UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Eliza Helena Areias Gomes

UMA ABORDAGEM DE RESERVA ANTECIPADA DE RECURSOS EM AMBIENTES OPORTUNISTAS

Florianópolis

2013

Eliza Helena Areias Gomes

UMA ABORDAGEM DE RESERVA ANTECIPADA DE RECURSOS EM AMBIENTES OPORTUNISTAS

Dissertação submetida ao Programa de Pós Graduação em Ciência da Computação para a obtenção do Grau de Mestre em Ciência da Computação.
Orientador: Prof. Dr. Mario Antonio Ribeiro Dantas

Florianópolis

2013

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Gomes, Eliza Helena Areias
Uma Abordagem de Reserva Antecipada de Recursos em Ambientes Oportunistas / Eliza Helena Areias Gomes ; orientador, Mario Antonio Ribeiro Dantas - Florianópolis, SC, 2013.
111 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da Computação. 3. Sistema de Computação. 4. Escalonamento. 5. Computação Oportunista. I. Dantas, Mario Antonio Ribeiro. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

Eliza Helena Areias Gomes

UMA ABORDAGEM DE RESERVA ANTECIPADA DE RECURSOS EM AMBIENTES OPORTUNISTAS

Esta Dissertação foi julgada aprovada para a obtenção do Título de "Mestre em Ciência da Computação", e aprovada em sua forma final pelo Programa de Pós Graduação em Ciência da Computação.

Florianópolis, 26 de setembro 2013.

Prof. Dr. Ronaldo dos Santos Mello
Coordenador do Curso

Banca Examinadora:

Prof. Dr. Mario Antonio Ribeiro Dantas
Orientador

Prof(a). Dra. Alba Cristina Magalhães Alves de Melo
Universidade de Brasília

Prof. Dr. Rômulo Silva de Oliveira
Universidade Federal de Santa Catarina

Prof. Dr. Frank Augusto Siqueira
Universidade Federal de Santa Catarina

Dedico este trabalho de mestrado à Terezinha, Luiza e Carlos, que estiveram e sempre estarão ao meu lado me apoiando em qualquer fase de minha vida.

AGRADECIMENTOS

Primeiramente, gostaria de agradecer a Deus pela oportunidade a mim proporcionada. Pois, sem Seus feitos, não teria ingressado neste programa de pós-graduação e, consequentemente, conquistado tudo que conquistei nesses dois anos.

Meus sinceros agradecimentos ao meu orientador professor Dr. Mario Dantas, que acreditou na minha capacidade intelectual em um momento que foi desprezada. E ainda, pelas oportunidades a mim proporcionadas e por ter me ajudado a crescer intelectual e academicamente. Agradeço pelo companheirismo, dedicação e paciência (e ficarei esperando pela publicação de seu novo livro).

Agradeço a minha mãe Terezinha por todo apoio e dedicação, pelas vindas ao laboratório de madrugada para me acompanhar, pelos jantares prontos me esperando, enfim, pelo companheirismo que me deu nessa fase de minha vida e que tem dado minha vida toda. Agradeço também à minha irmã Luiza Helena que esteve e está ao meu lado me apoiando em todos os momentos. Ao meu namorado e designer Carlos Eduardo por fazer e arrumar repetidas vezes as figuras desta dissertação. Agradeço a ele também, por ter estado e por ainda estar ao meu lado, mesmo nos momentos de estresse e mau humor. Aos meus tios Marli e João Carlos pelo suporte que me deram em um momento difícil dessa fase.

Aos meus amigos e colegas do laboratório, que mesmo depois de muita análise, descobrir que o problema era uma linha de código comentada, ainda estavam dispostos a me ajudar na implementação do meu projeto. Pelos pequenos momentos de descontração, de questionamentos e discussões intrigantes como: será que a formiga sente cheiro?, como é feito o leite em pó?, quando eu era pequeno..., na época do disquete.... Não podendo esquecer é claro, de agradecer aos amigos do laboratório vizinho, da sala 312, a secretária Katiana pois, estiveram sempre dispostos e com paciência para me ajudar e ao professor Dr. Roberto Willrich por emprestar as máquinas que foram utilizadas para o desenvolvimento do protótipo.

