Estudo e simulação de algoritmos de escalonamento
para grades móveis voltados à conectividade dos
dispositivos móveis

Zhang Yifei

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Zhang Yifei

Estudo e simulação de algoritmos de escalonamento para grades móveis voltados à conectividade dos dispositivos móveis

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Paulo Sérgio Lopes de Souza

USP – São Carlos
Maio de 2016

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

Y634e Yifei, Zhang
Estudo e simulação de algoritmos de escalonamento
para grades móveis voltados à conectividade dos
dispositivos móveis / Zhang Yifei; orientador Paulo
Sérgio Lopes de Souza. - São Carlos - SP, 2016.
53 p.

Dissertação (Mestrado - Programa de Pós-Graduação
em Ciências de Computação e Matemática Computacional)
- Instituto de Ciências Matemáticas e de Computação,
Universidade de São Paulo, 2016.

1. Grade móvel. 2. Escalonamento. 3. Simulação.
4. Tolerância a falha. I. Souza, Paulo Sérgio Lopes
de, orient. II. Título.

Zhang Yifei

**Study and simulation of scheduling algorithms for mobile
grids focused on connectivity mobile devices**

Master dissertation submitted to the Instituto de
Ciências Matemáticas e de Computação – ICMC-
USP, in partial fulfillment of the requirements for the
degree of the Master Program in Computer Science
and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and
Computational Mathematics

Advisor: Prof. Dr. Paulo Sérgio Lopes de Souza

USP – São Carlos
May 2016

AGRADECIMENTOS

Os agradecimentos principais são direcionados à Profa. Dra. Sarita Mazzini Bruschi e ao Prof. Dr. Paulo Sérgio Lopes de Souza pela orientação, sabedoria transmitida e dedicação no desenvolvimento da pesquisa.

Os agradecimentos especiais são direcionados aos colegas do LaSDPC pela ajuda e discussões durante o desenvolvimento do trabalho, pelo tempo que convivemos e batalhamos juntos. Aos professores do LaSDPC pelo compartilhamento de conhecimento.

À CAPES pela oportunidade e pelo apoio financeiro.

RESUMO

YIFEI, Z.. **Estudo e simulação de algoritmos de escalonamento para grades móveis voltados à conectividade dos dispositivos móveis**. 2016. 53 f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Este projeto apresenta o desenvolvimento de um conjunto de algoritmos que objetivam o escalonamento de tarefas em grades móveis, com foco no problema de conectividade e tolerância às falhas. Algoritmos de aprendizado de máquina e algoritmos estocásticos são aplicados na organização de recursos da grade durante o processo de escalonamento de tarefas. Além disso, uma tolerância a falhas foi considerada em conjunto ao algoritmo proposto e o desempenho do algoritmo foi avaliado por meio de simulação. Os resultados mostram que a queda da quantidade de falhas é significativa com a utilização dos algoritmos desenvolvidos, principalmente quando há uma grande quantidade de tarefas e poucos recursos disponíveis. O aumento do tempo de execução é inevitável neste caso, porém é controlável pelas variáveis definidas nos algoritmos.

Palavras-chave: Grade móvel, Escalonamento, Simulação, Tolerância a falha.

ABSTRACT

YIFEI, Z.. **Estudo e simulação de algoritmos de escalonamento para grades móveis voltados à conectividade dos dispositivos móveis**. 2016. 53 f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

This project presents the development of a set of algorithms for task scheduling in a mobile grid, focusing on connectivity problem and fault tolerance. Machine learning algorithms and stochastic algorithms are applied to organize the grid resources during the process of task scheduling. In addition, a technique of fault tolerance is considered in conjunction with the proposed algorithm, and the performance of algorithm is evaluated by simulation. The results show a substantial decrease in the failure rate due to the use of developed algorithms, mainly for the case of large amount of tasks with limited resources. The increase in execution time is inevitable in this case, but it is controllable by variables defined in the algorithms.

Key-words: Mobile grid, Scheduling, Simulation, Fault tolerance.

LISTA DE ILUSTRAÇÕES

Figura 1 – Sistema de escalonamento. Adaptado de (CASAVANT; KUHL, 1988)	9
Figura 2 – Exemplo de classificação utilizando algoritmo KNN	15
Figura 3 – Exemplo do algoritmo K-means	16
Figura 4 – Exemplo de uma cadeia de Markov. Adaptado de (MARKOV, 1971)	18
Figura 5 – Módulos do SimGrid	19
Figura 6 – Arquitetura do OptorSim	20
Figura 7 – Arquitetura do GridSim	21
Figura 8 – Arquitetura do MicroGrid	22
Figura 9 – Modelo da arquitetura proposta	26
Figura 10 – Fluxo do conjunto de algoritmos propostos para o escalonamento de tarefas em uma grade móvel	26
Figura 11 – Pré-processamento dos dados de registro	27
Figura 12 – Transição de estado dos dispositivos	28
Figura 13 – Reestruturação dos recursos	28
Figura 14 – Alterações realizadas no simulador <i>GridSim</i>	32
Figura 15 – Primeiro conjunto de experimentos - Tempo total de execução em relação a carga de trabalho	38
Figura 16 – Primeiro conjunto de experimentos - Quantidade de falha em relação a carga de trabalho	39
Figura 17 – Segundo conjunto de experimentos - Quantidade de falha em relação ao crescimento da carga de trabalho	39
Figura 18 – Segundo conjunto de experimentos - Tempo total de execução em relação ao crescimento da carga de trabalho	40
Figura 19 – Segundo conjunto de experimentos - Taxa X (Falha/Tempo) em relação ao crescimento da carga de trabalho	40
Figura 20 – Terceiro conjunto de experimentos - Tempo de execução dos 3 Logs em relação ao crescimento da carga de trabalho	41
Figura 21 – Terceiro conjunto de experimentos - Quantidade de falha dos 3 logs em relação ao crescimento da carga de trabalho	42
Figura 22 – Quarto conjunto de experimentos - Tempo de execução e quantidade de falhas	43
Figura 23 – Quarto conjunto de experimentos - Relação entre quantidade de falha e tempo de execução	43

Figura 24 – Quinto conjunto de experimentos - Comparações de quantidade de falha, tempo de execução e variávelX	44
Figura 25 – Quinto conjunto de experimentos - Influência dos fatores nos resultados em relação ao tempo de execução	44
Figura 26 – Quinto conjunto de experimentos - Influência dos fatores nos resultados em relação à quantidade de falha	45
Figura 27 – Comparação do tempo total de execução das 2 repetições do último conjunto de experimento	46
Figura 28 – Comparação da quantidade de falha das 2 repetições do último conjunto de experimento	46

LISTA DE QUADROS

Quadro 1 – Comparação entre algoritmos de escalonamento	11
Quadro 2 – Parâmetros para o primeiro conjunto de experimentos	36
Quadro 3 – Detalhes para o primeiro conjunto de experimentos	36
Quadro 4 – Detalhes para o segundo conjunto de experimentos	36
Quadro 5 – Detalhes para o terceiro conjunto de experimentos	37
Quadro 6 – Detalhes para o quarto conjunto de experimentos	37
Quadro 7 – Parâmetros para o quinto conjunto de experimentos	37
Quadro 8 – Detalhes de execução para o quinto conjunto de experimentos	38

LISTA DE ALGORITMOS

Algoritmo 1 – Algoritmo para organização de recurso	30
Algoritmo 2 – Atualização de confiabilidade	30
Algoritmo 3 – Execução do algoritmo de escalonamento	31

LISTA DE ABREVIATURAS E SIGLAS

API		Application Programming Interface
CCR		<i>Communication to Computing Ratio</i>
DLS		<i>Dynamic Level Scheduling</i>
FIFS		<i>First Input First Service</i>
GRAS		Grid Reality and Simulation
ICMC		Instituto de Ciências Matemáticas e de Computação
MOEA	...	<i>Multi-Objective Evolutionary Algorithm</i>
MOEP	...	<i>Multiobjective Evolutionary Programming</i>
MOGA	...	<i>Multiobjective Genetic Algorithm</i>
MPI		Message Passing Interface
MSG		Meta-SimGrid
PBPFT	...	<i>Primary and Backup replica Partition based Fault-tolerant scheduling algorithm</i>
RASD		<i>Reliable Aware Scheduling Distribution Algorithm</i>
RDGS		<i>Reliable Distributed Grid Scheduler Algorithm</i>
RR		<i>Repars and Refine Algorithm</i>
RRank	...	<i>Reliability priority Rank</i>
SPEC		Standard Performance Evaluation Corporation
XBT		Extended Bundle of Tools

SUMÁRIO

1	INTRODUÇÃO	1
1.1	Contextualização e motivação	1
1.2	Objetivos	2
1.3	Estrutura do trabalho	3
2	REVISÃO BIBLIOGRÁFICA	5
2.1	Considerações Iniciais	5
2.2	Grades Computacionais	5
2.2.1	<i>Grades Computacionais Tradicionais</i>	6
2.2.2	<i>Grades Computacionais Móveis</i>	6
2.2.3	<i>Escalonamento em Grades Computacionais</i>	8
2.2.4	<i>Escalonamento em Grades Móveis com foco na confiabilidade</i> . . .	10
2.2.5	<i>Escalonamento em Grades com foco na confiabilidade e tolerância a falhas</i>	12
2.3	Classificação de dados em Aprendizado de Máquina	13
2.3.1	<i>O método KNN (K Nearest Neighbor)</i>	14
2.3.2	<i>O método K-means</i>	15
2.3.3	<i>Aprendizado Bayesiano</i>	16
2.4	Cadeia de Markov	17
2.5	Ferramentas de Simulação	18
2.5.1	<i>SimGrid</i>	18
2.5.2	<i>OptorSim</i>	19
2.5.3	<i>GridSim</i>	20
2.5.4	<i>MicroGrid</i>	22
2.6	Considerações Finais	23
3	ESCALONAMENTO CIENTE DA MOBILIDADE DOS DISPOSITIVOS	25
3.1	Considerações Iniciais	25
3.2	Arquitetura do sistema de escalonamento	25
3.3	Organização de recursos	26
3.4	Controle de falha	29
3.5	Simulação	31

3.6	Considerações Finais	33
4	RESULTADOS	35
4.1	Considerações Iniciais	35
4.2	Planejamento dos Conjuntos de Experimentos	35
4.3	Análise dos resultados	38
4.4	Considerações finais	47
5	CONCLUSÕES	49
5.1	Contribuições	49
5.2	Trabalhos futuros	50
	REFERÊNCIAS	51

INTRODUÇÃO

1.1 Contextualização e motivação

Na década de 90, a *Internet* sofreu uma rápida expansão, tornando-se uma ferramenta fundamental de comunicação e cooperação entre as pessoas que estavam dispersas pelo mundo. Além disso, a *Internet* também teve e tem seu papel fundamental na produção, compartilhamento e utilização de conhecimento, por ser um meio que reúne inúmeros recursos computacionais.

Devido ao aumento da complexidade das aplicações científicas, surgiu a ideia de agrupar este grande número de recursos computacionais espalhados na *Internet*, formando um “supercomputador” para executar algumas tarefas que eram consideradas complexas. Esta ideia originou o desenvolvimento de uma infraestrutura de computação distribuída inspirada nas redes de energia elétrica (BUYA; ABRAMSON; VENUGOPAL, 2005). Nessa infraestrutura, conhecida como grade computacional, seus recursos computacionais geograficamente distribuídos podem ser compartilhados de maneira coordenada, resolvendo os problemas complexos de maneira colaborativa.

Do início do século XXI até os dias atuais, os dispositivos móveis deixaram de ser simplesmente equipamentos dedicados, como telefone celular, câmera digital ou GPS, e se transformaram em pequenos computadores com funções mais genéricas concentrando quase todas as funcionalidades dos dispositivos móveis antigos. Devido a sua grande capacidade computacional, tais dispositivos móveis como *smartphones*, *tablets* e *notebooks* foram incorporados no conceito de grade computacional, a qual considerava apenas o compartilhamento de recursos fixos. A integração entre dispositivos móveis e grades computacionais originou o novo termo grade móvel.

Existem diversas visões sobre a função do dispositivo móvel no contexto de grade, e um grande número de propostas sobre como atingir tais funções (RODRIGUEZ; ZUNINO; CAMPO, 2011). Duas abordagens são mais discutidas neste contexto: a primeira trata o dispositivo móvel

como uma interface entre homem e grade, acessando a grade computacional que tem seus recursos fixos para processar os problemas, e os resultados são retornados para dispositivo móvel. Já na segunda abordagem, os dispositivos móveis são considerados como provedores de recursos, participando diretamente na execução dos problemas.

A expansão de pesquisas voltadas para grade móvel em poucos anos pode ser justificada por dois motivos (RODRIGUEZ; ZUNINO; CAMPO, 2011). O primeiro é a rápida evolução tecnológica dos dispositivos móveis tanto em termos de hardware (por exemplo, processador multicore) quanto de software (por exemplo, SOs estáveis e seus vários aplicativos). O segundo é o grande número de dispositivos móveis comercializados e seu crescimento contínuo na venda, principalmente *smartphones* e *tablets*. De acordo com dados da consultoria Gartner ¹, em 2015, a venda de *smartphones* cresceu como nos períodos passados, chegando a 353 milhões de unidades somente no terceiro trimestre, com aumento de 18,4% comparando com mesmo período de 2014. Além disso, o baixo custo aliado a grande capacidade está acelerando o processo da substituição de PCs por tablets. Portanto, existe uma grande quantidade de recursos computacionais a serem explorados por meio de grade móvel.

