em todos os experimentos é o tempo médio das requisições em segundos. As colunas **azuis** representam a política **escalonamento circular** e as **vermelhas** a política **escalonamento limite**. Por exemplo, na Figura 35, a primeira coluna vermelha representa o experimento 9. Nele, usando a política escalonamento limite, obteve-se um tempo médio de execução das requisições de 1,5 segundo. Isso ocorreu porque, o número de requisições foi 200, a capacidade de cada servidor também era 200. Sendo assim, alocou-se as 200 requisições para o servidor mais rápido, com TME de 1,5 segundo. A Figura 35 apresenta os resultados dos experimentos 1 a 16. A Figura 36 mostra os resultados dos experimentos 17 a 32. A Figura 37 os resultados dos experimentos 33 a

Tabela 32 – Fatores e níveis escolhidos. Política escolhida: **escalonamento circular**.

E	NS	NR	C1	C2	C3	C4	T1	T2	T3	T4
1	2	200	200	200	0	0	1,5	2,5	0	0
2	2	400	200	200	0	0	1,5	2,5	0	0
3	2	200	100	300	0	0	1,5	2,5	0	0
4	2	400	100	300	0	0	1,5	2,5	0	0
5	4	200	100	100	100	100	1,5	2,5	2,4	0,9
6	4	400	100	100	100	100	1,5	2,5	2,4	0,9
7	4	200	180	220	100	100	1,5	2,5	2,4	0,9
8	4	400	180	220	100	100	1,5	2,5	2,4	0,9

Tabela 33 – Fatores e níveis escolhidos. Política escolhida: **escalonamento limite**.

E	NS	NR	C1	C2	C3	C4	T1	T2	T3	T4
9	2	200	200	200	0	0	1,5	2,5	0	0
10	2	400	200	200	0	0	1,5	2,5	0	0
11	2	200	100	300	0	0	1,5	2,5	0	0
12	2	400	100	300	0	0	1,5	2,5	0	0
13	4	200	100	100	100	100	1,5	2,5	2,4	0,9
14	4	400	100	100	100	100	1,5	2,5	2,4	0,9
15	4	200	180	220	100	100	1,5	2,5	2,4	0,9
16	4	400	180	220	100	100	1,5	2,5	2,4	0,9

48. O **eixo vertical** representa o tempo médio das requisições em segundos. O **eixo horizontal** representa o número de servidores e o número de requisições.

Figura 35 – Gráfico dos experimentos 1 ao 16.

Tabela 34 – Fatores e níveis escolhidos. Política escolhida: **escalonamento circular**.

E	NS	NR	C1	C2	C3	C4	T1	T2	T3	T4
17	2	200	200	200	0	0	2,5	2,8	0	0
18	2	400	200	200	0	0	2,5	2,8	0	0
19	2	200	100	300	0	0	2,5	2,8	0	0
20	2	400	100	300	0	0	2,5	2,8	0	0
21	4	200	100	100	100	100	2,5	2,8	3	1,1
22	4	400	100	100	100	100	2,5	2,8	3	1,1
23	4	200	180	220	100	100	2,5	2,8	3	1,1
24	4	400	180	220	100	100	2,5	2,8	3	1,1

Tabela 35 – Fatores e níveis escolhidos. Política escolhida: **escalonamento limite**.

E	NS	NR	C1	C2	C3	C4	T1	T2	T3	T4
25	2	200	200	200	0	0	2,5	2,8	0	0
26	2	400	200	200	0	0	2,5	2,8	0	0
27	2	200	100	300	0	0	2,5	2,8	0	0
28	2	400	100	300	0	0	2,5	2,8	0	0
29	4	200	100	100	100	100	2,5	2,8	3	1,1
30	4	400	100	100	100	100	2,5	2,8	3	1,1
31	4	200	180	220	100	100	2,5	2,8	3	1,1
32	4	400	180	220	100	100	2,5	2,8	3	1,1

Figura 36 – Gráfico dos experimentos 17 ao 32.

Tabela 36 – Fatores e níveis escolhidos. Política escolhida: **escalonamento circular**.

E	NS	NR	C1	C2	C3	C4	T1	T2	T3	T4
33	2	200	200	200	0	0	2,5	4	0	0
34	2	400	200	200	0	0	2,5	4	0	0
35	2	200	100	300	0	0	2,5	4	0	0
36	2	400	100	300	0	0	2,5	4	0	0
37	4	200	100	100	100	100	2,5	4	2,7	3
38	4	400	100	100	100	100	2,5	4	2,7	3
39	4	200	180	220	100	100	2,5	4	2,7	3
40	4	400	180	220	100	100	2,5	4	2,7	3

Tabela 37 – Fatores e níveis escolhidos. Política escolhida: **escalonamento limite**.

E	NS	NR	C1	C2	C3	C4	T1	T2	T3	T4
41	2	200	200	200	0	0	2,5	4	0	0
42	2	400	200	200	0	0	2,5	4	0	0
43	2	200	100	300	0	0	2,5	4	0	0
44	2	400	100	300	0	0	2,5	4	0	0
45	4	200	100	100	100	100	2,5	4	2,7	3
46	4	400	100	100	100	100	2,5	4	2,7	3
47	4	200	180	220	100	100	2,5	4	2,7	3
48	4	400	180	220	100	100	2,5	4	2,7	3

Figura 37 – Gráfico dos experimentos 33 ao 48.

5.6.6 Julgamento

Neste estudo de caso, compararam-se diferentes PEs para o EAR. Ele focou-se no módulo **Planejador** do MAPE-C, pois é nele que a PO pode ser aplicada, visando criar um modelo matemático para implementar o aspecto de **auto-otimização**. Entretanto, como já foi citado em capítulos anteriores, uma vasta gama de diferentes técnicas podem ser empregadas para implementar o módulo planejador.

A ADSC pode e deve ser usada para comparar o desempenho de diferentes implementações do módulo planejador. Em um SDA real, alguns aspectos adicionais precisam ser considerados. Por exemplo, qual seria a **política de descarte** de requisições, caso o número de requisições for maior do que a capacidade somada dos servidores.