Agradeço novamente ao professor Dr. Roberto Willrich pela oportunidade de trabalhar e participar do SBBD e WebMedia 2011, bem como ao meu orientador por me proporcionar a participação no WSCAD 2012.

Por fim, gostaria de agradecer ao professor Dr. Alfredo Goldman, da

USP, pela sua dedicação em discutir parte deste trabalho. Ao professor Dr. Renato Cerqueira e ao Dr. Renato Maia, da PUC-Rio, pela ajuda com a linguagem Lua e OiL e à todos os integrantes do suporte do InteGrade, que com muita dedicação e paciência resolveram os problemas e dúvidas recorrentes durante a implementação da proposta.

Quando tudo nos parece dar errado, acontecem coisas boas que não teriam acontecido se tudo tivesse dado certo.

Renato Russo

RESUMO

A grade computacional é muito utilizada quando se deseja alto desempenho para resolução de problemas que requerem alto poder de processamento. As grades oportunistas, um tipo de grade computacional, possuem o diferencial de utilizar recursos computacionais ociosos de máquinas pessoais para a resolução destes problemas, o que torna esse ambiente mais barato e, consequentemente, mais interessante, principalmente para a comunidade acadêmica. No entanto, nos ambientes oportunistas a disputa por recursos torna-se maior devido à instabilidade e constante uso de seus recursos. Problemas como o excesso de solicitações de alocação de recursos em um mesmo período podem ser recorrentes, o que pode tanto prejudicar o desempenho do sistema quanto tornar o processo de solicitação de recursos trabalhoso para o usuário, uma vez que este terá que repetir o processo até que haja recursos disponíveis para a sua execução. Uma maneira eficiente de resolver tal problema é com a utilização da reserva antecipada de recursos. Este mecanismo permite que o usuário selecione um conjunto de recursos para que sejam utilizados em um período no futuro, considerando oportunisticamente os recursos disponíveis. Diante disso, esta dissertação propôs a utilização do mecanismo de reserva antecipada em um ambiente de grade oportunista. O objetivo foi melhorar a vazão do uso de recursos oportunistas, de modo a oferecer a possibilidade de alocação dos recursos durante um longo período de tempo, e não apenas no momento da solicitação. Estudos de caso foram realizados para ilustrar o comportamento de um ambiente oportunista com a abordagem proposta, bem como para comparar os ambientes que utilizam e não utilizam reserva antecipada. Os resultados mostraram a eficiência e validade da utilização de tal abordagem em um ambiente distribuído.

Palavras-chave: Grade Oportunista; Reserva Antecipada de Recursos; Alocação de Recursos; Escalonamento Distribuído.

ABSTRACT

Grid computing is widely used when high performance is desired to resolve problems that require high processing power. Opportunistic grids, a type of grid computing system, have the differential to use idle computing resources of personal machines to solving these problems, making the solution cheaper and consequently more interesting, specially for the academic community. However, opportunistic environments have more competition for resources due to instability and constant use of their resources. Problems such as excessive requests of resources allocation can be recurrent and can both degrade the performance of the system and make the process of resource requests harder to user, once the user will have to repeat this process until there are available resources to the execution. An efficient way to solve this problem is through the use of advanced reservation of resources. This mechanism allows user to select a set of resources to be used in the future opportunistically considering available resources. Therefore, this dissertation proposed the use of the advanced reservation mechanism in an opportunistic grid environment. The goal was to improve the use flow of the opportunistic resources in order to offer the resources allocation possibility for a long period of time and not only at the time of request. Case studies were carried out to illustrate the behaviour of an opportunistic environment with the proposed approach, as well as to compare environments that use with others that do not use advanced reservation. The results show the efficiency and validity of the approach in a distributed environment.

Keywords: Opportunistic Grid; Advanced Reservation of Resource; Resource Allocation; Distributed Scheduling.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo é apresentada a contextualização e o problema do tema de pesquisa. Na seção de pergunta de pesquisa é apresentada a hipótese para resolução do problema, bem como sua justificativa. A partir da pergunta de pesquisa e da hipótese são determinados os objetivos desta pesquisa. Por fim, são apresentados os métodos utilizados para a realização desta dissertação.