É importante notar que uma infraestrutura para compartilhamento de recursos oferecidos pelos dispositivos móveis constitui um ambiente altamente volátil. A disponibilidade dos recursos pode ser afetada por diversos problemas como conectividade e capacidade de bateria, interrompendo e prejudicando a execução das aplicações submetidas.

O estado de conectividade de dispositivo móvel é uma influência forte para o desempenho e o sucesso de execução das aplicações. É fundamental que uma grade móvel seja capaz de escolher os dispositivos com maior disponibilidade para submeter tarefas (PARK; KO; KIM, 2003).

1.2 Objetivos

O objetivo deste projeto de pesquisa é propor, através de estudo, simulação e avaliação de desempenho, um conjunto de algoritmos para escalonamento de tarefas considerando técnicas de aprendizado de máquina e processos estocásticos, de modo a otimizar o desempenho das aplicações através do gerenciamento das tarefas da grade móvel, levando em conta também uma política de tolerância a falhas. Tais algoritmos deverão gerenciar todos os dispositivos móveis presentes em uma grade móvel pela sua confiabilidade, e assegurar o cumprimento dos requisitos de qualidade de serviço das tarefas submetidas pelos usuários. O mecanismo de escalonamento deverá levar em consideração que a confiabilidade de um dispositivo móvel é diretamente relacionada a fatores como qualidade de conexão e vida útil de bateria.

¹ <<http://www.gartner.com/newsroom/id/3169417>>

1.3 Estrutura do trabalho

O restante desta dissertação está organizada da seguinte forma: o Capítulo 2 aborda os conceitos fundamentais para melhor entendimento deste trabalho. É apresentada uma revisão bibliográfica sobre as grades computacionais tradicionais e móveis, seus mecanismos de escalonamento, os métodos de aprendizado de máquina e de processos estocásticos estudados para o desenvolvimento do algoritmo de escalonamento. Além disso, são apresentadas ferramentas de simulação de grade móvel que podem ser utilizadas para a avaliação de desempenho do algoritmo proposto. O Capítulo 3 detalha o projeto e as metodologias empregadas, a arquitetura considerada e os procedimentos de simulação, além da descrição do sistema desenvolvido. O Capítulo 4 apresenta os resultados das simulações realizadas e a uma análise desses resultados. E por fim, o Capítulo 5 conclui o trabalho e apresenta sugestões de trabalhos futuros.

REVISÃO BIBLIOGRÁFICA

2.1 Considerações Iniciais

Neste capítulo é apresentada a fundamentação teórica com finalidade de fornecer os conceitos básicos e subsídios ao projeto de pesquisa. Na Seção 2.2, é realizado um estudo sobre grades computacionais, incluindo os algoritmos de escalonamento existentes e trabalhos relacionados. Na Seção 2.3 e Seção 2.4, são apresentados os conceitos para o entendimento dos algoritmos propostos. Por fim, na Seção 2.5, são apresentados os simuladores para grade computacional.

2.2 Grades Computacionais

A partir da década de 1990, a grade computacional surgiu como um paradigma para a coordenação do compartilhamento de recursos computacionais e solução dos problemas científicos e comerciais complexos. Um sistema de computação em grade nada mais é do que um sistema distribuído em larga escala projetado para agregar recursos de várias máquinas que estão geograficamente distantes (KATSAROS; POLYZOS, 2007). Usuários destes sistemas têm a oportunidade de aproveitar de maneira única os recursos computacionais, tais como armazenamento e processamento, os quais seriam impossíveis de atingir de outra forma. É importante ressaltar que na maioria dos casos, tais recursos seriam desperdiçados se não fossem agregados dentro da grade.

A grade computacional é uma importante iniciativa de computação que envolve a agregação de computadores conectados em rede para formar um sistema distribuído de grande escala. Esforços anteriores de computação em grade incluíram vários projetos em diversas áreas, tais como projeto da busca de vida extraterrestre, da visualização molecular, da distribuição de conteúdo multimídia, etc. (PHAN; HUANG; DULAN, 2002).

2.2.1 Grades Computacionais Tradicionais

Uma grade computacional deve prover acesso consistente aos recursos disponíveis de forma que as descontinuidades físicas como as diferenças entre hardwares, sistemas operacionais e canais de comunicação, tornem-se completamente transparentes ao usuário final. Dessa forma, a grade computacional transforma um ambiente altamente heterogêneo em um ambiente virtual homogêneo. Considerando todas essas características, uma grade computacional pode ser definida como uma infraestrutura que fornece acesso confiável, consistente e transparente, possibilitando a busca, descoberta e compartilhamento de recursos computacionais, independentemente da localização geográfica de tais recursos ou dos usuários (BUYYA; MURSHED, 2002).

De acordo com Krauter, Buyya e Maheswaran (BUYYA; MURSHED, 2002), as grades computacionais podem ser categorizadas em função da atividade principal a que se destinam. Neste caso, são consideradas três categorias, descritas a seguir:

- **Grade de Processamento:** a capacidade de processamento dos recursos que compõem a grade é utilizada para resolver problemas com alto custo computacional. Um exemplo é o *folding@home3*, desenvolvido pela universidade de Stanford. Nesse projeto, computadores espalhados pelo mundo são utilizados para estudar problemas relacionados ao dobramento de proteínas (*protein folding*)¹. Cada computador possui um programa cliente instalado que, além de executado em background, faz uso do processador apenas em momentos de ociosidade.
- **Grade de Serviços:** os recursos computacionais disponíveis são utilizados para prover serviços que possuem dificuldades de serem oferecidos individualmente. Como exemplo tem-se as plataformas de colaboração à distância, que permitem a integração de usuário e aplicação em ambientes virtuais compartilhados.
- **Grade de Dados:** a finalidade deste tipo de grade é oferecer uma infraestrutura para armazenamento e gerenciamento de dados. Como exemplo, pode-se citar a integração de bancos de dados heterogêneos distribuídos geograficamente.

2.2.2 Grades Computacionais Móveis

Uma grade móvel pode ser considerada como uma extensão de grade computacional tradicional baseada em recursos fixos, a qual permite a participação dos dispositivos móveis como *smartphones*, *tablets* e *notebooks* nas grades computacionais. Atualmente existem duas abordagens que são discutidas em relação à grade móvel, as quais se definem em função dos papéis exercidos pelos dispositivos móveis: consumidor ou provedor de recursos computacionais (RODRIGUEZ; ZUNINO; CAMPO, 2011).

¹ <<https://folding.stanford.edu/>>

Alguns pesquisadores utilizam dispositivos móveis como uma interface entre o homem e o sistema de grade, argumentando que tais dispositivos não possuem recursos suficientes para compartilhar com a grade, mas são suficientemente potentes para acessar e consumir recursos da grade. Em contraste, outros pesquisadores defendem dispositivos móveis como provedores de recursos no sistema de grade.

Independente da abordagem, uma grade móvel deve levar em consideração as restrições presentes nos dispositivos móveis. Com a evolução tecnológica crescente, algumas restrições estão sendo tratadas rapidamente, tais como capacidade de processamento e de armazenamento, dado que já estão disponíveis no mercado alguns modelos de *smartphones high-end* com capacidade de processamento e armazenamento superiores a da maioria dos *notebooks* e *desktops*. Mas ainda existem algumas restrições que não possuem previsão de boas soluções em um curto prazo, como por exemplo, o tempo de vida da bateria e a qualidade de conectividade dos dispositivos móveis relacionada com a sua mobilidade, a qual podem afetar a disponibilidade dos recursos ofertados.

Dispositivos móveis podem oferecer três tipos de recursos para um sistema de grade: tempo de processamento, espaço de armazenamento e informações de sensor. O tipo de recurso compartilhado depende do objetivo principal da grade. Como já mencionado na seção anterior, as três categoria de grades (Grade de processamento, Grade de serviços e Grade de dados) podem ser encaixadas neste contexto.

As discussões sobre a incapacidade de usar os dispositivos móveis como provedores de recursos surgem devido a aspectos como desconexões e comunicação intermitente. Portanto, é necessário estabelecer algumas medidas que garantam acesso confiável aos recursos oferecidos. A arquitetura baseada em *proxy* proposto por Park, Ko e Kim ([PARK; KO; KIM, 2003](#)) pode ser uma possível solução. Nesta arquitetura, os *proxies* funcionam como mediadores que cuidam da submissão, monitoramento e retorno das requisições realizadas pelos dispositivos móveis.

A adoção da abordagem de considerar dispositivos móveis como provedores de recursos pode ser justificada pelo grande crescimento da sua comercialização, juntamente com a evolução tecnológica desses dispositivos, pois, como já descrito, o poder computacional desses dispositivos móveis já não é tão inferior ao de computadores normais, tipo *desktop*.

De acordo com Katsaros e Polyzos ([KATSAROS; POLYZOS, 2007](#)), a provisão de recursos móveis pode ser realizada em duas arquiteturas distintas: grades móveis *on-site* e grades móveis *ad hoc*. No caso da grade móvel *on-site*, dispositivos móveis localizados em uma determinada área são coordenados por uma entidade central associada a um ponto de acesso. As informações sobre a localização e a disponibilidade dos recursos são monitoradas pela entidade central, que possui uma lista com informações detalhadas de cada dispositivo que faz parte deste conjunto. Já no caso da grade móvel *ad hoc*, não existe uma infraestrutura organizada como o caso de *on-site*. A topologia da rede é formada de maneira dinâmica, com os dispositivos móveis autônomos e livres de entrar ou sair da grade a qualquer momento. A arquitetura proposta

neste projeto de mestrado é baseada na arquitetura *on-site*, estabelecendo algumas regras de escalonamento através da análise de histórico de acesso.

2.2.3 Escalonamento em Grades Computacionais

O escalonamento em grades computacionais pode ser definido como o processo de atribuição de tarefas aos recursos que estão distribuídos entre diversas organizações virtuais. Este processo é mais complexo comparado com os sistemas distribuídos tradicionais, como aplicações comerciais (reservas de bilhetes, bancos), devido às características presentes na grade computacional. Algumas dessas características são descritas a seguir (XHAFÁ; ABRAHAM, 2010):

- **Dinamismo:** as informações sobre o *status* dos recursos são necessárias para a realização do escalonamento.
- **Heterogeneidade de recursos:** os recursos podem ser bem distintos em termos de potência de processamento, armazenamento, meio de comunicação e sistema operacional.
- **Heterogeneidade de tarefa:** as tarefas submetidas à grade podem ter exigências de recursos computacionais muito distintas. A previsão de carga é difícil de ser realizada.
- **Segurança:** tarefas diferentes podem exigir níveis de segurança diferentes, sendo que seus requisitos de segurança devem ser atendidos.

O escalonamento em grade computacional deve ser realizado em três etapas: descoberta dos recursos, seleção dos recursos e submissão das tarefas (DONG; AKL, 2006). O processo começa pela obtenção de um conjunto de recursos que atende aos requisitos impostos pela tarefa submetida para execução. A partir desses recursos, considerando a política de escalonamento adotada, são selecionados os recursos que irão executar a tarefa após a verificação de suas disponibilidades.

O problema de escalonamento em grades computacionais pode ser dividido em três componentes principais de acordo com Casavante e Kuhl (CASAVANT; KUHLE, 1988): consumidor, recurso e política. Como qualquer outro problema de gerenciamento ou de controle, a melhor maneira de entender o funcionamento do escalonamento é através da observação do seu efeito sobre o ambiente de funcionamento. Além disso, é importante considerar dois fatores enquanto se avalia um sistema de escalonamento: a satisfação do consumidor em relação ao gerenciamento dos recursos realizado pelo escalonador (desempenho); e a satisfação do consumidor em relação ao custo para acessar os recursos. O relacionamento entre escalonador, políticas, recursos e consumidores é ilustrado na Figura 1.

Para o escalonamento na área de grade computacional existem diversos algoritmos, sendo que alguns são apresentados a seguir:

Figura 1 – Sistema de escalonamento. Adaptado de (CASAVANT; KUHL, 1988)

Fonte: Casavant e Kuhl (1988).

- **Algoritmo do Vagalume:** como seu próprio nome diz, foi inspirado no comportamento social dos vagalumes. O foco principal do algoritmo é completar as tarefas em um mínimo *makespan* (tempo de conclusão), com a utilização eficiente dos recursos disponíveis. O processo de otimização do algoritmo segue quatro regras básicas de acordo com Yang (YANG, 2010) e Senthilnath (SENTHILNATH; OMKAR; MANI, 2011):

1. Um vagalume atrai e pode ser atraído pelos outros vagalumes;
2. Os vagalumes mais brilhantes atraem os menos brilhantes;
3. Os brilhos decrescem com o aumento da distância;
4. O vagalume mais brilhante pode caminhar aleatoriamente.