Ademais, diversas técnicas de cálculo do t_{ij} podem ser empregadas, pelo módulo de análise. Essas técnicas podem ser comparadas usando ADSC.

Diferentes modelos de arquitetura do GA podem ser implementados e comparados com a ADSC. Em um SDA real, pode-se realizar um conjunto de planejamento de experimentos, avaliando cada módulo do MAPE-C isoladamente, e depois um planejamento de experimentos integrando os diferentes módulos, vendo como ocorre a interação entre eles.

Enfim, cada SDA possui suas necessidades específicas e não existe conjunto de RFs e RNFs, ou modelo matemático, que sirva para todos os SDAs existentes. Por fim, vale lembrar que o PU é baseado em componentes e iterativo. Ou seja, sempre que houver uma mudança nos RFs ou RNFs, ou ainda, algum avanço tecnológico (por exemplo, um novo SGBD) ou avanço científico (um algoritmo mais eficiente), o SDA pode ter seus componentes modificados para incorporar tais mudanças, sem grande impacto nos demais componentes do SDA.

5.7 Considerações finais

Os estudos de caso apresentados, têm basicamente dois objetivos: **(i)** servir como uma prova de conceito para o uso do PDS2DA e **(ii)** servir como um guia de uso para o desenvolvimento de SDAs usando o PDS2DA.

Entretanto, é necessário ter em mente que nem todas as possibilidades fornecidas pelo PDS2DA foram apresentadas e abordadas nos estudos de caso. É preciso fazer uma ligação entre as propostas do PDS2DA e o que realmente foi concretizado nos estudos de caso.

No primeiro estudo de caso, foram realizadas duas tarefas importantes: **(i)** definição de uma arquitetura base, que por acaso é a solução ótima (poderia não ser se não houvesse tempo para encontra-la) e **(ii)** redução do tamanho do espaço de busca, percorrido pelo SDA em execução.

Como foi mencionado no capítulo do PDS2DA e na revisão da literatura na seção de

ADSC, existem várias metodologias diferentes de ADSC. Elas podem ser empregadas em vários SDAs para otimização *off-line*, por exemplo: (i) *benchmarks*; (ii) análise do estado das *threads*; (iii) análise da complexidade assintótica dos algoritmos; (iv) listas de checagem padrão; (v) análise de influência dos fatores e assim por diante. No livro de (GREGG, 2014) são apresentadas **dezenas** de metodologias de ADSC.

No segundo estudo de caso, foi implementado o serviço prestado pelo SDA, usando-se a PO para sua formalização e ADSC para validar os resultados dos algoritmos propostos de maneira quantitativa. Entretanto, considerou-se uma carga de trabalho padrão e dois algoritmos para solução do problema.

Em (PRADO, 2012) foram implementados e avaliados usando-se ADSC dez algoritmos para a seleção de *Web services* baseada em QoS. Sendo assim, não apenas dois mas qualquer número de algoritmos podem ser comparados usando ADSC. Além disso, metodologias de ADSC podem ser empregadas para modelar diferentes cargas de trabalho baseando-se em cargas reais ou mesmo usando registros de requisições anteriores.

Por fim, pode-se usar a ADSC para avaliar diferentes modelos de arquitetura para o serviço provido pelo SDA (duas camadas, três camadas, etc.).

No terceiro estudo de caso, foi implementado o módulo planejador de um gerenciador autônomo, pois trata-se do mais importante e mais amplamente citado nos trabalhos relacionados. Usando a PO e ADSC novamente, foi possível formalizar e avaliar quantitativamente os resultados das duas políticas de escalonamento definidas.

Entretanto, todos os módulos presentes no MAPE-C podem ser implementados e avaliados, seguindo os mais diversos critérios que forem necessários. Ademais, diferentes arquiteturas para o laço de controle MAPE-C podem ser implementadas e avaliadas.

Resumindo, os estudos de caso funcionam como um guia inicial para as possibilidades do PDS2DA e não como um limite de suas aplicações.

CONCLUSÕES E TRABALHOS FUTUROS

Este capítulo tem o intuito de apresentar as conclusões obtidas com o desenvolvimento desta tese, considerando os resultados obtidos a partir dos três estudos de caso propostos e avaliados, discutir as dificuldades que surgiram no decorrer do projeto, enfatizar as contribuições obtidas e apresentar sugestões para trabalhos futuros.

6.1 Conclusões

Como foi citado anteriormente, os SDs estão ficando cada vez mais complexos, devido a diversos fatores, tais como: **(i)** crescente número de parâmetros e módulos carregáveis das aplicações distribuídas; **(ii)** crescente heterogeneidade de sistemas presentes dentro de uma mesma empresa e **(iii)** aumento do uso de aplicações de terceiros (como por exemplo, Web services).

Além disso, os SDs atuais possuem cláusulas de SLA, exigindo alguns valores específicos para atributos de QoS que devem ser cumpridos. O não cumprimento dessas cláusulas, ocasiona multas e até mesmo a quebra de contratos. É fácil perceber, que a consequência de não garantir um bom QoS são prejuízos financeiros e até mesmo a reputação da empresa em questão.

A CA surge como uma forma de solucionar os problemas clássicos dos SDs, como por exemplo: escalabilidade, balanceamento de carga, QoS, tratamento de falhas e assim por diante. Esses são problemas inerentes a qualquer SD. A CA propõe o auto-gerenciamento, por meio de seus quatro aspectos fundamentais: autoconfiguração, auto-otimização, autocura e autoproteção. O laço de controle retro-alimentado MAPE-C já foi usado com sucesso em dezenas de trabalhos relacionados.

Sendo assim, a CA tem sido empregada com sucesso, para solucionar os mais variados problemas de SDs, em uma série de sistemas diferentes, desde sistemas embarcados até ambientes de computação em nuvem. Resumidamente, os trabalhos relacionados iniciam fornecendo uma

visão geral do problema que desejam solucionar (por exemplo, proteger um servidor Web de ataques de negação de serviço). Após isso, é feita uma breve revisão sobre as técnicas atuais, para depois mostrar a solução proposta (nesse exemplo, usando algum algoritmo para implementar autoproteção). Por fim, é feita uma comparação com as técnicas atuais, mostrando quais os benefícios da nova técnica proposta.