1.1 CONTEXTUALIZAÇÃO

Grade computacional pode ser entendida como um ambiente de alto desempenho que compartilha serviços e recursos geograficamente distribuídos para solucionar problemas que exigem alto poder computacional (DANTAS, 2005). Com o amadurecimento desse ambiente, tem-se criado diferentes tipos de infraestrutura de grade, dentre essas a grade oportunista.

Grade oportunista, por sua vez, pode ser conceituada como uma infraestrutura de grade que utiliza recursos computacionais de máquinas não dedicadas, geralmente de máquinas pessoais. A motivação de se utilizar esse tipo de infraestrutura, segundo Foster e Kesselman (2011), é que ciclos computacionais não utilizados representam grande quantidade de computação. A grade oportunista pode ser subdividida em duas categorias (PONCIANO; BRASILEIRO, 2012): Grade *Desktop* e Grade *Peer-to-Peer*. Na primeira categoria os recursos são doados e seus donos não esperam nenhum retorno. Como exemplo para esse tipo de grade pode-se citar: SETI@home (SETI@HOME, 2013), BOINC (BOINC, 2013) e InteGrade (INTEGRADE, 2012). Já a grade *Peer-to-Peer* possui o diferencial que os provedores de recursos esperam que seus favores (disponibilização de seus recursos computacionais) sejam retribuídos, como por exemplo, maior prioridade na execução de sua aplicação. Um exemplo desse tipo de grade é o *middleware* OurGrid (OURGRID, 2013).

O sistema de gerenciamento de recursos (RMS - *Resource Management System*) tem o papel importante no gerenciamento das tarefas e recursos geograficamente distribuídos (DANTAS, 2005). Assim, o usuário submete uma aplicação ao *software* gerenciador de recursos e esse, por sua vez, a escalona da melhor maneira na configuração disponível. No entanto, sistemas de grade oportunista que permitem que o usuário solicite a execução de sua aplicação podem sofrer sobrecarga e degradação, uma vez que poderá haver, por um lado, períodos com grandes quantidades de requisições e, por outro, longos períodos de ociosidade, ou seja, sem requisições. Além disso, o

processo de solicitação de execução pode se tornar trabalhoso para o usuário, uma vez que esse, em um período de sobrecarga do sistema, poderá ter que realizar repetidas requisições, até que ache a quantidade de recursos necessários disponíveis para executar sua aplicação.

Diante disso, é proposto neste trabalho de pesquisa estudar um modelo que contemple uma melhor alocação de recursos, de maneira oportunista. Por se tratar de uma grade oportunista, deve-se considerar a dinamicidade do ambiente. Assim, propõe-se implementar um protótipo que possa indicar a adequação do modelo no ambiente experimental.

1.2 PERGUNTA DE PESQUISA

Este trabalho de pesquisa busca responder a seguinte pergunta: **Como melhorar a vazão de uso de recursos computacionais ociosos em um ambiente distribuído?**

Para responder a essa pergunta é sugerido a utilização da reserva antecipada de recursos para ambientes distribuídos. O conceito de vazão, no contexto desta dissertação, pode ser entendido como a distribuição da utilização dos recursos computacionais no decorrer do tempo. Diante disso, a reserva antecipada pode melhorar a vazão de uso de recursos pois esse mecanismo possibilita que usuários reservem recursos computacionais para que sejam utilizados no futuro. Com isso, as execuções podem iniciar em um determinado período de tempo no futuro e não apenas no momento da solicitação de execução. Além disso, as execuções estariam organizados em fila esperando a hora de se iniciarem. Desta forma, aumentaria a organização e a previsibilidade do sistema, uma vez que é possível saber antecipadamente quando e onde a aplicação será executada.

1.3 OBJETIVOS

De acordo com a pergunta de pesquisa descrita acima, esta pesquisa apresenta os objetivos geral e específicos descritos nas próximas subseções.

1.3.1 Objetivo Geral

O objetivo geral desta dissertação é investigar uma abordagem que possibilite uma melhor utilização dos recursos computacionais de forma oportunista para a execução de aplicações paralelas e distribuídas.