A execução do algoritmo é dividida também em quatro fases (YOUSIF, 2011):

1. Os parâmetros como população inicial, aptidão e atratividade são inicializados, e os recursos disponíveis e lista de tarefas são identificados;
2. O brilho de cada vagalume é obtido através da função de aptidão e as distâncias são calculadas. Os vagalumes menos brilhantes são movidos para os vagalumes mais brilhantes;
3. A nova solução é avaliada e a intensidade de brilho é atualizada;
4. Os vagalumes são classificados e o melhor global é identificado. Os parâmetros de iteração são atualizados.

O processo de quatro fases se repete até atingir a condição de parada, que pode ser número de iteração ou determinado valor de aptidão.

- **Algoritmo de Colônia de Formigas:** trata-se de um algoritmo bioinspirado, derivado do comportamento social das formigas. As formigas trabalham juntas para achar o menor caminho entre o alimento e seu ninho. Enquanto as formigas se movimentam, cada uma

delas deposita uma substância química chamada de feromônio, e o menor caminho é encontrado através do feromônio.

Com a mesma ideia das formigas aplicada no escalonamento em grade computacionais, cada recurso possui um valor de feromônio, sendo que o seu valor é incrementado ao completar uma tarefa, ou decrementado ao não executar uma tarefa. O único problema neste caso é a estagnação, que pode acontecer quando várias tarefas são submetidas para os mesmos recursos com alto valores de feromônio. A solução deste problema de estagnação foi estudado por Nasir e Ku-Mahamud ([NASIR; KU-MAHAMUD, 2010](#)), onde um broker de recurso é responsável por remover ou ocultar o recurso com maior valor de feromônio alocado pela tarefa submetida, e disponibilizar tal recurso novamente após a conclusão de tarefa e atualização do valor de feromônio.

- **Algoritmo de Escalonamento Baseado em Agrupamento:** tem como foco a melhor utilização dos recursos. Antes de atribuir as tarefas aos recursos, estes são agrupados de acordo com sua capacidade de processamento: milhões de instruções por segundo (MIPS), largura de banda (Mb/s) e tamanho de memória (Mb). Dessa maneira, o algoritmo de escalonamento determina o grupo de tarefas e seleciona os recursos para esse grupo de tarefas. Assim que as tarefas são agrupadas e associadas aos recursos selecionados, as tarefas são direcionadas aos recursos para processamento ([GOMATHI; MANIMEGALAI, 2011](#)).
- **Algoritmo de Escalonamento Ciente de Comunidade:** trata-se de um algoritmo dinâmico descentralizado, ou seja, as tarefas podem ser reescaloadas e distribuídas uniformemente a todos os recursos. O algoritmo considera os recursos como nós. Se um nó A recebe uma tarefa, ele atua como nó iniciante, encaminhando mensagens de requisição de processamento para todos os outros nós que possuem comunicação com o nó A. Os nós que estão dispostos a receber a tarefa respondem com uma mensagem de aceite, e o nó A analisa os históricos dos nós respondidos para atribuir a tarefa ao melhor nó. O nó que recebeu a tarefa do nó iniciante procura a tarefa com maior tempo de espera na lista de tarefas, e atribuí-a a outro nó, repetindo o processo que o nó A realizou ([HUANG et al., 2013](#)).

Os algoritmos descritos são bem diferentes quando comparados uns com os outros, sendo que cada um deles funciona de uma maneira totalmente diferente do outro. A Tabela 1 a seguir descreve e compara as vantagens e desvantagens dos algoritmos listados acima, fornecendo uma visualização mais clara.

2.2.4 Escalonamento em Grades Móveis com foco na confiabilidade

Em um ambiente de grade móvel, os recursos são altamente voláteis devido a não estabilidade da conexão, comunicação intermitente e tempo útil da bateria, fazendo como que

o desempenho das aplicações submetidas seja prejudicado. Dessa forma, a escolha da política de escalonamento que considere tais problemas é fundamental para o desempenho da grade inteira. São apresentados a seguir alguns trabalhos que propõem soluções para o escalonamento, considerando alguns mecanismos que asseguram a sua confiabilidade, promovendo a tolerância a falhas.

Quadro 1 – Comparação entre algoritmos de escalonamento

Algoritmos	Vantagens	Desvantagens
Vagalume	Convergência e minimização de custos	População inicial fixo
Agrupamento	Boa utilização de recursos	Complexidade de tempo alta
Ciente de Comunidade	Adaptabilidade	Overhead de mensagens
Colônia de Formigas	Adaptabilidade e boa robustez em ambiente dinâmico	Informação de cada recurso necessária, longo tempo de processamento devido a atualização constante do valor de feromônio

Park, Ko e Kim ([PARK; KO; KIM, 2003](#)) propuseram um algoritmo de escalonamento que considera as probabilidades de um dispositivo estar inicialmente conectado ou desconectado para calcular o tempo médio de espera para que aconteça a reconexão. Classificando e ordenando os recursos da maior taxa de disponibilidade para a menor taxa de disponibilidade, as tarefas são atribuídas aos dispositivos a partir desta lista de recursos ordenada.

Jang e Lee ([JANG; LEE, 2011](#)) desenvolveram um algoritmo de escalonamento que leva em consideração a confiabilidade dos recursos ofertados pelos dispositivos móveis. A confiabilidade de um recurso é obtida estatisticamente com base de cinco métricas: tempo médio de resposta, tempo médio de reconexão, disponibilidade de processador, disponibilidade de memória e capacidade de bateria. Semelhante ao algoritmo proposto por Park, Ko e Kim ([PARK; KO; KIM, 2003](#)), a atribuição de tarefa a recursos segue uma lista ordenada da maior para a menor confiabilidade.

Em Huang et al. ([HUANG et al., 2006](#)), os autores apresentaram um modelo de escalonamento hierárquico de dois níveis, com objetivo de prover uma utilização eficiente de energia dos dispositivos móveis que participam de uma grade computacional. O primeiro nível adota o algoritmo *First Input First Service* (FIFS) para escalonar as tarefas em nós fixos, que podem atuar como proxies entre a grade e um domínio sem fio como por exemplo um hotspot de *WLAN*. O segundo nível emprega uma heurística *Min-Min* modificada para realizar o escalonamento em cada um dos domínios sem fio. Neste caso, a seleção dos elementos móveis é conduzida de forma a minimizar a energia dissipada durante a execução das tarefas.

2.2.5 Escalonamento em Grades com foco na confiabilidade e tolerância a falhas

Como foi mencionado no início desta monografia, o foco do projeto é o escalonamento que considere o problema de conectividade nas grades móveis como foco. A seguir são apresentados alguns algoritmos de escalonamento existentes cujos mecanismos são voltados para tolerância a falhas que asseguram a confiabilidade.

O Algoritmo *Reliable Aware Scheduling Distribution Algorithm* (RASD) foi desenvolvido tendo como base o algoritmo de busca do melhor caminho de comunicação com confiabilidade e introduziu o conceito de *Reliability priority Rank* (RRank) para estimar a prioridade de tarefa durante o escalonamento (YANG, 2010). As tarefas são atribuídas aos processadores e as replicações são utilizadas para promover a confiabilidade do sistema. É importante resaltar que o algoritmo RASD é adequado apenas para sistemas distribuídos heterogêneos de pequena escala, pois o mecanismo da replicação de tarefas pode gerar *overhead* no sistema.

O Algoritmo *Repars and Refine Algorithm* (RR) é dividido nos algoritmos *Repar* e *Refine*. O *Repar* foi proposto para escalonar as tarefas periódicas num sistema heterogêneo com objetivo de promover a confiabilidade enquanto o tempo limite de execução das tarefas não é violado. Já o *Refine* foi desenvolvido como um algoritmo voltado para tolerância a falhas (LUO; QIN; BELLAM, 2007). São calculados primeiramente os custos de confiança para todas as tarefas, onde uma tarefa com menor custo de confiança significa que tal tarefa é menos exigente em relação ao prazo para entrega do resultado, por exemplo.

O algoritmo *Repar* escolhe o melhor recurso candidato (recurso com maior confiabilidade) para uma tarefa com menor custo de confiança, desta forma, a probabilidade de sucesso na entrega do resultado é maximizada e o tempo de conclusão das tarefas é minimizado. O algoritmo *Refine* utiliza a estratégia *First-Fit*, combinando as tarefas e os recursos com custo de confiança e confiabilidade equivalentes.

O Algoritmo *Multi-Objective Evolutionary Algorithm* (MOEA) é composto pelos algoritmos *Multiobjective Genetic Algorithm* (MOGA) e *Multiobjective Evolutionary Programming* (MOEP), desenvolvido para minimizar o tempo de escalonamento (*Schedule Length*) e probabilidade de falha (CHITRA; VENKATESH; RAJARAM, 2011). As abordagens do MOGA consistem em técnicas probabilísticas, pesquisa global e a sua implementação é dividida em seis etapas: inicialização, cruzamento, mutação, avaliação, classificação e seleção.

O Algoritmo *Reliable Distributed Grid Scheduler Algorithm* (RDGS) tem como objetivo explorar o fator de confiabilidade e maximizar a quantidade total das tarefas concluídas com sucesso. O RDGS foi desenvolvido baseando no *Communication to Computing Ratio* (CCR), que por sua vez é utilizado para decidir se o escalonamento da tarefa vai ser local ou remoto, ou seja, indicar o local da grade apropriado para as tarefas (RAM; RAO; GOVARDHAN, 2011). O escalonador cria uma fila onde coloca as tarefas atribuídas aos nós deficientes (recursos que não

conseguiram concluir a execução das suas tarefas), e essas tarefas enfileiradas são reescaladas a novos pontos.

O algoritmo *Primary and Backup replica Partition based Fault-tolerant scheduling algorithm* (PBPFT) (QIU *et al.*, 2013) utiliza as técnicas de *phasing delay* e agrupamento de processador, para reduzir os processamentos redundantes e o tempo de execução das chamadas de teste, promovendo a eficiência e reduzindo a complexidade do algoritmo. A réplica de tarefa é executada em 3 formas: a réplica de uma tarefa é executada se esta tarefa falhou durante a execução (Passivo), a réplica de uma tarefa sempre é executada junto com a tarefa original (Comum-ativo), somente uma parte da réplica de uma tarefa é executada na ausência de falhas (Diferido-ativo). A tarefa com tempo de execução mais curta tem maior prioridade durante o escalonamento, e para cada tarefa, sua cópia original tem prioridade sobre suas réplicas.

O algoritmo *Hierarchical Grouping based QoS Algorithm* (JOHN; MOHAMED, 2015) é composto por duas etapas: agrupamento de recurso e escalonamento baseado no QoS. Os recursos são agrupados pelas suas informações como largura de banda, poder de processamento (MIPS), quantidade de falha e localização, o *Resource Value* é calculado através dessas informações que representa a qualidade de cada recurso. A taxa da restrição de QoS é calculado para cada tarefa, quando maior a taxa de restrição, menor o tamanho de dados acessados. O algoritmo sempre aloca as tarefas para os recursos com maiores *Resource Value*, considerando a largura da banda e a taxa da restrição de QoS.

O algoritmo *fault recovery-based DLS algorithm* (QI; LI, 2015) foi desenvolvido tendo como base o algoritmo tradicional *Dynamic Level Scheduling* (DLS), junto com um mecanismo de tolerância às falhas considerando o modelo cognitivo bayesiano, reduzindo a quantidade de falha durante escalonamento. De acordo com o modelo cognitivo bayesiano, uma falha é dividida em falha de *hardware* e falha de *software*, sendo que uma falha de *hardware* é dividida em falha de comunicação e falha de nó, e todas falhas são divididas em falhas recuperáveis e não recuperáveis. O trabalho de escalonamento é realizado de acordo com as probabilidade de falhas recuperáveis, falhas não recuperáveis e execuções sucedidas.

2.3 Classificação de dados em Aprendizado de Máquina

Nos últimos anos, a Inteligência Artificial ganhou um papel fundamental no cenário mundial de estudos relacionados à computação. As pesquisas foram iniciadas logo após o fim da Segunda Guerra Mundial e o seu nome foi definido no final da década de 50 (RUSSELL; NORVIG, 1996).

A inteligência artificial cobre uma vasta área de estudos com muitas subáreas. Neste projeto o foco principal é o Aprendizado de Máquina, dedicado ao desenvolvimento de algoritmos que permitam ao computador aperfeiçoar seu desempenho, ou seja, programar computadores para aprender um determinado comportamento ou padrão automaticamente a partir de exemplos

ou observações. Os estudos sobre aprendizado de máquina dividem-se em três grupos básicos: aprendizagem supervisionada, não-supervisionada e por reforço, as quais são apresentadas com mais detalhes a seguir:

- **Aprendizado supervisionado:** o objetivo é induzir conceitos a partir de exemplos que estão pré-classificados, ou seja, exemplos que estão rotulados com uma classe conhecida. Se as classes possuírem valores discretos, o problema é categorizado como classificação. Caso as classes possuam valores contínuos, o problema é categorizado como regressão. O aprendizado supervisionado é utilizado quando, em um banco de dados, tem-se tanto as perguntas como as respostas. É usado para a realização de treinamento de redes neurais na obtenção de classificação, funções de aproximação ou modelagem e previsões baseadas no tempo.
- **Aprendizado não-supervisionado:** existe a incerteza sobre o resultado esperado, desta forma, é necessário utilizar os métodos probabilísticos para simular uma experiência não vivida. Para realizar tais procedimentos, é amplamente difundida a utilização da aprendizagem bayesiana (MITCHELL, 1997).
- **Aprendizado por esforço:** é baseado em dados de um ambiente completamente observável. Sua meta é aprender o quanto a política de classificação é boa, ou seja, descobrir a sua utilidade.