Esse roteiro tem sido usado pela maioria dos autores, para os trabalhos que implementam um ou mais dos aspectos da CA. Essa abordagem é muito útil, auxiliando os demais cientistas nos seguintes pontos: (i) mostrando vários algoritmos e técnicas que podem ser usados para implementar cada um dos aspectos da CA e (ii) servindo de exemplo de como integrar a CA a um SD convencional.

De fato, o módulo do MAPE-C mais citado e que recebe a maior atenção, tem sido o módulo planejador. É ele que toma as decisões, de como agir e o que deve ser feito, quando alguma ação for necessária. Porém, é importante frisar que a criação do módulo planejador de um SDA é apenas uma fração de todo o processo de desenvolvimento de um *software*. Sendo assim, deve-se levar em conta todas as fases, desde a definição dos requisitos até a posterior manutenção do *software*.

Entretanto, não foi encontrado nenhum processo de desenvolvimento de *software* focado em sistemas distribuídos autônomicos. O laço de controle MAPE-C, fornece apenas um modelo teórico de arquitetura, dividindo os componentes do sistema relacionados a CA, nos seus respectivos módulos: monitoramento, análise, planejamento, execução e base de conhecimento. Nota-se que entre os RFs e RNFs definidos, até chegar a um SDA realmente implementado, com todos os módulos do MAPE-C, há uma lacuna muito grande.

Sendo assim, a ideia desta tese foi propor um processo de desenvolvimento de *software* focado em sistemas distribuídos autônomicos (PDS2DA). Primeiramente, foram analisadas quais áreas de conhecimento poderiam ser integradas, visando a criação de um processo sistemático, flexível e que permitisse que o SDA fosse modificado com facilidade. Dessa forma, iniciou-se com o PU, que tem sido amplamente usado na indústria e na academia por décadas. Ele possui as vantagens de ser iterativo e incremental, baseado na arquitetura (algo essencial para um SD), flexível e baseado em componentes. O fato de ser baseado em componentes, faz com que cada componente do SDA que será desenvolvido possa ser modificado (sempre que necessário) sem acarretar um impacto significativo nos demais componentes.

Os SDs e a CA são outras áreas do conhecimento que fazem parte do PDS2DA, por razões óbvias. A ideia básica é que o SD realiza as funcionalidades necessárias e a CA entra como uma forma de auto-gerenciamento, transformando esse SD em um SDA. Sendo assim, um servidor Web autônomico continua atendendo requisições HTTP, porém possui um módulo relacionado a CA, que pode cuidar por exemplo da proteção contra ataques de negação de serviço usando o aspecto de autoproteção. Conclui-se então, que é necessário ter conhecimento tanto do SD, quanto da CA em si.

As duas outras áreas de conhecimento que foram integradas ao PSD2DA são: pesquisa operacional e avaliação de desempenho de sistemas computacionais. De fato, a avaliação de desempenho pode ser empregada em todas as fases do desenvolvimento do SDA em questão.

Como foi mencionado anteriormente, a PO pode ser usada em três ocasiões diferentes: (i) para formalizar e solucionar a otimização *off-line* do SDA (estudo de caso 1); (ii) para implementar as funcionalidades ou um subconjunto das funcionalidades do SDA (estudo de caso 2) e (iii) para prover o auto-gerenciamento, por meio de algum aspecto da CA, implementando os módulos do MAPE-C (estudo de caso 3). De fato, a PO tem sido usada em alguns trabalhos nos casos (ii) e (iii), porém não foi encontrado nenhum trabalho que tenha usado-a nas três situações citadas e de maneira integrada.

A ADSC pode ser usada em todas as etapas do desenvolvimento do SDA, para comparar, de maneira sistemática, quantitativa e com significância estatística, as mais diversas soluções para cada componente do SDA. Na otimização *off-line*, ela pode ser usada individualmente para comparar os componentes (por exemplo, ver qual SGBD possui menor TME) ou em conjunto com a PO (estudo de caso 1). Durante a criação do SDA, também pode ser usada isoladamente ou em conjunto com a PO, para avaliar diferentes soluções para as funcionalidades que o SDA deve possuir (estudo de caso 2). Ou ainda, para avaliar diferentes soluções para o auto-gerenciamento, podendo comparar diferentes arquiteturas para o gerenciador autônomo (centralizada, distribuída e hierárquica) ou diferentes algoritmos para implementar qualquer um dos módulos do MAPE-C (estudo de caso 3).

Outro aspecto importante que deve ser considerado, é o emprego da ADSC para realizar *benchmarks* e testar os possíveis gargalos e a escalabilidade do SDA. Como foi citado no capítulo de ADSC, as técnicas mais tradicionais, como o planejamento de capacidade estão em crescente desuso. Isso se deve a diversos fatores. Primeiro, o paradigma onde as empresas **necessitavam** fazer planejamento de capacidade, para que posteriormente comprassem *clusters* com grande capacidade computacional, visando a escalabilidade vertical, acabou. Com a computação em nuvem, as empresas podem adquirir maior poder computacional quase que de imediato e sem maiores burocracias, é a chamada escalabilidade horizontal. Segundo, cada vez mais os SDs possuem componentes de terceiros, que não fornecem os dados necessários para uma correta modelagem do sistema. Por exemplo, taxa de chegada de requisições, TME, vazão e assim por diante, são dados não disponíveis, quando o componente em questão é de responsabilidade de terceiros (por exemplo, um Web service que verifica o CPF de um cliente, para possíveis dívidas, fornecido por uma entidade externa). Terceiro, os SDAs não possuem uma arquitetura bem definida e estática. Pelo contrário, a ideia é justamente que o SDA seja modificado, sempre que necessário. Fazer uma modelagem que preveja todas as possíveis combinações de componentes e o comportamento do SDA é deverás complexo. Quarto, no ambiente de computação em nuvem, é muito difícil prever o desempenho que o SDA terá. Isso ocorre pela própria natureza dos serviços prestados em nuvem. O próprio conceito de vCPU varia de acordo com a empresa que

está fornecendo o serviço e com a quantidade de usuários atuais. Por fim, os custos envolvidos em alugar temporariamente uma infraestrutura de nuvem maior, para que os benchmarks sejam realizados, é pequena em relação ao tempo, esforço e mão-de-obra necessária para modelagem do sistema usando planejamento de capacidade.