1.3.2 Objetivos Específicos

Dentre os principais objetivos específicos pode-se citar:

- Identificar um modelo capaz de melhorar a vazão de uso de recursos em ambientes distribuídos;
- Propor uma arquitetura de que melhore a distribuição da utilização dos recursos computacionais em ambientes oportunistas;
- Desenvolver um mecanismo de busca por intervalos livres capaz de realocar os pedidos de reserva para o período de tempo mais próximo do escolhido pelo usuário;
- Propor um serviço que auxilie o usuário na escolha do melhor período para solicitar a reserva;
- Avaliar o funcionamento da arquitetura proposta por meio do desenvolvimento de um protótipo;
- Demonstrar a aplicabilidade do protótipo em estudos de caso;

1.4 MÉTODO DE PESQUISA

As etapas realizadas para alcançar os objetivos desta dissertação foram:

- Pesquisa do estado da arte nas áreas de reserva antecipada de recursos e grades oportunistas;
- Estudo da arquitetura e funcionamento do *middleware* utilizado como ambiente para desenvolvimento do protótipo;
- Implementação do protótipo baseado no modelo proposto;
- Aplicação do protótipo em um ambiente de *cluster* para a realização de estudos de caso;
- Análise dos resultados gerados a partir dos estudos de caso.

1.5 ORGANIZAÇÃO DO TRABALHO

Esta dissertação está subdividida em 6 capítulos. No *capítulo 2 - Sistemas Distribuídos* é realizado um estudo geral sobre sistemas distribuídos, apresentado os diferentes tipos de arquiteturas e softwares utilizados nesse tipo de ambiente.

O *capítulo 3 - Escalonamento em Ambientes Oportunistas* apresenta conceitos fundamentais de reserva antecipada, meta-escalonamento, grades oportunistas e o ambiente utilizado para o desenvolvimento da arquitetura proposta. Além disso, são destacados alguns trabalhos relacionados à área de pesquisa.

O *capítulo 4 - Reserva Antecipada Oportunista* apresenta o modelo e a arquitetura propostos, bem como os detalhes da implementação do modelo e os algoritmos desenvolvidos.

O *capítulo 5 - Ambiente e Resultados Experimentais* descreve o ambiente utilizado para a obtenção dos resultados experimentais, bem como os estudos de casos realizados como prova do funcionamento do protótipo proposto.

O *capítulo 6 - Conclusões e Trabalhos Futuros* apresenta as conclusões sobre a pesquisa realizada, bem como as indicações para trabalhos futuros.

2 SISTEMAS DISTRIBUÍDOS

Na década de 90, verificou-se a disponibilização de conexões de rede mais rápidas, e computadores pessoais tornaram-se configurações mais potentes. Diante desse quadro, a interconexão de computadores, local ou geograficamente distribuídos, torna-se uma forma a maximizar o poder computacional. A partir de então, popularizam-se as chamadas configurações dos sistemas distribuídos.

Para Coulouris, Dollimore e Kindberg (2007), sistema distribuído é o sistema no qual os componentes de hardware ou software, localizados em computadores interconectados por uma rede, comunicam-se e coordenam-se através da troca de mensagens.

No conceito de Tanenbaum e Steen (2002), um sistema distribuído é uma coleção de computadores independentes que se apresenta como um único sistema aos seus usuários. Para esses autores, a construção de um ambiente distribuído deve atender aos seguintes requisitos:

- **Tornar os recursos acessíveis**: facilitar para o usuário o acesso a recursos remotos e possibilitar o compartilhamento de forma controlada e eficiente;
- **Transparência na distribuição**: esconder do usuário o fato de que os processos e recursos estão fisicamente distribuídos em vários computadores;
- **Abertura**: oferecer para o usuário serviços de acordo com as regras padrão que descrevem sua sintaxe e semântica;
- **Escalabilidade**: para atender a esse requisito o ambiente distribuído deve adicionar facilmente usuários e recursos no sistema, permitir que usuários e recursos estejam distantes e ser de fácil gerenciamento, mesmo que se estenda por organizações administrativamente independentes.

Diante disso, este capítulo tem como objetivo o detalhamento da arquitetura e dos componentes de hardware e software necessários para a construção de uma infraestrutura distribuída.