2.3.1 O método KNN (*K Nearest Neighbor*)

O método de *KNN* é um método de aprendizado de máquina supervisionado, baseado em instância, ou seja, um novo exemplo a ser classificado é comparado com os exemplos já conhecidos.

Quando surge uma nova instância para ser classificada, um conjunto de instâncias similares que estão armazenadas é recuperado e usado para classificação da nova instância. Tendo uma instância x como entrada, ao ser inserida num classificador *KNN*, serão encontrados os k vizinhos mais próximos, sendo que a classificação dessa nova instância x é realizada através do critério de maioria dos votos entre seus k vizinhos. O conceito de proximidade implica o uso de algumas métricas para o cálculo da similaridade entre as instâncias, e um exemplo básico é a distância euclidiana, representada pela Equação 2.1, onde D representa distância, p e q são os pontos.

$$D(p, q) = \sqrt{\sum_{i=0}^n (p_i - q_i)^2} \quad (2.1)$$

O algoritmo *KNN* calcula a distância entre a instância a ser classificada x e todas as instâncias que já pertencem ao conjunto de treinamento do classificador, seleciona as k instâncias

mais próximas em relação a x , e a classe atribuída a x será a classe mais comum entre os k vizinhos selecionados. A Figura 2 apresenta um exemplo do processo de classificação feita pelo algoritmo *KNN*, onde os pontos representam as instâncias da classificação e os pontos com a mesma cor pertencem à mesma classe. É importante notar que com a mudança de valor k , o resultado da classificação pode variar também.

Figura 2 – Exemplo de classificação utilizando algoritmo *KNN*

O método de *KNN* geralmente é aplicado com sucesso onde cada classe no conjunto de treinamento possui vários dos padrões possíveis, e a borda de decisão é muito irregular. Como o *KNN* precisa consultar todas as instâncias de treinamento para classificar uma nova instância, o processo de classificação tem complexidade alta, que torna o método caro computacionalmente para conjuntos de treinamento muito grandes.

Um dos maiores problemas do método de *KNN* é justamente a escolha do seu único parâmetro k , sendo que a melhor escolha para k muda para cada caso do problema. Geralmente, um valor grande para k reduz o efeito de ruídos (sobreposição de classes) na classificação, mas torna a borda de decisão entre as classes menos distintas. Na maioria das vezes, a escolha do valor de k é feita através de tentativa e erro.

2.3.2 O método *K-means*

O *K-means* é um método bastante utilizado na área de classificação e agrupamento de dados, que particiona automaticamente o conjunto de dados em k grupos. O funcionamento do método é basicamente escolhendo k centros de grupo e refinando os grupos iterativamente seguindo duas regras:

1. Cada instância d_i é atribuída ao seu grupo mais próximo;
2. Cada centro de grupo C_j é atualizado para ser a média das suas instâncias constituintes (WAGSTAFF *et al.*, 2001).

Um exemplo da aplicação de método *K-means* é ilustrado na Figura 3.

A parte (a) da Figura 3 representa um conjunto de dados distribuídos em um espaço bidimensional euclidiano. Os dois centros iniciais escolhidos são representados pelos símbolos

Figura 3 – Exemplo do algoritmo K-means

Fonte: Wagstaff *et al.* (2001).

X em azul e vermelho. Na parte (b), as instâncias de dados são classificadas em dois grupos diferentes, de acordo com a distância mais próxima aos centros X. Na parte (c), os centros de grupo são recalculados utilizando a média das suas instâncias. As partes (d), (e), (f), (g), (h) e (i) da Figura 3 mostram que o procedimento repete até a convergência final do algoritmo (BISHOP, 2006).

2.3.3 Aprendizado Bayesiano

Métodos de aprendizado bayesiano são baseados na abordagem probabilística. A ideia central destes métodos é considerar que os problemas são determinados por uma distribuição de probabilidade e as decisões ótimas podem ser tomadas através dessas probabilidades junto aos dados observados. Os benefícios entre várias decisões de classificação são quantificados usando probabilidade e custo envolvido e é considerado também que os problemas são definidos em termos probabilísticos e seus valores probabilísticos relevantes são conhecidos.

Segundo Mitchell (MITCHELL, 1997), o aprendizado bayesiano é atrativo por duas razões: os algoritmos de aprendizado bayesiano são práticos e permitem calcular a probabilidade de uma hipótese explicitamente; e provê base para o entendimento de outros métodos de classificação que não lidam com probabilidades diretamente. Dentre as outras características, pode-se destacar:

- O conhecimento armazenado no conjunto de treinamento pode ser combinado com novas instâncias para calcular a probabilidade final de uma hipótese;
- Novas hipóteses podem ser classificadas através da combinação de várias hipóteses, ponderadas pelas suas probabilidades;
- Cada novo dado classificado pode aumentar ou diminuir a probabilidade estimada de uma hipótese correta.

O aprendizado bayesiano tem como base o teorema de Bayes. Considere $P(i)$ a probabilidade de que o evento i ocorra e $P(j)$ a probabilidade de que o evento j seja observado, a probabilidade $P(i|j)$ denota a probabilidade de ocorrência do evento i dado a ocorrência do evento j . No aprendizado bayesiano, é fornecida uma maneira de calcular a probabilidade $P(i|j)$ dado as probabilidades $P(i)$, $P(j)$ e $P(j|i)$, e a relação entre essas probabilidades é ilustrada pela Equação 2.2.

$$P(i|j) = \frac{P(j|i)P(i)}{P(j)} \quad (2.2)$$

A equação de Bayes mostra que dada a ocorrência do evento j , pode-se converter a probabilidade $P(i)$ em uma probabilidade $P(i|j)$, que é a probabilidade de ocorrer o evento i dado a ocorrência do evento j , ou associando à classificação de dados, é a probabilidade de que a hipótese i ocorra dado o conjunto de treinamento j .

2.4 Cadeia de Markov

A Cadeia de Markov é um caso particular de processo estocástico com estados discretos e apresenta a propriedade Markoviana, onde o estado anterior é irrelevante para a predição do estado seguinte, desde que o estado atual seja conhecido. Sendo assim, os processos markovianos são caracterizados como sem memória e, por isso, tais processos possuem grande aplicabilidade no mundo real (MARKOV, 1971).

Ao considerar um processo estocástico como $\{X_t, t = 0, 1, 2, \dots\}$ que pode assumir um número finito de possíveis valores, e $X_t = i$ é um processo que está no estado i do instante t (ROSS, 2014). Dessa forma, uma cadeia de Markov é um processo estocástico definido como a Equação 2.3.

$$P\{X_{t+1} = j | X_t = i, X_{t-1} = i_{t-1}, \dots, X_1 = i_1, X_0 = i_0\} = P_{ij} \quad (2.3)$$

Para todos os estados i, j e $t \geq 0$, o P_{ij} é a probabilidade de transição do estado i para o estado j . Assim, o estado futuro X_{t+1} para uma cadeia de Markov é independente dos estados passados $X_0, X_1, \dots, X_{t-1}$ e o estado presente X_t (MARKOV, 1971).

A Figura 4 apresenta um exemplo clássico da Cadeia de Markov, onde os círculos q_0 e q_1 representam os estados do processo e as arestas representam as transições entre os estados com as suas respectivas probabilidades P .

A Cadeia de Markov pode ser visualizada também através de uma máquina de estado finito. Considerando um estado Y e outro estado X , a probabilidade de transição do estado Y no tempo T para o estado X no tempo $T + 1$ não depende de T mas sim do estado atual Y .

Figura 4 – Exemplo de uma cadeia de Markov. Adaptado de (MARKOV, 1971)

Fonte: Markov (1971).

No contexto desta pesquisa, a cadeia de Markov é utilizada para representar as relações entre os diferentes estados que um dispositivo móvel (estado da sua conexão) compõe.

2.5 Ferramentas de Simulação

É comum a utilização das simulações na área de grades computacionais, pois a dificuldade e a complexidade da implementação de plataformas reais para o estudo das mesmas são grandes. Nesta seção são apresentados alguns simuladores de grades computacionais estudados.

2.5.1 SimGrid

O *SimGrid* foi criado em 1999 por *Henri Casanova* para que fosse possível o estudo de algoritmos de escalonamento sem a necessidade de se implementar um sistema real para isso. Inicialmente, o *SimGrid* consistia em uma ferramenta de simulação orientada a eventos, permitindo construir simulações de aplicações específicas. O simulador implementava uma interface, ou Application Programming Interface (API), escrita em linguagem C, denominada *SG*. Com tal API, era possível especificar a simulação do escalonamento de tarefas em determinados recursos.

Os módulos do *SimGrid* apresentados na Figura 5 são detalhados a seguir:

- **Meta-SimGrid (MSG):** o primeiro ambiente de programação distribuído desenvolvido no *SimGrid*. Com foco na simulação como agentes de comunicação, omitindo vários detalhes técnicos da plataforma, pois o realismo da simulação não é o objetivo principal neste caso.

Figura 5 – Módulos do SimGrid

Fonte: Casanova (2001).

- **Grid Reality and Simulation (GRAS):** oferece a implementação de aplicações distribuídas em plataformas heterogêneas, favorecendo o desenvolvimento do programa usado em sistema distribuído;
- **Extended Bundle of Tools (XBT):** representa o núcleo de ferramentas no SimGrid, implementando diversas funcionalidades que auxiliam o módulo *SURF*, tais como estrutura de dados, serviço de autenticação e geração de grafos;
- **SMPI:** permite a simulação de programas Message Passing Interface (MPI) sem precisar de qualquer modificação;
- **SimDag:** fornece funcionalidades para simulação de tarefas de programação em que é possível especificar relações de dependências entre tarefas de programa paralelo;
- **SURF:** fornece as principais ferramentas para execução de simulação, servindo como base para todos outros módulos no SimGrid.

2.5.2 OptorSim

O *OptorSim* é projetado para ambientes de grade que utilizam o conceito de réplica de dados, ou seja, trata-se de um simulador voltado para grades de dados, nos quais as transferências de dados constituem um importante fator limitante do desempenho das tarefas executadas. A arquitetura do *OptorSim* é apresentada na Figura 6.

Com a arquitetura baseada no *EU DataGrid Project* (STOCKINGER *et al.*, 2003), a simulação é construída assumindo que a grade consiste de vários sites, e que cada um pode prover recursos computacionais e de armazenamento para os trabalhos submetidos. Cada site é composto

Figura 6 – Arquitetura do OptorSim

Fonte: Bell *et al.* (2003).

de zero ou mais elementos de computação ou zero ou mais elementos de armazenamento (BELL *et al.*, 2003).

O *OptorSim* utiliza três arquivos de configuração para efetuar a simulação. A plataforma da grade a ser simulada no *Optorsim* é especificada em um arquivo de configurações (*grid.conf*), o qual é lido pela aplicação. Neste arquivo descreve-se a topologia de cada site e detalhes de seus componentes dispostos em uma matriz como, por exemplo, o número de elementos computacionais do site, o número de armazenamento do site, o tamanho dos elementos de armazenamento em *megabytes* e a indicação de tamanho mínimo da largura de banda do canal.

As tarefas a serem executadas são indicadas no arquivo *jobs.conf*, no qual são especificadas informações sobre as tarefas e os arquivos utilizados por elas e a política de cada site. O último arquivo contém os parâmetros de entrada como, por exemplo, o caminho para os dois arquivos citados anteriormente, o tipo de usuário, a estratégia de escalonamento que será utilizada, o algoritmo de otimização, a distribuição inicial dos arquivos, o intervalo de tempo entre as submissões de tarefas, entre outras informações.

2.5.3 GridSim

O GridSim (BUYA; MURSHED, 2002) é um toolkit que propõe facilitar a simulação de diferentes classes de recursos heterogêneos, usuários, aplicações e escalonadores. Ele pode ser usado para simular escalonadores de tarefas de sistemas distribuídos como clusters e grades.

As principais funcionalidades e características da ferramenta são:

- Permite a modelagem de recursos heterogêneos;
- Permite o compartilhamento dos recursos;

- Uso do padrão estabelecido pelo *benchmarks Standard Performance Evaluation Corporation (SPEC)* para definir a capacidade dos recursos;
- Flexibilidade de tempo no gerenciamento de recursos (Mapeamento de feriados, identificação de fuso-horário);
- Simulação de Grades de dados;
- Permite especificação de diferentes topologias de rede;
- Sem limite de alocação de tarefas para qualquer recurso;
- Permite múltiplo acesso a um mesmo recurso no mesmo tempo;
- Permite a utilização de escalonador de tarefa;
- Possui métodos de análise estatística.

Figura 7 – Arquitetura do GridSim

Fonte: [Buyya e Murshed \(2002\)](#).