Apesar da CA ter sido empregada há mais de uma década, tanto na indústria como no meio acadêmico, fica claro que os desenvolvedores de aplicação ainda não tem pleno conhecimento de como emprega-la, como avaliar quantitativamente as diferentes soluções e como é sua relação com as demais fases e fluxos de trabalho do desenvolvimento de *software*. Como foi citado no capítulo de trabalhos relacionados, diversos artigos propõe algum algoritmo, técnica ou modelo de arquitetura, porém não executam nenhuma forma de validação. Alguns, constroem protótipos, provando sua viabilidade, mas sem realizar nenhuma forma de avaliação de desempenho. Tendo em vista a crescente complexidade dos SDs, é necessário ter em mente não apenas suas funcionalidades (ou RFs), mas também seus requisitos não funcionais (ou RNFs, ou atributos de QoS). A ADSC pode e deve ser empregada para garantir o cumprimento desses atributos de QoS, nas mais variadas situações.

6.2 Dificuldades relacionadas ao projeto

Primeiramente, esta tese envolveu uma série de áreas de conhecimento ortogonais: PU, SD, CA, PO e ADSC. Sendo assim, a primeira dificuldade foi descobrir quais áreas de conhecimento seriam úteis para o desenvolvimento de um SDA. Os SDs e a CA são os componentes óbvios do PDS2DA. Entretanto, quando considera-se os algoritmos que podem ser usados para implementar a CA, muitos trabalhos focam na área de inteligência artificial.

Entretanto, a PO possui uma série de outros algoritmos que já são amplamente usados em combinação com SDs. Além disso, formalizar um problema usando a PO não impede que algoritmos de IA sejam usados para solucioná-lo. Na dissertação de mestrado do autor, (PRADO, 2012), o problema de seleção de Web services baseado em QoS (problema da mochila) teve alguns algoritmos genéticos implementados. Ou seja, uma vez formalizado o problema usando a PO, pode-se usar algoritmos clássicos de PO, algoritmos heurísticos ou alguma técnica da IA. Além disso, como já foi mencionado, a PO pode ser usado em diferentes fases do desenvolvimento do SDA, sendo assim, sua incorporação ao PDS2DA trouxe vários benefícios.

Uma dificuldade foi não aprofundar demais em cada uma das áreas, mostrando apenas o necessário para que seja possível compreender o papel que cada uma possui no PDS2DA. Para prova de conceito e uma forma de guia de uso, foram implementados os três estudos de caso.

Por fim, mostrar porque é interessante ter um processo focado em SDAs e como ele contribui com as diversas tarefas envolvidas na criação de um SDA, foi provavelmente o desafio mais interessante.

6.3 Contribuições

A primeira contribuição relevante é a revisão e categorização dos trabalhos relacionados. A partir dela, foi possível identificar algumas lacunas nos trabalhos relacionados, além de se ter um panorama geral do foco dos trabalhos, de quais aspectos da CA tem sido mais frequentemente implementados, as formas de validação usadas, a quantidade de trabalhos que visam implementar uma solução de CA para um SDA específico, quantos propõe diferentes modelos de arquitetura para o SDA, quantos propõe *frameworks* que auxiliam na criação de SDAs e assim por diante. Mais importante, notou-se a falta de um processo de desenvolvimento de *software* focado em SDAs.

A segunda contribuição é a compilação dos problemas inerentes aos SDs e os respectivos aspectos da CA que podem ser usados para solucioná-los. Apesar de parecer trivial relacionar os problemas dos SDs aos aspectos da CA, dentre as centenas de trabalhos relacionados estudados, nenhum fez esse mapeamento. Isso serve como um ponto de partida e um conhecimento interessante, para aqueles que desejam criar um novo SDA ou converter um SD em um SDA. Além disso, foi feita uma compilação das diversas técnicas usadas para implementar cada um dos aspectos da CA.

A terceira e mais significativa contribuição consiste na criação do PDS2DA. Com ele, uniu-se o conhecimento do PU, SD, CA, PO e ADSC, criando um processo de desenvolvimento de *software* contemplando desde a definição dos requisitos até a manutenção do SDA criado. Ao invés de focar em um SDA específico, o PDS2DA pode ser usado para construir uma vasta gama de SDAs diferentes, podendo ser usado para implementar qualquer um dos quatro aspectos da CA. Por não se tratar de um *framework*, não está limitado a uma determinada linguagem de programação ou modelo de computação distribuída. O PDS2DA pode ser usado tanto para criar um SDA do princípio, como para converter um SD em um SDA. Ademais, até mesmo um SDA já existente, pode usar os conceitos de PO e ADSC, visando criar novas soluções e melhorar o seu desempenho. Por fim, o fato de ser baseado em componentes, permite que ele seja ideal para um ambiente em constante mudanças, seja em tempo de execução (ex. carga de trabalho) ou mudança dos requisitos (funcionais ou não-funcionais).

A quarta contribuição é a criação do algoritmo HG para a seleção de Web services baseada em QoS. Como foi mostrado, em muitos casos ele obtém o ótimo global e na maioria dos casos chega bem próximo dele. São poucos os casos em que ele apresenta um desempenho inferior. Por coincidência, esses são justamente os casos onde é possível usar o algoritmo BE.

A quinta contribuição foi a **redução do espaço de busca**, efetuada no estudo de caso 1. Apesar de ter sido usado especificamente para o problema de seleção de Web services baseada em QoS, essa redução pode ser usada em outros problemas de otimização combinatória, sempre que um ou mais dos atributos de QoS relacionados possuir uma taxa de atualização muito superior do que os demais.