2.1 ARQUITETURAS COMPUTACIONAIS

A classificação da arquitetura de computadores mais aceita é conhecida como taxonomia de Flynn. Proposta por Michael J. Flynn em

1966 (FLYNN, 1972), esta taxonomia leva em consideração a quantidade de instruções executadas em paralelo pelo conjunto de dados para os quais as instruções são submetidas. A Figura 1 apresenta as classificações de computadores, que são subdivididas em: SISD, SIMD, MISD e MIMD (FLYNN, 1972).

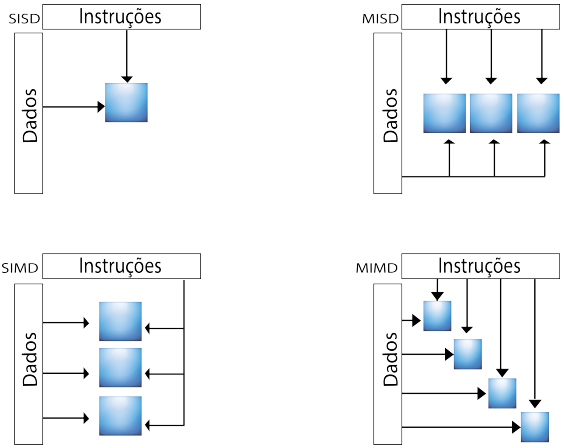

Figura 1 – Taxonomia de Flynn

- **SISD** (*Single Intruction Single Data*): executa uma instrução de um programa por vez, ou seja, são computadores que apresentam apenas um processador. Este modelo pode ser representado pelo computador pessoal, onde as instruções são executadas sequencialmente.
- **SIMD** (*Single Intruction Multiple Data*): executa uma única instrução, porém esta instrução é processada sob diferentes itens de dados. Isso ocorre devido à existência de facilidades de hardware para armazenamento de dados, como vetores e *array*.
- **MISD** (*Multiple Intruction Single Data*): neste tipo de arquitetura são executadas múltiplas instruções sob um único conjunto de dados. No entanto, não se tem conhecimento da existência de computadores com essa classificação.
- **MIMD** (*Multiple Intruction Multiple Data*): executa múltiplas instruções sob múltiplos dados. Computadores com esta arquitetura possuem múltiplos processadores em que cada um pode executar instruções independentemente dos demais.

As arquiteturas MIMD são classificadas, de modo geral, como multiprocessadores e multicomputadores, nas quais se diferenciam pelo compartilhamento ou não de memórias (MEFFE; MUSSI; MELLO, 2006).

- **Multiprocessadores**: arquitetura conhecida como fortemente acoplada, uma vez que processadores e memórias são interligados através de um sistema local de interconexão. Estas arquiteturas são caracterizadas por possuírem diversos processadores compartilhando uma ou um conjunto de memórias. A escalabilidade de uma arquitetura multiprocessada chega a milhares de processadores.
- **Multicomputadores**: nesta configuração cada processador possui suas próprias memórias locais. Diante disto, é conhecida como fracamente acoplada, uma vez que não há um compartilhamento forte, ou seja, a comunicação entre processos é efetuada através de troca de mensagens entre os processos que estão executando nos processadores.

2.2 CLUSTERS

A ideia inicial de *cluster* computacional foi desenvolvida nos anos 1960 pela IBM como uma forma de conectar grandes *mainframes* para prover um paralelismo comercial de baixo custo (BUYYA, 1999).

Diante disso, os *clusters*, conhecidos também em português como agregados computacionais, são definidos por Buyya (1999) como um sistema de processamento paralelo ou distribuído formado por uma coleção de computadores interconectados que trabalham juntos como um único recurso computacional integrado.

Por outro lado, Dantas (2005) entende que as configurações de *clusters* podem ser como uma agregação de computadores de forma dedicada, ou não, para a execução de aplicações específicas de uma organização.

Um *cluster* é composto por dois ou mais computadores (também chamados de nós) com um único processador ou por sistemas multi-processados, interconectados por uma rede. Seu principal objetivo é realizar o processamento da aplicação de forma distribuída e transparente. Em outras palavras, é compartilhar o processamento das aplicações entre os nós, mas de forma que pareça estar sendo processado em um único computador.