A arquitetura do *GridSim* é construída em forma de múltiplas camadas, como ilustrado na Figura 7. De baixo para cima, a primeira camada representa as interfaces entre o próprio simulador e a máquina virtual Java, com suporte aos sistemas que possuem múltiplos processadores. A segunda camada é construída pela infraestrutura de simulação orientada a eventos discretos. A terceira camada é composta pelos recursos e serviços de informação, voltados para a modelagem e a simulação das entidades básicas da grade. Nesta camada, encontram-se também a modelagem da aplicação e do acesso uniforme à interface, junto com um framework para a criação de entidades em alto nível. A quarta camada refere-se aos agregadores de recursos da simulação, chamado de *Grid Resource Brokers*. Os algoritmos de escalonamento de tarefas que este projeto está propondo se enquadram nessa camada. A camada superior na arquitetura do *GridSim* trabalha com as partes da aplicação e da modelagem dos recursos voltados para usuários, e faz uso dos serviços providos pelas camadas inferiores para a avaliação das políticas de escalonamento de tarefas e gerenciamento de recursos.

2.5.4 *MicroGrid*

O *MicroGrid* emula um ambiente de grade virtual, modelando a rede, os recursos e os serviços de informação de forma transparente. Com isso, objetiva-se proporcionar meios para a realização de experimentos controláveis e reproduzíveis de gerenciamento de recursos dinâmicos (XIA *et al.*, 2004).

A funcionalidade do *MicroGrid* é simular o comportamento de aplicações em ambiente de grade virtual. A ferramenta suporta aplicações que fazem uso da infraestrutura de *middleware* do *Globus Grid*. A arquitetura básica do *MicroGrid* é mostrada na Figura 8, cada componente na figura corresponde a um dos maiores desafios na construção de um grid virtual fiel a uma grade real, os recursos locais provêm um ambiente de recursos em uma grade virtual, e o simulador de rede é o responsável pelas interações entre os recursos locais virtualizados.

Figura 8 – Arquitetura do *MicroGrid*

Fonte: Song *et al.* (2000).

2.6 Considerações Finais

Neste capítulo, foram abordados os conceitos fundamentais para o projeto proposto nesta dissertação. Foi realizada a caracterização das grades móveis, a discussão de estratégias de escalonamento com foco na confiabilidade e tolerância a falhas em sistemas distribuídos, além disso, os métodos para a organização de recurso e as arquiteturas dos simuladores para grade computacional foram detalhados. No próximo capítulo, será apresentada a proposição dos algoritmos e simulações para o problema de escalonamento ciente da mobilidade dos dispositivos em grades móveis.

ESCALONAMENTO CIENTE DA MOBILIDADE DOS DISPOSITIVOS

3.1 Considerações Iniciais

Um ambiente de grade móvel é caracterizado pela sua alta heterogeneidade e volatilidade. Portanto, a escolha de uma política de escalonamento que considere essas características é muito importante para o desempenho geral do sistema. Dentre os vários problemas acarretados pela heterogeneidade e volatilidade, três deles são frequentemente discutidos: consumo de energia, conectividade e gerenciamento de recursos.

De modo a estudar como a conectividade influencia no desempenho de uma grade móvel, foi definido um ambiente de grade móvel, do tipo *on site*, onde o broker implementa um algoritmo de escalonamento que leva em consideração o nível de confiabilidade dos dispositivos móveis.

Este capítulo apresenta a arquitetura definida, a técnica utilizada para previamente classificar os dispositivos móveis de acordo com um histórico de conectividade, o mecanismo para controle de falhas e a simulação desenvolvida para analisar o desempenho da arquitetura.

3.2 Arquitetura do sistema de escalonamento

A arquitetura escolhida para ser utilizada neste projeto foi a que utiliza o modelo *on-site*, descrito no Capítulo 2, onde a entidade Broker é essencial para o controle da grade móvel. Além do modelo da grade, deve ser definido também o modelo das aplicações que serão executadas. Neste projeto, as aplicações são compostas por tarefas independentes, ou seja, elas não precisam ser executadas em uma ordem específica, o que facilita a definição da política de escalonamento. O tempo útil de bateria não influenciará a disponibilidade de recursos desde que seu nível esteja maior do que 30% da energia total. Caso contrário, esse fator entrará no algoritmo como uma variável que diminuirá a disponibilidade dos recursos ofertados pelo seu dispositivo.

Para a implementação da arquitetura proposta foram considerados dois fatores: a confiabilidade dos recursos e a tolerância a falha. Ambos fatores são representados por dois módulos RI (*Resource Information*) e RC (*Failure Control*), representados na Figura 9, a qual ilustra a visão geral da arquitetura e a diagrama na Figura 10 ilustra o fluxo do conjunto de algoritmos propostos.

Figura 9 – Modelo da arquitetura proposta

Figura 10 – Fluxo do conjunto de algoritmos propostos para o escalonamento de tarefas em uma grade móvel

Os recursos (*Resources*) são classificados e organizados em uma lista ordenada de acordo com sua confiabilidade. Cada recurso tem um valor inicial de confiabilidade, obtido com base na classificação que será explicada na Seção 3.3, e tem também uma probabilidade de falha. De acordo com essa probabilidade, o recurso pode falhar ou não e essa falha servirá de base para a atualização da confiabilidade, ou seja, os valores de confiabilidade são dinâmicos e o recurso perde o valor de confiabilidade quando falha e ganha o valor de confiabilidade caso contrário. Os mecanismos descritos são detalhados nas próximas seções.

3.3 Organização de recursos

Para se ter um valor inicial da confiabilidade dos recursos presentes na grade móvel, este trabalho propõe que seja utilizado um log que contenha informações sobre esses recursos, principalmente o tempo que o recurso permaneceu conectado na rede e a frequência de desconexão.

Essas informações são importantes para que o algoritmo de escalonamento já tenha um valor inicial para ser utilizado mais condizente com a realidade da grade móvel.

Um log de acesso da rede sem fio do Instituto de Ciências Matemáticas e de Computação (ICMC) foi considerado para extrair os dispositivos como recurso. Esse log utilizado possui aproximadamente 25000 registros de acesso à rede do Instituto, sendo que os usuários são identificados pelo seu endereço MAC. Além dessa informação, cada registro possui também o horário de conexão, horário de desconexão, tempo total de permanência na rede, quantidade de dados transmitidos e etc. A partir desse registro de acesso, é possível obter informações tais como: tempo total de permanência na rede em período de um mês, tempo de permanência na rede por dia e frequência de desconexão na rede.

Entre todos os registros de acesso a rede do ICMC considerados, foram extraídos aproximadamente 1800 dispositivos móveis e estes foram utilizados como parâmetros de entrada dos experimentos durante o desenvolvimento e simulação dos algoritmos propostos. Antes de iniciar o procedimento da classificação, os registros são transformados em formato adequado para a classificação, como ilustra a Figura 11. O tempo total de permanência no estado de conectado tem o minuto como unidade de tempo, e a frequência de desconexão é a quantidade total das conexões realizadas durante o período do tempo total de permanência na rede.

Figura 11 – Pré-processamento dos dados de registro

1	Client MAC Address	Association Time	Disassociation Time	Session Duration
2	00:07:abaf:05:d4	12/09/2012 21:00:56	12/09/2012 21:05:56	BR:5 min 0 sec
3	00:07:abaf:05:d4	12/09/2012 16:45:53	12/09/2012 16:50:53	BR:5 min 0 sec
4	00:07:abaf:05:d4	12/09/2012 16:25:52	12/09/2012 16:30:52	BR:5 min 0 sec
5	00:07:abaf:05:d4	12/09/2012 16:10:52	12/09/2012 16:15:52	BR:5 min 0 sec
6	00:07:abaf:05:d4	12/09/2012 15:55:52	12/09/2012 16:00:52	BR:5 min 0 sec
7	00:07:abaf:05:d4	12/09/2012 15:25:51	12/09/2012 15:45:51	BR:20 min 0 sec
8	00:07:abaf:05:d4	12/09/2012 15:10:51	12/09/2012 15:15:51	BR:5 min 0 sec
9	00:07:abaf:05:d4	12/09/2012 14:15:51	12/09/2012 15:05:51	BR:50 min 0 sec
10	00:07:abaf:05:d4	12/09/2012 13:55:51	12/09/2012 14:00:51	BR:5 min 0 sec
11	00:07:abaf:05:d4	12/08/2012 13:20:39	12/08/2012 13:25:39	BR:5 min 0 sec
12	00:07:abaf:05:d4	12/08/2012 13:15:39	12/08/2012 13:20:39	BR:5 min 0 sec
13	00:07:abaf:05:d4	12/08/2012 12:55:38	12/08/2012 13:15:39	BR:20 min 0 sec
14	00:07:abaf:05:d4	12/08/2012 12:35:38	12/08/2012 12:50:38	BR:15 min 0 sec
15	00:07:abaf:05:d4	12/08/2012 12:15:38	12/08/2012 12:20:38	BR:5 min 0 sec
16	00:07:abaf:05:d4	12/08/2012 11:55:38	12/08/2012 12:10:38	BR:15 min 0 sec
17	00:07:abaf:05:d4	12/08/2012 09:35:37	12/08/2012 09:40:37	BR:5 min 0 sec
18	00:07:abaf:05:d4	12/08/2012 09:30:37	12/08/2012 09:35:37	BR:5 min 0 sec
19	00:07:abaf:05:d4	12/07/2012 20:25:26	12/07/2012 20:40:26	BR:15 min 0 sec
20	00:07:abaf:05:d4	12/07/2012 19:50:25	12/07/2012 20:10:26	BR:20 min 0 sec
21	00:07:abaf:05:d4	12/07/2012 19:40:25	12/07/2012 19:45:25	BR:5 min 0 sec
22	00:07:abaf:05:d4	12/07/2012 19:30:25	12/07/2012 19:35:25	BR:5 min 0 sec
23	00:07:abaf:05:d4	12/07/2012 18:30:04	12/07/2012 18:35:04	BR:5 min 0 sec
24	00:07:abaf:05:d4	12/07/2012 18:24:55	12/07/2012 18:30:04	BR:5 min 9 sec
25	00:07:abaf:05:d4	12/07/2012 18:19:55	12/07/2012 18:24:55	BR:5 min 0 sec
26	00:07:abaf:05:d4	12/07/2012 18:14:54	12/07/2012 18:19:55	BR:5 min 0 sec
27	00:07:abaf:05:d4	12/07/2012 17:54:54	12/07/2012 17:59:54	BR:5 min 0 sec

⇒

	Recurso	Frequencia de desconexao	Tempo total de permanencia
1	user1	26	825
2	user2	17	135
3	user3	32	1265
4	user4	8	460
5	user5	16	650
6	user6	9	355
7	user7	31	1475
8	user8	47	2245
9	user9	19	895
10	user10	27	1120
11	user11	21	1365
12	user12	51	2895
13	user13	42	2750
14	user14	33	1965
15	user15	16	780
16	user16	15	865
17	user17	27	2250
18	user18	19	1795
19	user19	9	955
20	user20	6	75
21	user21	23	1555
22	user22	21	1780
23	user23	25	2235
24	user24	12	685
25	user25	4	125
26	user26	57	1675
27	user27	64	3275
28	user28	17	1125
29	user29	8	765
30	user30	16	565
31	user31	28	1635
32	user32	33	2200
33	user33	31	1980

Tendo os dados do registro de acesso pré-processado, é aplicado o algoritmo KNN para o processo de classificação. Considerando o tempo total de permanência e a frequência de desconexão, os dispositivos são classificados em três grupos pelas suas confiabilidades, como por exemplo um grupo de dispositivos com alta confiabilidade, um grupo de dispositivos com média confiabilidade e um grupo de dispositivos com baixa confiabilidade. Dado o tempo total de permanência como t e a frequência de desconexão como f , a distância entre o recurso x e o

recurso y é dado pela Equação 3.1.

$$D(x, y) = \sqrt{(t_x - t_y)^2 + (f_x - f_y)^2} \quad (3.1)$$

Para otimizar o resultado obtido, o estado de conexão e desconexão dos dispositivos é visto como um modelo da Cadeia de Markov, apresentado na Figura 12. A probabilidade de um dispositivo permanecer conectado é representado por $P(c)$ e a probabilidade de permanecer desconectado é representado por $P(d)$, $P(cd)$ e $P(dc)$ representam as probabilidades das transições de estado.

Figura 12 – Transição de estado dos dispositivos

Figura 13 – Reestruturação dos recursos

É possível calcular a probabilidade $P(X)$ de conexão e desconexão dos dispositivo no determinado instante X de acordo com a fórmula markoviana definida na Equação (3.2). As

variáveis $P(cc)$, $P(cd)$, $P(dc)$ e $P(dd)$ representam as probabilidades de um dispositivo móvel permanecer conectado, desconectar-se, conectar-se e permanecer desconectado respectivamente. Com as probabilidades dos estados de conexão no instante X e instante $X + 1$ calculados, os três grupos já classificados posteriormente são ajustados de acordo com a seguinte regra: dispositivos com valor de P_{dd} maior que P_{cc} perde a confiabilidade e os dispositivos com P_{dd} menor que P_{cc} ganha confiabilidade como a Figura 13 ilustra, e os valores de confiabilidade dos dispositivos classificados são atualizados, formando a lista final dos recursos ordenados de acordo com a confiabilidade.

$$P = P_{cd} = \begin{vmatrix} P_{cc} & P_{cd} \\ P_{dc} & P_{dd} \end{vmatrix} \text{ and } P(X) = P_0 \cdot P^X \quad (3.2)$$

Desse modo, calculando as probabilidades de desconexão dos dispositivos dentro de um determinado período, as posições dos dispositivos já classificados são reajustadas, onde os dispositivos com grande quantidade de desconexões perdem sua confiabilidade, mesmo tendo maior tempo de acesso, e os dispositivos com pequena quantidade de desconexões ganham confiabilidade, mesmo tendo menor tempo de acesso. Em outras palavras, a Cadeia de Markov reestrutura o conjunto de dados processados pelo algoritmo *KNN*, otimizando a lista de recursos de acordo com a métrica de confiabilidade.