Por fim, todos os estudos de caso serviram como prova de conceito e um guia de como usar o PDS2DA. Entretanto, como já foi mencionado, cada SDA possui suas próprias necessidades específicas. Sendo assim, a quantidade de diagramas UML criados, a criação de pseudocódigos e assim por diante, dependerá do SDA, da equipe envolvida, do cronograma e assim por diante.

6.4 Trabalhos Futuros

Para os trabalhos futuros, existem diversas possibilidades a serem exploradas. Primeiramente, outros aspectos da CA podem ser abordados. Ademais, os SDAs podem ser criados e implantados em diferentes modelos de computação distribuída: cluster, nuvem e assim por diante.

No estudo de caso 1, foi usada a PO em combinação com a ADSC para realização da otimização *off-line*. Entretanto, como já foi mencionado, a ADSC também pode ser usada sozinha, para avaliar o desempenho de diferentes componentes do SDA em questão (SGBD, Servidor Web, etc.). Além disso, existem dezenas de metodologias de avaliação de desempenho diferentes, todas podendo ser empregadas para melhorar o desempenho do SDA em questão de maneira *off-line*.

No estudo de caso 2, dois algoritmos foram implementados para seleção de Web services baseada em QoS. Entretanto, o número de soluções implementadas para um determinado problema, pode ser bem maior. Na dissertação de mestrado do autor, foram implementados dez algoritmos para solucionar esse problema. Sendo assim, diversas arquiteturas, algoritmos e técnicas podem ser avaliados, dentro das possibilidades dos papéis definidos no PDS2DA.

No estudo de caso 3, o módulo planejador foi abordado. Entretanto, todos os módulos podem ser implementados e avaliados, além da própria arquitetura do SDA. Sendo assim, realizar estudos de caso enfocando todos os módulos do MAPE-C, além da arquitetura, de preferência em um ambiente real de nuvem, seria um trabalho futuro deverás interessante.

6.5 Publicações Oriundas do Trabalho

É interessante citar quais são as publicações oriundas do trabalho, bem como as publicações já submetidas e as que ainda serão submetidas.

O primeiro artigo publicado, refere-se a alguns resultados preliminares do projeto, em uma conferência internacional com o título "Cobapas: Combinatorial Optimization based Approach for Autonomic Systems". Além disso, um artigo foi submetido para a conferência nacional WPerformance, ainda aguardando o resultado da avaliação.

Também foi enviado um artigo para a revista IEEE América Latina (em português),

aonde espera-se que o trabalho seja aceito.

Um artigo contendo uma versão mais detalhada e categorizada dos trabalhos relacionados citados, será enviado a uma revista internacional que aceite artigos de revisão da literatura.

Posteriormente, pretende-se enviar um artigo em inglês para a revista *Plos One*, contendo todos os resultados e contribuições presentes nesta tese.

Por fim, um livro será publicado, com uma versão mais didática do PDS2DA, com o intuito de que ele possa ser usado inicialmente por estudantes e eventualmente pela indústria.

REFERÊNCIAS

- AHUJA, K.; DANGEY, H. Autonomic computing: An emerging perspective and issues. In: . [S.l.: s.n.], 2014. Citado na página 74.
- AL-SHISHTAWY, A.; HOGLUND, J.; POPOV, K.; PARLAVANTZAS, N.; VLASSOV, V.; BRAND, P. Distributed control loop patterns for managing distributed applications. In: **Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops**. [S.l.: s.n.], 2008. p. 260 – 264. Citado nas páginas 46 e 71.
- ALOMARI, F. B.; MENASCE, D. A. Self-protecting and self-optimizing database systems: Implementation and experimental evaluation. In: **ACM Cloud and Autonomic Computing Conference**. [S.l.: s.n.], 2013. p. 1 – 10. Citado nas páginas 27, 63 e 69.
- ANGLANO, C.; MONTANI, S. Achieving self-healing in autonomic software systems: a case-based reasoning approach. In: **International Conference on Intelligent Computing**. [S.l.: s.n.], 2007. p. 1–19. Citado na página 68.
- ARDAGNA, D.; PANICUCCI, B.; TRUBIAN, M.; ZHANG, L. Energy-aware autonomic resource allocation in multitier virtualized environments. **IEEE Transactions on Services Computing**, v. 5, n. 1, p. 2 – 19, 2012. Citado nas páginas 27 e 63.
- ARDAGNA, D.; PERNICI, B. Global and local qos guarantee in web service selection. In: **IEEE International Conference on Web services**. [S.l.: s.n.], 2005. p. 32 – 46. Citado nas páginas 42 e 96.
- ARENALES, M.; ARMENTANO, V.; MORABITO, R.; YANASSE, H. **Pesquisa operacional para cursos de engenharia**. [S.l.]: Campus - Elsevier, 2007. Citado nas páginas 50, 51 e 52.
- AWAD, M.; MENASCE, D. A. Dynamic derivation of analytical performance models in autonomic computing environments. In: **Computer Measurement Group Conference**. [S.l.: s.n.], 2014. p. 1 – 11. Citado na página 67.
- BERTOLINO, A.; CALABRO, A.; ANGELIS, G. D. Adaptive sla monitoring of service choreographies enacted on the cloud. In: **IEEE International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems**. [S.l.: s.n.], 2013. p. 92 – 101. Citado na página 73.
- BHAKTI, M. A. C.; ABDULLAH, A. B.; JUNG, L. T. Autonomic, self-organizing services-oriented architecture in service ecosystem. In: **IEEE International Conference on Digital Ecosystems and Technologies**. [S.l.: s.n.], 2010. Citado nas páginas 27 e 71.
- BORRO, L. C. **Escalonamento em grades moveis: uma abordagem ciente do consumo de energia**. Dissertação (Mestrado) — USP, 2014. Citado na página 65.
- BRAZIER, F. M.; KEPHART, J. O.; PARUNAK, H. V. D.; HUHNS, M. N. **Agents and Service-Oriented Computing for Autonomic Computing: A research agenda**. 2009. IEEE Internet Computing. Citado na página 27.

- BUYYA, R.; CALHEIROS, R. N.; LI, X. Autonomic cloud computing: Open challenges and architectural elements. In: . [S.l.: s.n.], 2012. p. 8. Citado nas páginas 39 e 73.
- CALINESCU, R.; GRUNSKE, L.; KWIATKOWSKA, M.; MIRANDOLA, R.; TAMBURRELLI, G. Dynamic qos management and optimization in service-based systems. **IEEE Transactions on Software Engineering**, v. 37, n. 3, p. 387 – 409, 2011. Citado na página 71.
- CANFORA, G.; ESPOSITO, R. A lightweight approach for qos-aware service composition. In: **International Conference on Service Oriented Computing (ICSOC)**. [S.l.: s.n.], 2004. Citado na página 96.
- CANFORA, G.; PENTA, M. D.; ESPOSITO, R.; VILLANI, M. L. An approach for qos-aware service composition based on genetic algorithms. In: **ACM Genetic and Evolutionary Computation Conference**. [S.l.: s.n.], 2005. p. 1069–1075. Citado nas páginas 27, 42, 64 e 96.
- CASALICCHIO, E.; SILVESTRI, L. Mechanisms for sla provisioning in cloud-based service providers. p. 795 – 810, 2012. Citado na página 96.
- CHAARI, T.; FAKHFAKH, K. Semantic modeling and reasoning at runtime for autonomous systems engineering. In: **9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing**. [S.l.: s.n.], 2012. p. 415 – 422. Citado na página 69.
- CHECIU, L.; SOLOMON, B.; IONESCU, D.; LITOU, M.; ISZLAI, G. Observability and controllability of autonomic computing systems for composed web services. In: **IEEE International Symposium on Applied Computational Intelligence and Informatics**. [S.l.: s.n.], 2011. p. 269 – 274. Citado nas páginas 27 e 64.
- CHEN, B.; PENG, X.; YU, Y.; ZHAO, W. Requirements-driven self-optimization of composite services using feedback control. **IEEE Transactions on Services Computing**, v. 8, n. 1, p. 107 – 120, 2015. Citado na página 64.
- CHEN, Q.; ABDELWAHED, S.; ERRADI, A. A model-based validated autonomic approach to self-protect computing systems. **IEEE INTERNET OF THINGS JOURNAL**, v. 1, n. 5, p. 446 – 460, 2014. Citado na página 69.
- CHEN, W.; QIAO, X.; WEI, J.; HUANG, T. A two-level virtual machine self-reconfiguration mechanism for the cloud computing platforms. In: **9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing**. [S.l.: s.n.], 2012. Citado na página 65.
- CLAUDEL, B.; PALMA, N. D.; LACHAIZE, R.; HAGIMONT, D. Self-protection for distributed component-based applications. In: **International Symposium on Stabilization, Safety, and Security of Distributed Systems**. [S.l.: s.n.], 2006. p. 184–198. Citado na página 69.
- COETZEE, L.; EKSTEEN, J. The internet of things – promise for the future? an introduction. In: **IST-Africa 2011 Conference**. [S.l.: s.n.], 2011. p. 1 – 9. Citado na página 39.
- COMER, D. **Internetworking with TCP/IP: principles, protocols, and architecture**. 4. ed. [S.l.]: Prentice Hall, 2000. Citado na página 40.
- COSTA, M.; CROWCROFT, J.; CASTRO, M.; ROWSTRON, A.; ZHOU, L.; ZHANG, L.; BARHAM, P. Vigilante: End-to-end containment of internet worms. In: **ACM symposium on Operating systems principles**. [S.l.: s.n.], 2005. p. 133–147. Citado na página 69.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Sistemas distribuídos - Conceitos e projeto**. Quarta edição. [S.l.]: Bookman, 2007. Citado nas páginas 26, 27, 36 e 37.

ERRADI, A.; MAHESHWARI, P. A broker-based approach for improving web services reliability. In: **IEEE International Conference on Web Services (ICWS)**. [S.l.: s.n.], 2005. p. 355 – 362. Citado na página 40.

EWING, J. M.; MENASCE, D. A. A meta-controller method for improving run-time self-architecting in soa systems. In: **ACM/SPEC International Conference on Performance Engineering**. [S.l.: s.n.], 2014. p. 173 – 184. Citado nas páginas 27, 64 e 67.

FANJIANG, Y. Y.; SYU, Y.; WU, C. H.; KUO, J. Y. Genetic algorithm for qos-aware dynamic web services composition. In: **International Conference on Machine Learning and Cybernetics**. [S.l.: s.n.], 2010. p. 3246 – 3251. Citado nas páginas 43 e 96.

FARKAS, P.; CHARAF, H. Web services planning concepts. **Journal of .NET Technologies**, v. 1, p. 9 – 12, 2003. Citado nas páginas 40 e 41.

FELLER, E.; RILLING, L.; MORIN, C. Snooze: A scalable and autonomic virtual machine management framework for private clouds. In: **IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing**. [S.l.: s.n.], 2012. p. 482 – 489. Citado nas páginas 68 e 72.

FERRIS, C.; FARRELL, J. **What are web services?** 2003. Communications of the ACM Magazine. Citado na página 40.

GANEK, A. G.; CORBI, T. A. The dawning of the autonomic computing era. **IBM Systems Journal**, v. 42, n. 1, p. 5 – 18, 2003. Citado nas páginas 26 e 44.

GERGIN, I.; SIMMONS, B.; LITOIU, M. A decentralized autonomic architecture for performance control in the cloud. In: **IEEE International Conference on Cloud Engineering**. [S.l.: s.n.], 2014. p. 574 – 579. Citado na página 27.

GHANDI, N.; HELLERSTEIN, J.; PAREKH, L.; TILBURY, D. M. Managing the performance of lotus notes:a control theoretic approach. In: **International Computer Measurement Group Conference**. [S.l.: s.n.], 2001. p. 1 – 12. Citado na página 67.