No fim do processo de organização dos recursos, os dispositivos móveis que representam os recursos são registrados em forma de lista ordenada de acordo com os dispositivos do grupo de alta confiabilidade, média confiabilidade e baixa confiabilidade. Os dispositivos do mesmo grupo não passam por nenhuma outra métrica de ordenação, são simplesmente colocados na lista pela sua ordem original, pois estes são considerados iguais em questão de confiabilidade. O algoritmo 1 representa o processo descrito acima.

3.4 Controle de falha

Tendo a lista de recurso criada, é muito importante manter as informações nela atualizadas, pois o estado de conexão dos recursos varia constantemente, e o responsável por manter a lista de recursos atualizada é o módulo FC localizado no *Broker* da arquitetura (Figura 9).

O processo da atualização da lista de recurso funciona de acordo com uma regra básica: dispositivos ganham confiabilidade quando executam tarefas com sucesso e perdem confiabilidade quando falham. O Algoritmo 2 mostra o funcionamento da atualização de confiabilidade dos recursos, onde a falha do dispositivo causa a perda de confiabilidade em escala exponencial e uma conclusão de tarefa faz com que o dispositivo ganhe a confiabilidade em escala linear.

Ao atingir o valor mínimo de confiabilidade que é representado pelo valor 0, o recurso é descartado da lista de recursos. No caso de aparecimento de um novo recurso, ele pode ser colocado no topo da lista com o valor máximo de confiabilidade, ou no meio da lista com o

Algoritmo 1: Algoritmo para organização de recurso**Input:** A set of resource devices $\langle d_1, d_2, \dots, d_n \rangle$ **Output:** An ordered list of resource devices $\langle r_1, r_2, \dots, r_n \rangle$

```

1  $i \leftarrow 1$ 
2  $j \leftarrow 1$ 
3  $list \leftarrow New\_resource\_list$ 
4 while  $i \leq n$  do
5    $KNN \leftarrow d_i$ 
6    $i \leftarrow i + 1$ 
7 while  $j \leq n$  do
8   if  $d_j.P(cc) < d_j.P(dd)$  then  $d_j.group \leftarrow d_j.group - 1$ 
9    $j \leftarrow j + 1$ 
10
11   if  $d_j.P(cc) > d_j.P(dd)$  then  $d_j.group \leftarrow d_j.group + 1$ 
12    $j \leftarrow j + 1$ 
13
14  $list \leftarrow sort(group3 + group2 + group1)$ 
15 return  $list$ 

```

Algoritmo 2: Atualização de confiabilidade**Input:** A set of reliability value $\langle a_1, a_2, \dots, a_n \rangle$ and failure quantity $\langle b_1, b_2, \dots, b_n \rangle$ **Output:** A set of reliability value $\langle a_1, a_2, \dots, a_n \rangle$

```

1  $i \leftarrow 0$ 
2 if  $failure$  then while  $i < n$  do
3   if  $b_i = 0$  then  $a_i \leftarrow a_i - 1$ 
4
5   if  $b_i \leq 2$  then  $a_i \leftarrow a_i - b_i$ 
6
7   if  $b_i \leq 5$  then  $a_i \leftarrow a_i - 2b_i$ 
8
9   if  $b_i > 5$  then  $a_i \leftarrow a_1 - b_i * b_i$ 
10
11    $i \leftarrow i + 1$ 
12 return set of reliability value  $\langle a_1, a_2, \dots, a_n \rangle$ 
13
14 else
15   while  $i < n$  do
16      $a_i \leftarrow a_i + 1$ 
17      $i \leftarrow i + 1$ 
18   return set of reliability value  $\langle a_1, a_2, \dots, a_n \rangle$ 

```

valor 50 de confiabilidade, dependendo da exigência do sistema empregado na taxa de acerto das execuções de tarefa.

Além desses métodos descritos, a técnica de tolerância a falhas *Replication* foi utilizada para testar sua influência no desempenho do sistema, pois o tratamento sobre tarefas não sucedidas no modo padrão desta arquitetura proposta é simplesmente atribuir novamente esta tarefa para o primeiro dispositivo da lista de recurso.

Algoritmo 3: Execução do algoritmo de escalonamento

Input: A set of Task $\langle t_1, t_2, \dots, t_n \rangle$ and A set of Resource devices $\langle d_1, d_2, \dots, d_n \rangle$

Output: A set of Task results $\langle r_1, r_2, \dots, r_n \rangle$

```

1  $i \leftarrow 1$ 
2  $j \leftarrow 1$ 
3  $RI \leftarrow \text{Classification } \langle d_1, d_2, \dots, d_n \rangle$ 
4 while  $i \leq n$  do
5   if  $\text{Execution}(t_i, d_j) = \text{False}$  then  $\text{Failure}_j \leftarrow \text{Failure}_j + 1$ 
6    $j \leftarrow j + 1$ 
7
8   else
9      $r_i \leftarrow \text{Result}(t_i, d_j)$ 
10     $i \leftarrow i + 1$ 
11  if  $i \% w = 0$  then  $RI \leftarrow \text{Algoritmo3 } \langle d_1, d_2, \dots, d_n \rangle$ 
12
13 return set of Task results  $\langle r_1, r_2, \dots, r_n \rangle$ 

```

Com todas as técnicas descritas acima, um procedimento para a execução deste sistema começa pelo tratamento dos dispositivos usados como recursos da grade móvel. Os recursos são classificados e ordenados de acordo com suas confiabilidades pelos métodos *KNN* e Cadeia de Markov, antes de serem salvos no módulo RI (*Resource Information*). Tendo os recursos registrados e ordenados no RI, as tarefas já podem ser alocadas nos recursos obedecendo a regra do escalonador, e durante o tratamento das tarefas, o módulo FC que cuida do controle de falha atualiza constantemente a lista de recurso no RI, de acordo com seu comportamento no tratamento das tarefas, sempre mantendo os recursos mais confiáveis no topo da lista. A representação desse processo é ilustrada pelo Algoritmo 3, onde a variável w representa a quantidade de execução para a próxima atualização de confiabilidade. Neste caso a regra de escalonamento é FIFO e para cada 5 tarefas realizadas com sucesso, a lista de recurso é atualizada, ou seja $w = 5$.

3.5 Simulação

Devido à dificuldade e à complexidade de implementação de plataformas reais para o estudo das grades móveis, é comum a utilização de simulações de maneira a validar uma arquitetura ou um novo algoritmo. O simulador *GridSim* foi escolhido para realização das simu-

lações neste projeto, por ter funcionalidades mais completas e também devido a sua facilidade de alteração comparando com os outros simuladores.

Para a simulação do algoritmo de escalonamento na grade móvel, foi necessária a implementação do mecanismo que simula o estado de desconexão dos dispositivos móveis na classe *gridsim.resFailure* do simulador, pois o GridSim foi desenvolvido para grades computacionais tradicionais que não possuem todas características da grade móvel.

O valor de confiabilidade para cada dispositivo de recurso é definido através do *log* de acesso mencionado na Seção 3.3. De acordo com os dados históricos de utilização dos recursos, um valor entre 1 e 100 é atribuído a cada dispositivo de recurso como o valor de confiabilidade. Este valor de confiabilidade é usado como a probabilidade de conexão sem falha no simulador, ou seja, um recurso com o valor 99 na confiabilidade tem apenas 1 por cento de chance para acontecer uma falha, e um recurso com o valor 60 na confiabilidade tem 40 por cento de chance para falhar.

Figura 14 – Alterações realizadas no simulador *GridSim*

Os métodos de classificação e ordenação dos recursos foram implementados separadamente do simulador *GridSim*, devido seu grande custo no tempo de execução, que pode sobrecarregar a realização dos testes de simulação para validação dos resultados quando a quantidade inicial de recurso é bem grande. O algoritmo para a ordenação da lista de recurso é composto pelo classificador *KNN* e Markoviano, que foi detalhado na seção anterior junto com o Algoritmo 1.

O resultado do processo de organização de recurso é representado por uma lista de dispositivos ordenados pelos seus valores de confiabilidade, e é encaminhada para *GridSim* como os parâmetros dos recursos, salvos no módulo RI (*Resource Information*), foi necessário

a implementação e adaptação na classe *GridResource* para a leitura desta lista de recurso. Os métodos para o controle de falha dos recursos como atualização de confiabilidade, adição de novo recurso e exclusão de recurso sem confiabilidade foram implementados e incorporado no módulo *Grid Resource Broker* do simulador. As alterações realizadas no simulador *GridSim* podem ser vistas na Figura 14.

3.6 Considerações Finais

Neste capítulo, foram detalhados os algoritmos propostos. Entre eles o algoritmo para a organização de recurso composto por algoritmo KNN e modelo da cadeia de Markov, que é executado antes das simulações, e o algoritmo para o controle de falha, executado durante o processo de escalonamento. Além disso, as informações para realização das simulações foram esclarecidas, e os planejamentos para os conjuntos de simulação são detalhados no próximo capítulo.

RESULTADOS

4.1 Considerações Iniciais

Como descrito no Capítulo 3, foi proposta uma arquitetura que tem como objetivo estudar o impacto de considerar confiabilidade dos recursos presentes na grade móvel para auxiliar no escalonamento das tarefas. De modo a atingir esse objetivo, foram propostos 5 conjuntos de experimentos utilizando a simulação descrita na Seção 3.5. Todos os resultados foram obtidos através das modificações introduzidas no simulador *GridSim*. Foi introduzido o módulo Controle de Falha, onde é implementado o mecanismo de atualização da confiabilidade, com uma frequência de atualização da lista de recurso a cada 5 execuções. A classificação e a ordenação dos recursos são realizadas separadamente das simulações, devido a complexidade da integração no simulador e o custo de tempo que dificulta a realização de simulações com muitos recursos. Para iniciar uma simulação, um conjunto de recursos são selecionados primeiramente para passar no procedimento de classificação e ordenação, formando uma lista ordenada de recursos, com seus respectivos valores de confiabilidade, e a lista de recurso é interpretada como dados de entrada para a simulação, servindo como base de informações para o módulo RI.

4.2 Planejamento dos Conjuntos de Experimentos

Foram planejados e realizados 5 conjuntos de experimentos, de modo a validar e avaliar os algoritmos propostos. Os primeiros testes são relativamente mais simples, com poucos parâmetros, carga de trabalho e recursos envolvidos para testar as funcionalidades parciais, analisar o comportamento e custo da cada parte do algoritmo. O Quadro 2 apresenta as variáveis do primeiro conjunto de experimentos.

Duas variáveis de respostas são considerados: tempo total de execução e quantidade de falhas detectadas. Com 3 fatores e 2 níveis e utilizando um planejamento fatorial completo, foram definidos 8 experimentos (Quadro 3), e cada experimento foi executado 30 vezes, totalizando

Quadro 2 – Parâmetros para o primeiro conjunto de experimentos

Fatores	Nível 1	Nível 2
Carga de trabalho	50	100
Qtde de recursos	30	50
Tolerância a falha	Sim	Não

240 execuções. O algoritmo de tolerância a falha utilizado é o de replicação de tarefas, sendo que o objetivo da sua presença nesse primeiro conjunto de experimento é comparar o seu custo com o custo do algoritmo proposto, pois tanto o algoritmo de classificação e ordenação de recurso quando o algoritmo de controle de falha não deixam de ser um mecanismo de tolerância às falhas também.

Quadro 3 – Detalhes para o primeiro conjunto de experimentos

Carga de trabalho	Qtde de recursos	Tolerância a falha
50	30	Sim
50	30	Não
50	50	Sim
50	50	Não
100	30	Sim
100	30	Não
100	50	Sim
100	50	Não

No segundo conjunto de experimentos, o foco está na variação da carga de trabalho. A quantidade de recursos foi fixada em 100 e a carga de trabalho foi variada de 25 para 500, como ilustra o Quadro 4. Para cada variação da carga de trabalho foram realizadas 30 execuções, totalizando 210 execuções.

Quadro 4 – Detalhes para o segundo conjunto de experimentos

Carga de trabalho	Qtde de recursos	Variável de resposta
25	100	Tempo total / Qtde de falha
50	100	Tempo total / Qtde de falha
100	100	Tempo total / Qtde de falha
200	100	Tempo total / Qtde de falha
300	100	Tempo total / Qtde de falha
400	100	Tempo total / Qtde de falha
500	100	Tempo total / Qtde de falha

O terceiro conjunto de experimentos realizado tem os mesmos parâmetros que o anterior, mas foi adicionado um outro fator que é o conjunto de dados que irá formar a lista de recursos. O Quadro 5 mostra as informações básicas deste conjunto de simulação, onde as listas de recurso 1, 2 e 3 correspondem a, respectivamente, primeiro a centésimo dispositivo, centésimo primeiro a ducentésimo dispositivo e ducentésimo primeiro a tricentésimo dispositivos extraídos do *log* do ICMC.