GIUSTI, R. **Descoberta de regras de conhecimento utilizando computação evolutiva multi-objetivo**. Dissertação (Mestrado), 2010. Citado nas páginas 52 e 53.

GREGG, B. **Systems Performance - Enterprise and the Cloud**. [S.l.]: Prentice Hall, 2014. Citado nas páginas 56, 57, 58, 85 e 124.

GUINEA, S.; SAEEDI, P. Coordination of distributed systems through self-organizing group topologies. In: **IEEE Software Engineering for Adaptive and Self-Managing Systems (SEAMS)**. [S.l.: s.n.], 2012. p. 63 – 72. Citado na página 46.

HARBAOUI, A.; SALMI, N.; DILLENSEGER, B.; VINCENT, J.-M. Introducing queuing network-based performance awareness in autonomic systems. In: **2010 Sixth International Conference on Autonomic and Autonomous Systems (ICAS)**. [S.l.: s.n.], 2010. p. 7 – 12. Citado na página 80.

JACOB, B.; LANYON-HOGG, R.; YASSIN, A. F. **A practical guide to the ibm autonomic computing toolkit**. [S.l.: s.n.], 2004. Citado na página 44.

- JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. **The Unified Software Development Process**. [S.l.]: Addison-Wesley, 2005. Citado nas páginas 31, 32, 33, 34, 35 e 36.
- JAIN, R. **The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling**. [S.l.]: Wiley-Interscience, 1991. Citado nas páginas 54 e 56.
- JIANG, J.; LU, J.; ZHANG, G. An innovative self-adaptive configuration optimization system in cloud computing. In: **Ninth IEEE International Conference on Dependable, Autonomic and Secure Computing**. [S.l.: s.n.], 2011. p. 621 – 627. Citado nas páginas 68 e 73.
- KALEPU, S.; KRISHNASWAMY, S.; LOKE, S. W. Verity: A qos metric for selecting web services and providers. In: **International Conference on Web Information Systems Engineering Workshops**. [S.l.: s.n.], 2003. p. 131 – 139. Citado na página 41.
- KELLERER, H.; PFERSCHY, U.; PISINGER, D. **Knapsack Problems**. [S.l.]: Springer, 2004. Citado nas páginas 50, 53 e 54.
- KEPHART, J.; CHESS, D. The vision of autonomic computing. n. 36, p. 41 – 50, 2003. Citado na página 45.
- KHALID, A.; HAYE, M.; KHAN, M.; SHAMAIL, S. Survey of frameworks, architectures and techniques in autonomic computing. In: **Fifth International Conference on Autonomous and Autonomic Systems (ICAS)**. [S.l.: s.n.], 2009. p. 220 – 225. Citado na página 66.
- KO, J. M.; KIM, C. O.; KWON, I.-H. Quality-of-service oriented web service composition algorithm and planning architecture. **The Journal of Systems and Software**, v. 81, n. 11, p. 2079 – 2090, 2008. Citado nas páginas 42 e 43.
- KREGER, H. **Web Services Conceptual Architecture (WSCA 1.0)**. 2001. Site. Disponível em: <http://users.cs.uoi.gr/~pitoura/courses/ds04_gr/webt.pdf>. Citado nas páginas 40 e 41.
- KUROSE, J.; ROSS, K. **Redes de Computadores e a Internet: Uma abordagem top-down**. [S.l.: s.n.], 2008. Citado nas páginas 25 e 42.
- LASIERRA, N.; ALESANCO, A.; GARCIA, J.; O’SULLIVAN, D. Data management in home scenarios using an autonomic ontology-based approach. In: **IEEE International Workshop on Managing Ubiquitous Communications and Services**. [S.l.: s.n.], 2012. p. 94 – 99. Citado na página 72.
- LIU, X.; BOUGUETTAYA, A.; WU, J.; ZHOU, L. Ev-lcs: A system for the evolution of long-term composed services. **IEEE Transactions on Services Computing**, v. 6, n. 1, p. 102 – 115, 2013. Citado na página 73.
- MENASCÉ, D.; BARBARÁ, D.; DODGE, R. Preserving qos of e-commerce sites through self-tuning: A performance model approach. In: **ACM Conference on e-commerce**. [S.l.: s.n.], 2001. p. 1 – 11. Citado nas páginas 27 e 64.
- NAKAMURA, L.; PRADO, P. F. do; LIBARDI, R.; NUNEZ, L.; MENEGUETTE, R.; ESTRELLA, J.; SANTANA, M.; SANTANA, R.; REIFF-MARGANIEC, S. Efficient web services selection based on qos through a distributed parallel semantic approach. In: **IFIP International Conference on New Technologies, Mobility and Security**. [S.l.: s.n.], 2015. Citado na página 45.