A variável de resposta X é calculada a partir de quantidade total de falha dividido pelo tempo total de execução, e o objetivo da utilização da taxa X é ajudar na visualização do desempenho do sistema e também facilitar a comparação dos resultados nas simulações mais complexas.

Quadro 5 – Detalhes para o terceiro conjunto de experimentos

Conjunto de simulação	Carga de trabalho	Lista de recurso	Variável de resposta
1	25 - 500	1	X
2	25 - 500	2	X
3	25 - 500	3	X

No quarto conjunto de experimentos, o foco está na variação da quantidade de recursos. A quantidade da carga de trabalho foi fixada em 200 e a quantidade de recursos foi variada de 25 a 500. O Quadro 6 apresenta os parâmetros deste conjunto, sendo que cada experimento foi repetido 30 vezes, resultando em 210 execuções, como no segundo conjunto de simulação.

Quadro 6 – Detalhes para o quarto conjunto de experimentos

Carga de trabalho	Qtde de recursos	Variável de resposta
200	25	Tempo total / Qtde de falha
200	50	Tempo total / Qtde de falha
200	100	Tempo total / Qtde de falha
200	200	Tempo total / Qtde de falha
200	300	Tempo total / Qtde de falha
200	400	Tempo total / Qtde de falha
200	500	Tempo total / Qtde de falha

O quinto e último conjunto de experimentos é semelhante ao primeiro conjunto, porém foram alterados os valores dos parâmetros "Carga de Trabalho" e "Qtde de recursos" de modo a obter informações mais abrangentes, como por exemplo as influências de fatores nos experimentos realizados. Nesse caso não foi considerado nenhuma técnica tradicional de tolerância a falha como replicação, pois as tarefas são submetidas novamente caso seu recurso falhe durante a execução da tarefa, e a função principal do próprio mecanismo de controle de falha é minimizar a quantidade de falha. Os Quadros 7 e 8 apresentam as variáveis envolvidas neste conjunto de experimentos. Por fim, este conjunto de experimentos foi repetido duas vezes, uma vez com o mecanismo de controle de falha e outra vez sem, com objetivo de analisar as influências deste algoritmo proposto no processo da execução de tarefa.

Quadro 7 – Parâmetros para o quinto conjunto de experimentos

Fatores	Níveis	Variáveis de resposta
Carga de trabalho	200/500	Tempo total/Qtde de falha
Qtde de recursos	100/600	Tempo total/Qtde de falha

Na próxima Seção são apresentados os resultados obtidos através das simulações de acordo com estes planejamentos.

Quadro 8 – Detalhes de execução para o quinto conjunto de experimentos

Carga de trabalho	Qtde de recursos	Qtde de execução
200	100	50
200	600	50
500	100	50
500	600	50

4.3 Análise dos resultados

Como já explicado, o custo do algoritmo de classificação e ordenação dos recursos é bem alto, e por causa disso, esse processo foi realizado antes do início das simulações, de modo que esse tempo não sobrecarregasse as simulações.

No caso do primeiro conjunto de experimentos, o qual considerou poucos recursos e tarefas, apesar de ter quase dobrado o tempo total de execução, a replicação de tarefas ainda tem um custo bem menor que o algoritmo da organização de recurso proposto, mas isso acontece devido à pequena quantidade de tarefas e recursos considerados nesta simulação.

O algoritmo de organização tem custo elevado devido a sua complexidade, e é influenciado pela quantidade de recursos, mas o aumento do seu custo é logarítmico de acordo com o aumento da quantidade de recursos.

Figura 15 – Primeiro conjunto de experimentos - Tempo total de execução em relação a carga de trabalho

Nas Figuras 15 e 16 são apresentados os resultados das 240 execuções em relação ao tempo total de execução e quantidade de falhas, onde o variável *N* representa a não utilização da replicação de tarefa e o variável *R* representa a utilização da replicação de tarefa. O aumento da

Figura 16 – Primeiro conjunto de experimentos - Quantidade de falha em relação a carga de trabalho

carga de trabalho teve maior impacto na quantidade de falha do que o tempo total de execução. Isso acontece devido a pequena quantidade de recursos considerados neste caso. As próximas simulações mostram mais detalhes das influências causadas pelos parâmetros considerados.

Os resultados do segundo conjunto de experimentos são apresentados nas Figuras 17, 18 e 19, onde a carga de trabalho varia de 25 para 500 e a quantidade de recursos permanece em 100.

Figura 17 – Segundo conjunto de experimentos - Quantidade de falha em relação ao crescimento da carga de trabalho

A quantidade de falhas é pequena quando a carga de trabalho é baixa como mostra a Figura 17. Isso acontece devido a grande quantidade de recursos disponíveis, o que permitiu a alocação dos recursos com melhor confiabilidade para estas tarefas. Com o aumento no nível da carga de trabalho, a quantidade de falhas cresce junto e começa a estabilizar a partir de

Figura 18 – Segundo conjunto de experimentos - Tempo total de execução em relação ao crescimento da carga de trabalho

Figura 19 – Segundo conjunto de experimentos - Taxa X (Falha/Tempo) em relação ao crescimento da carga de trabalho

300 tarefas aproximadamente, pois neste momento a quantidade de tarefas executadas superaram bastante da quantidade de recursos, e o algoritmo do controle de falha já eliminou os recursos de baixa qualidade.

O aumento do tempo mostrado na Figura 18 ao longo do crescimento na carga de trabalho é esperado, pois o tempo de execução é diretamente proporcional a quantidade de tarefas a serem processadas. A quantidade de falhas aumenta gradativamente e começa a estabilizar no meio da simulação, isso acontece devido a lista ordenada de recurso já estar bem otimizada pelo algoritmo de controle de falhas, e é importante notar que parte dos recursos foram descartados da lista por terem perdido totalmente seu valor de confiabilidade.

Na Figura 19 é apresentada a taxa X. Uma interpretação simples da taxa X é: quando menor o valor de X, melhor o desempenho do mecanismo do controle de falha, desde que o aumento do tempo total de execução é linear. Caso ao contrário, o grande custo no tempo de execução prejudica ou até anula o ganho de desempenho promovido pelo sistema do controle de falhas.

O terceiro conjunto de experimentos tem seu planejamento exatamente igual ao conjunto de experimentos anterior, com a diferença que os recursos do *log* utilizados em cada repetição

Figura 20 – Terceiro conjunto de experimentos - Tempo de execução dos 3 Logs em relação ao crescimento da carga de trabalho

pertencem a faixas diferentes da ordenação. O objetivo desse experimento era confirmar se o comportamento dos resultados do segundo conjunto de experimentos permanecia utilizando diferentes conjuntos de recursos. A Figura 20 apresenta uma visão geral do tempo de execução e da quantidade de falhas para cada nível de carga de trabalho, considerando o intervalo de confiança de 95%. Pode-se observar que o tempo de execução aumenta proporcionalmente com o aumento da quantidade de tarefas de maneira semelhante nos 3 logs utilizados. O comportamento da quantidade de falhas também é o mesmo nos 3 logs, estabilizando a quantidade de falhas a partir de 300 tarefas, como pode ser observado na Figura 21. Isso mostra que os recursos que falharam frequentemente já foram penalizados, chegando ao valor de confiabilidade 0, e foram eliminados.

O quarto conjunto de experimentos tem seu planejamento semelhante ao segundo conjunto de experimentos, porém, ao invés de variar a carga de trabalho, foi variada a quantidade de recurso de 25 para 500, com carga de trabalho fixada em 200.

A partir da Figura 22, é possível observar que o tempo de execução chega ao pico com 200 recursos e volta a diminuir logo depois, quando se tem mais do que 200 recursos. Quando se tem uma grande quantidade de recursos, entre 400 e 500, os primeiros recursos que atenderam as tarefas submetidas são classificados como alta confiabilidade no topo da lista de recursos, e a quantidade de falhas é baixa pelo mesmo motivo. No caso de pequenas quantidades de recursos, entre 25 e 50, os recursos são processados rapidamente pelo algoritmo de controle de falha e todos os recursos que sobram na lista são de ótima confiabilidade. É importante notar que

Figura 21 – Terceiro conjunto de experimentos - Quantidade de falha dos 3 logs em relação ao crescimento da carga de trabalho

quando maior a quantidade de recurso em relação à quantidade de tarefas, menor o tempo total de execução.

A Figura 23 mostra a relação entre a quantidade de falhas e o tempo de execução ao longo da variação na quantidade de recursos, representado pela variável X. A queda do valor de X com o aumento da quantidade de recursos significa a melhoria no desempenho do sistema, devido a grande quantidade de recursos com alta confiabilidade disponível e que são colocados no topo da lista de recurso para priorizar a sua utilização.

No último conjunto de experimentos foram definidos como fatores a carga de trabalho de 200 e 500 tarefas, e a quantidade de recursos entre 100 e 600, de modo a ter uma avaliação mais abrangente com quantidades maiores de recursos e carga de trabalho. Considerando o planejamento fatorial de 2 fatores e 2 níveis, as combinações das simulações atendem os casos:

Figura 22 – Quarto conjunto de experimentos - Tempo de execução e quantidade de falhas

Figura 23 – Quarto conjunto de experimentos - Relação entre quantidade de falha e tempo de execução

maior carga de trabalho com menor quantidade de recurso, menor carga de trabalho com maior quantidade de recurso, como já foi apresentado no Quadro 8.

Na Figura 24 são apresentadas as comparações dos resultados de todas as combinações planejadas neste conjunto de experimentos. É possível observar que a grande quantidade de recursos ajuda na redução do tempo de execução e da quantidade de falhas, e consequentemente no desempenho geral do sistema representado pela variável X (menor o valor de X, melhor o desempenho). No caso dos resultados com carga de trabalho com 500 tarefas, apesar de ter menor tempo de execução e quantidade de falhas quando a quantidade de recurso é 600, o desempenho do sistema é estatisticamente igual.

Para analisar melhor o peso dos fatores considerados neste experimento, foi construído

Figura 24 – Quinto conjunto de experimentos - Comparações de quantidade de falha, tempo de execução e variável X

Figura 25 – Quinto conjunto de experimentos - Influência dos fatores nos resultados em relação ao tempo de execução

o gráfico da influência de fatores representado nas Figuras 25 e 26. As variáveis A, B e AB representam a carga de trabalho, quantidade de recurso e os dois fatores juntos respectivamente. A linha azul no meio do gráfico representa a linha normal, onde as variáveis mais afastadas da linha normal possuem maior influência de fator, ou seja, a alteração nos fatores mais afastados da linha normal gera mais impacto no resultado.

Figura 26 – Quinto conjunto de experimentos - Influência dos fatores nos resultados em relação à quantidade de falha

Neste caso do último conjunto de experimento, a variável B, que representa a quantidade de recursos, apresentou maior influência tanto na variável de resposta tempo de execução quanto quantidade de falhas.

Para a segunda repetição do último conjunto de experimento, que não foi empregado o mecanismo de controle de falha, foram realizadas as comparações com a primeira repetição em questão de tempo total de execução e a quantidade de falhas, mostradas pelas Figuras 27 e 28.

No caso do tempo total de execução (Figura 27), a segunda repetição apresentou um tempo total de execução um pouco menor pois não executou o processamento da parte do controle de falha e da atualização de confiabilidade. A diferença do tempo de execução é relativamente pequena neste conjunto de experimentos, mas dependendo da frequência de atualização da confiabilidade considerada no algoritmo, esta diferença pode ser bem maior se a variável w do Algoritmo 3 for menor que 5.

Para a quantidade de falhas (Figura 28), a segunda repetição apresentou uma quantidade bem maior em comparação com a primeira repetição, principalmente no caso com 500 tarefas e 100 recursos. Apesar de não ter utilizado o controle de falha na segunda repetição, o mecanismo da organização de recurso ainda oferece uma lista de recursos ordenada pelas suas confiabilidades, por isso o aumento da quantidade de falha não é tão grande na maioria dos casos. Quando a quantidade de tarefa é muito maior que a quantidade total de recursos, os recursos com baixa confiabilidade não são eliminados da lista por falta do mecanismo de controle de falha, que permanecem prejudicando a conclusão das tarefas.

Figura 27 – Comparação do tempo total de execução das 2 repetições do último conjunto de experimento

Figura 28 – Comparação da quantidade de falha das 2 repetições do último conjunto de experimento

As simulações realizadas validaram o modelo implementado, provendo as funcionalidades na organização de recursos para ajudar o sistema de escalonamento. Tanto o mecanismo da organização de recurso quanto o controle de falha têm seus custos significativos de execução, principalmente para a organização de recurso no caso da aplicação de modelo no sistema com grande conjunto de recurso inicial, mas estes custos serão compensados ao longo da execução

do sistema.

4.4 Considerações finais

O mecanismo implementado neste projeto engloba as funcionalidades de escalonamento de tarefas, organização de recurso e tolerância a falha. O processo da organização de recurso é muito custoso para sistemas de grande porte, mas pode ser realizado previamente, e a falta deste processo pode sobrecarregar a função do controle de falha que acaba prejudicando o desempenho do sistema inteiro.

A frequência da atualização de lista de recursos em todas as simulações realizadas foi a cada 5 execução de tarefa, e esta frequência pode ser alterada para adaptar melhor a situação de cada sistemas. Para um sistema com boa qualidade de recursos, uma pequena frequência da atualização pode ajudar bastante na economia do tempo total de execução, mas no caso do sistemas com baixa qualidade de recursos, a maior frequência da atualização elimina rapidamente os recursos de baixa qualidade e agiliza o processamento das tarefas.

Em comparação aos trabalhos relacionados, a utilização do conceito de confiabilidade não é novidade, mas a intenção do trabalho realizado não é somente desenvolver e avaliar um algoritmo de escalonamento, e também propor um modelo de organização de recurso e controle de falha para o escalonamento de tarefas na grade móvel, pois os módulos propostos são bem flexíveis. Caso algum algoritmo apresenta melhor desempenho nos trabalhos futuros, ele pode substituir o algoritmo do módulo atual sem precisar alterar algoritmo do outro módulo.

CONCLUSÕES

Durante o estudo e o desenvolvimento para este projeto, notou-se a importância de considerar a confiabilidade dos dispositivos no escalonamento de tarefas em uma grade móvel, pois a confiabilidade dos dispositivos móveis tem papel muito importante para o desempenho geral do sistema. Nesse caso, observou-se a necessidade da organização de recursos para garantir a confiabilidade dos dispositivos móveis alocados para o processamento das tarefas. Foi proposto o algoritmo de escalonamento baseado no classificador *KNN* com auxílio do modelo da cadeia de Markov para a organização dos recursos de acordo com sua confiabilidade antes do processamento de tarefa. Durante a execução das tarefas, o algoritmo de controle de falha foi aplicado para manter a lista de recursos atualizada de acordo com a métrica de confiabilidade definida junto com o algoritmo proposto, e este mecanismo dinâmico assegura a qualidade dos dispositivos móveis presentes na lista de recurso.

A avaliação dos algoritmos propostos foi realizada através de simulação, e os experimentos realizados também permitiram a validação de funcionalidade e eficiência dos algoritmos variando os parâmetros e cenários. Além disso, os resultados obtidos através das simulações foram promissores, evidenciando a viabilidade dos algoritmos desenvolvidos, sendo que dois deles se destacam: o algoritmo de organização dos recursos faz a seleção dos dispositivos móveis para ter uma lista de recursos ordenado de acordo com sua confiabilidade; e o algoritmo de controle de falha registra as falhas acontecidas durante a execução, descartando os dispositivos com baixa confiabilidade e priorizando os dispositivos com alta confiabilidade para a alocação das tarefas.

5.1 Contribuições

As principais contribuições deste projeto são apresentadas a seguir:

- O desenvolvimento da arquitetura de escalonamento de tarefa na grade móvel considerando

os algoritmos propostos.

- A proposta e implementação do algoritmo de classificação baseado no algoritmo *KNN* e modelo da cadeia de Markov para o processo da organização de recurso.
- A proposta e implementação do algoritmo de controle de falha para o dinamismo da lista de recursos, mantendo os dispositivos com alta confiabilidade no topo da lista.
- Modificação e adaptação do simulador *Gridsim* para realização das simulações com intuito de avaliar e validar o modelo de escalonamento e os algoritmos propostos.

5.2 Trabalhos futuros

Em um modelo de escalonamento na grade móvel com compartilhamento de recursos relativamente instáveis comparando com a grade tradicional, vários fatores que influenciam o desempenho do sistema não puderam ser considerados nesta pesquisa por causa do tempo e/ou escopo. Tais possíveis propostas são apresentadas a seguir:

- Propor um mecanismo de classificação de tarefas considerando os parâmetros como o tamanho de tarefa e a taxa de transmissão de dados, organizando as tarefas na entrada para a alocação de recursos de acordo com as informações de tarefa visando minimizar o desperdício do poder de processamento da grade móvel.
- Melhorar o algoritmo de escalonamento proposto considerando demais fatores, tais como o nível de bateria dos dispositivos móveis e desligamento do aparelho pelo fornecedor de recurso.
- Desenvolver um gerenciador de parâmetros que facilite as alterações nos atributos como a frequência de atualização da lista de recurso, o limite de confiabilidade para o descarte de recurso, a posição de entrada na lista do novo recurso e entre outras.
- Analisar os impactos da utilização do modelo de escalonamento proposto na grade móvel de pequeno porte e na grade móvel de grande porte respectivamente, em questão do custo de tempo principalmente.
- Expandir o modelo proposto com o módulo de gestão do consumo de energia para os dispositivos móveis.

REFERÊNCIAS

BELL, W. H.; CAMERON, D. G.; MILLAR, A. P.; CAPOZZA, L.; STOCKINGER, K.; ZINI, F. Optorsim: A grid simulator for studying dynamic data replication strategies. **International Journal of High Performance Computing Applications**, SAGE Publications, v. 17, n. 4, p. 403–416, 2003. Citado na página [20](#).

BISHOP, C. M. **Pattern recognition and machine learning**. [S.l.]: springer, 2006. Citado na página [16](#).

BUY YA, R.; ABRAMSON, D.; VENUGOPAL, S. The grid economy. **Proceedings of the IEEE**, IEEE, v. 93, n. 3, p. 698–714, 2005. Citado na página [1](#).

BUY YA, R.; MURSHED, M. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. **Concurrency and computation: practice and experience**, Wiley Online Library, v. 14, n. 13-15, p. 1175–1220, 2002. Citado 3 vezes nas páginas [6](#), [20](#) e [21](#).

CASANOVA, H. Simgrid: A toolkit for the simulation of application scheduling. In: IEEE. **Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on**. [S.l.], 2001. p. 430–437. Citado na página [19](#).

CASAVANT, T. L.; KUHLE, J. G. A taxonomy of scheduling in general-purpose distributed computing systems. **Software Engineering, IEEE Transactions on**, IEEE, v. 14, n. 2, p. 141–154, 1988. Citado 3 vezes nas páginas [11](#), [8](#) e [9](#).

CHITRA, P.; VENKATESH, P.; RAJARAM, R. Comparison of evolutionary computation algorithms for solving bi-objective task scheduling problem on heterogeneous distributed computing systems. **Sadhana**, Springer, v. 36, n. 2, p. 167–180, 2011. Citado na página [12](#).

DONG, F.; AKL, S. G. **Scheduling algorithms for grid computing: State of the art and open problems**. [S.l.], 2006. Citado na página [8](#).

GOMATHI, S.; MANIMEGALAI, D. An adaptive grouping based job scheduling in grid computing. In: IEEE. **Signal Processing, Communication, Computing and Networking Technologies (ICSCCN), 2011 International Conference on**. [S.l.], 2011. p. 192–196. Citado na página [10](#).

HUANG, C.-Q.; ZHU, Z.-T.; WU, Y.-H.; XIAO, Z.-H. Power-aware hierarchical scheduling with respect to resource intermittence in wireless grids. In: IEEE. **Machine Learning and Cybernetics, 2006 International Conference on**. [S.l.], 2006. p. 693–698. Citado na página [11](#).

HUANG, Y.; BESSIS, N.; NORRINGTON, P.; KUONEN, P.; HIRSBRUNNER, B. Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm. **Future Generation Computer Systems**, Elsevier, v. 29, n. 1, p. 402–415, 2013. Citado na página [10](#).

JANG, S. H.; LEE, J. S. Mobile resource reliability-based job scheduling for mobile grid. **KSII Transactions on Internet & Information Systems**, v. 5, n. 1, 2011. Citado na página 11.

JOHN, S. M.; MOHAMED, M. M. Workflow scheduling algorithm for grid services with hierarchical grouping based qos. **Asian Journal of Applied Sciences**, v. 8, n. 1, p. 1–15, 2015. Citado na página 13.

KATSAROS, K.; POLYZOS, G. C. Optimizing operation of a hierarchical campus-wide mobile grid for intermittent wireless connectivity. In: IEEE. **Local & Metropolitan Area Networks, 2007. LANMAN 2007. 15th IEEE Workshop on**. [S.l.], 2007. p. 111–116. Citado 2 vezes nas páginas 5 e 7.

LUO, W.; QIN, X.; BELLAM, K. Reliability-driven scheduling of periodic tasks in heterogeneous real-time systems. In: IEEE. **Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on**. [S.l.], 2007. v. 1, p. 778–783. Citado na página 12.

MARKOV, A. Extension of the limit theorems of probability theory to a sum of variables connected in a chain. John Wiley & Sons, Inc., 1971. Citado 3 vezes nas páginas 11, 17 e 18.

MITCHELL, T. M. Machine learning. 1997. **Burr Ridge, IL: McGraw Hill**, v. 45, 1997. Citado 2 vezes nas páginas 14 e 16.

NASIR, H. J. A.; KU-MAHAMUD, K. R. Grid load balancing using ant colony optimization. **Computer and Network Technology, International Conference on**, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 207–211, 2010. Citado na página 10.

PARK, S.-M.; KO, Y.-B.; KIM, J.-H. Disconnected operation service in mobile grid computing. In: **Service-Oriented Computing-ICSOC 2003**. [S.l.]: Springer, 2003. p. 499–513. Citado 3 vezes nas páginas 2, 7 e 11.

PHAN, T.; HUANG, L.; DULAN, C. Challenge: Integrating mobile wireless devices into the computational grid. In: ACM. **Proceedings of the 8th annual international conference on Mobile computing and networking**. [S.l.], 2002. p. 271–278. Citado na página 5.

QI, P.; LI, L. A fault recovery-based scheduling algorithm for cloud service reliability. **Security and Communication Networks**, Wiley Online Library, v. 8, n. 5, p. 703–714, 2015. Citado na página 13.

QIU, W.; ZHENG, Z.; WANG, X.; YANG, X. An efficient fault-tolerant scheduling algorithm for periodic real-time tasks in heterogeneous platforms. In: IEEE. **Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2013 IEEE 16th International Symposium on**. [S.l.], 2013. p. 1–7. Citado na página 13.

RAM, K.; RAO, R. S. M.; GOVARDHAN, A. A reliable distributed grid scheduler for independent tasks. Citeseer, 2011. Citado na página 12.

RODRIGUEZ, J. M.; ZUNINO, A.; CAMPO, M. Introducing mobile devices into grid systems: a survey. **International Journal of Web and Grid Services**, Inderscience, v. 7, n. 1, p. 1–40, 2011. Citado 3 vezes nas páginas 1, 2 e 6.

ROSS, S. M. **Introduction to probability models**. [S.l.]: Academic press, 2014. Citado na página 17.

RUSSELL, S. J.; NORVIG, P. **Inteligencia Artificial: un enfoque moderno**. [S.l.: s.n.], 1996. Citado na página [13](#).

SENTHILNATH, J.; OMKAR, S.; MANI, V. Clustering using firefly algorithm: performance study. **Swarm and Evolutionary Computation**, Elsevier, v. 1, n. 3, p. 164–171, 2011. Citado na página [9](#).

SONG, H. J.; LIU, X.; JAKOBSEN, D.; BHAGWAN, R.; ZHANG, X.; TAURA, K.; CHIEN, A. The microgrid: a scientific tool for modeling computational grids. In: IEEE. **Supercomputing, ACM/IEEE 2000 Conference**. [S.l.], 2000. p. 53–53. Citado na página [22](#).

STOCKINGER, H.; DONNO, F.; LAURE, E.; MUZAFFAR, S.; KUNSZT, P.; ANDRONICO, G.; MILLAR, P. Grid data management in action: Experience in running and supporting data management services in the eu datagrid project. **arXiv preprint cs/0306011**, 2003. Citado na página [19](#).

WAGSTAFF, K.; CARDIE, C.; ROGERS, S.; SCHRÖDL, S. *et al.* Constrained k-means clustering with background knowledge. In: **ICML**. [S.l.: s.n.], 2001. v. 1, p. 577–584. Citado 2 vezes nas páginas [15](#) e [16](#).

XHAFA, F.; ABRAHAM, A. Computational models and heuristic methods for grid scheduling problems. **Future generation computer systems**, Elsevier, v. 26, n. 4, p. 608–621, 2010. Citado na página [8](#).

XIA, H.; DAIL, H.; CASANOVA, H.; CHIEN, A. *et al.* The microgrid: Using online simulation to predict application performance in diverse grid network environments. In: IEEE. **Challenges of Large Applications in Distributed Environments, 2004. CLADE 2004. Proceedings of the Second International Workshop on**. [S.l.], 2004. p. 52–61. Citado na página [22](#).

YANG, X.-S. **Nature-inspired metaheuristic algorithms**. [S.l.]: Luniver press, 2010. Citado 2 vezes nas páginas [9](#) e [12](#).

YOUSIF, A. Scheduling jobs on grid computing using firefly algorithm. **Journal of Theoretical and Applied Information Technology**, Asian Research Publishing Network (ARPN), v. 33, n. 2, p. 155–164, 2011. Citado na página [9](#).