- NAKAMURA, L. H. V.; PRADO, P. F. do; LIBARDI, R.; NUNES, L.; ESTRELLA, J.; SANTANA, R.; SANTANA, M.; REIFF-MARGANIEC, S. Fast selection of web services with qos using a distributed parallel semantic approach. In: **IEEE International Conference on Web Services**. [S.l.: s.n.], 2014. p. 680 – 681. Citado na página 45.
- NALLUR, V.; BAHSOON, R. A decentralized self-adaptation mechanism for service-based applications in the cloud. **IEEE Transactions on Software Engineering**, v. 39, n. 5, p. 591 – 612, 2013. Citado na página 63.
- PALMA, N. D.; HAGIMONT, D.; BOYER, F.; BROTO, L. Self-protection in a clustered distributed system. **IEEE Transactions on Parallel and Distributed Systems**, v. 23, n. 3, p. 330 – 336, 2012. Citado nas páginas 63 e 69.
- PANNU, H. S.; LIU, J.; FU, S. Aad: Adaptive anomaly detection system for cloud computing infrastructures. In: . [S.l.: s.n.], 2012. Citado nas páginas 68 e 72.
- PAPAZOGLU, M. P.; GEORGAKOPOULOS, D. **Introduction: Service-oriented computing**. 2003. Communications of the ACM Magazine. Citado na página 40.
- PEDERSEN, J. M.; RIAZ, M. T.; JÚNIOR, J. C.; DUBALSKI, B.; LEDZINSKI, D.; PATEL, A. Assessing measurements of qos for global cloud computing services. In: **Ninth IEEE International Conference on Dependable, Autonomic and Secure Computing**. [S.l.: s.n.], 2011. p. 682 – 689. Citado na página 96.
- PRADO, P. F.; NAKAMURA, L.; ESTRELLA, J.; SANTANA, M.; SANTANA, R. A performance evaluation study for qos-aware web services composition using heuristic algorithms. In: **The Seventh International Conference on Digital Society (ICDS)**. [S.l.: s.n.], 2013. p. 53 – 58. Citado nas páginas 27, 64, 65 e 74.
- PRADO, P. F. do. **Desenvolvimento e avaliação de algoritmos para composição dinamica de web services baseada em QoS**. Dissertação (Mestrado) — Universidade de São Paulo (USP), 2012. Citado nas páginas 65, 98, 114, 124 e 128.
- PRADO, P. F. do; NAKAMURA, L.; ESTRELLA, J.; SANTANA, M.; SANTANA, R. A performance evaluation study for qos-aware web services composition using genetic algorithms. In: **Proceedings of: XI Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPerformance)**. [S.l.: s.n.], 2012. Citado na página 65.
- PRADO, P. F. do; NAKAMURA, L. H. V.; ESTRELLA, J.; SANTANA, M.; SANTANA, R. Different approaches for qos-aware web services composition focused on e-commerce systems. In: **13th Symposium on Computing Systems**. [S.l.: s.n.], 2012. p. 179 – 186. Citado na página 65.
- PRADO, P. F. do; SANTANA, M. J.; SANTANA, R. **Composição de serviços web: Uma abordagem baseada em qualidade**. [S.l.]: Novas Edições Acadêmicas, 2015. 108 p. Citado na página 65.
- RISH, I.; BRODIE, M.; ODINTSOVA, N.; MA, S.; GRABARNIK, G. Real-time problem determination in distributed systems using active probing. In: **Network Operations and Management Symposium**. [S.l.: s.n.], 2004. p. 133– 146. Citado na página 68.
- ROSA, L.; RODRIGUES, L.; LOPES, A.; HILTUNEN, M.; SCHLICHTING, R. Self-management of adaptable component-based applications. **IEEE Transactions on Software Engineer**, v. 39, n. 3, p. 403 – 421, 2013. Citado na página 80.

- SCHIMIDT, S. O.; PRADO, P. F. do; NEVES, A. J. da S. Infraestrutura de ti e tecnologias emergentes. In: **Fundamentos de Sistemas de Informação**. [S.l.: s.n.], 2014. Citado na página 65.
- SHAO, L.; GUO, Y.; CHEN, X.; HE, Y. Pattern-discovery-based response time prediction. In: _____. **Advances in Automation and Robotics**. [S.l.]: Springer, 2012. v. 2, p. 355 – 362. Citado na página 96.
- SHIVAM, P.; BABU, S.; CHASE, J. S. Learning application models for utility resource planning. In: **IEEE International Conference on Autonomic Computing**. [S.l.: s.n.], 2006. p. 255 – 264. Citado na página 67.
- SOLOMON, B.; IONESCU, D.; LITOU, M.; ISZLAI, G. Designing autonomic management systems for cloud computing. In: . [S.l.: s.n.], 2010. Citado na página 74.
- SONG, H.; LI, J.; LIU, X. Idlecached: An idle resource cached dynamic scheduling algorithm in cloud computing. In: **International Conference on Ubiquitous Intelligence and Computing and International Conference on Autonomic and Trusted Computing**. [S.l.: s.n.], 2012. p. 912 – 917. Citado na página 72.
- TANENBAUM, A. S.; STEEN, M. V. **Distributed Systems: Principles and Paradigms**. [S.l.]: Pearson Prentice Hall, 2006. Citado nas páginas 25, 38, 47 e 49.
- THOMAS, J. P.; THOMAS, M.; GHINEA, G. Modeling of web services flow. In: **IEEE International Conference on e-commerce**. [S.l.: s.n.], 2003. p. 391 – 398. Citado na página 40.
- VESEGNA, S. **IP Quality of Service**. [S.l.]: Cisco Press, 2001. Citado na página 41.
- WANG, K.; TIAN, N. Performance modelling of composite web services. In: **Pacific-Asia Conference on Circuits, Communications and Systems**. [S.l.: s.n.], 2009. p. 563 – 566. Citado na página 42.
- ZHANG, C.; MA, Y. Dynamic genetic algorithm for search in web service compositions based on global qos evaluations. In: **International Conference on Scalable Computing and Communications**. [S.l.: s.n.], 2009. p. 644 – 649. Citado na página 43.
- ZHANG, W.; CHANG, C. K.; FENG, T.; JIANG, H. yi. Qos-based dynamic web service composition with ant colony optimization. In: **IEEE Computer Software and Applications Conference**. [S.l.: s.n.], 2010. p. 493 – 502. Citado na página 96.
- ZHENG, Z.; ZHANG, Y.; LYU, M. R. Investigating qos of real-world web services. **IEEE Transactions on Services Computing**, n. 1, p. 32 – 39, 2014. Citado na página 96.
- ZHI-PENG, G.; JIAN, C.; XUE-SONG, Q.; LUO-MING, M. Qoe/qos driven simulated annealing-based genetic algorithm for web services selection. **The Journal of China Universities of Posts and Telecommunications**, v. 16, n. 1, p. 102 – 107, 2009. Citado na página 42.
- ZULKERNINE, F. H.; MARTIN, P. An adaptive and intelligent sla negotiation system for web services. **IEEE Transactions on Services Computing**, v. 4, n. 1, p. 31 – 43, 2011. Citado nas páginas 27 e 65.