Segundo Buyya (1999) e Dantas (2005) os *clusters* podem ser classificados de acordo com limite geográfico, utilização dos nós, tipo de hardware, aplicação alvo, sistema operacional dos nós e tipos de nós.

- **Limite Geográfico**: baseado em sua localização e quantidade, os *clusters* podem ser classificados como: pequenos, constituídos em sa-

las e laboratórios; médios, em nível de departamento; grande, para organizações.

- **Aplicação Alvo**: existem dois alvos principais às aplicações comuns: as aplicações que necessitam de um alto desempenho para sua execução e as aplicações que precisam de alta disponibilidade. Aplicações que procuram o primeiro alvo se preocupam com o número de processadores, quantidade de memória e espaço em disco. Por outro lado, as que tem foco no segundo alvo são caracterizadas por não tolerarem interrupções. Entretanto, é possível observar que existem aplicações que exigem os dois tipos de requisitos.

- **Utilização dos Nós**: pode ser estabelecida através da participação não dedicada ou dedicada dos nós que irão compor o *cluster*. Nos *clusters* não dedicados as aplicações são executadas por meio da utilização dos ciclos de CPU ociosos de máquinas pertencentes ao *cluster*. Por outro lado, os dedicados são projetados para executarem exclusivamente aplicações submetidas ao *cluster*. A Figura 2 apresenta um exemplo das configurações de *cluster* não dedicada e dedicada. Como pode ser visto, na configuração não dedicada convencional inúmeros computadores compartilham um único meio de comunicação, que é a rede local. Além disso, cada computador possui um conjunto de aplicativos locais e periféricos necessários para a execução das tarefas de seu usuário local. Já na configuração dedicada pode-se ver que os nodos são interligados por um dispositivo de rede do tipo *switch* e nenhum computador dispõe de monitor, teclado ou mouse, mostrando assim sua dedicação à execução de aplicações.

- **Tipo de Hardware**: os tipos de hardware empregados nos *clusters* são classificados como: Nows, CoPs ou PoPs, Cows e Clumps.

  As Nows (*Network of Workstations*) são caracterizadas pelo uso de estações de trabalho distribuídas em uma rede local para compor um ambiente de *cluster*. De forma similar às Nows, os CoPs (*Cluster of PCs*) ou PoPs (*Pile of PCs*) utilizam computadores pessoais do tipo PCs para formar o *cluster*.

  Configurações Cows (*Cluster of Workstations*) são geralmente constituídas por máquinas dedicadas e homogêneas à execução de aplicações específicas. Além disso, estas dispõem de uma rede específica para interconexão das máquinas.

  Os Clumps (*Cluster of SMPs*) são compostos de máquinas com arquiteturas SMP (*Symmetric Multiprocessors*) , que usam memória compartilhada.

- **Tipos de Nós**: os *clusters* podem ser classificados quanto à similaridade de software e hardware dos nós que compõem o ambiente, ou seja, podem ser homogêneos ou heterogêneos.

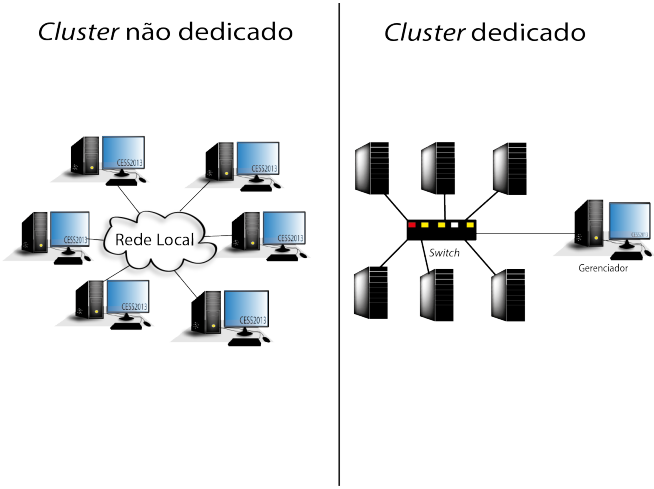

Figura 2 – Exemplo de ambiente de *cluster* não dedicado e dedicado

Nos *clusters* homogêneos todos os nós possuem arquiteturas semelhantes e executam o mesmo sistema operacional. Por outro lado, os heterogêneos possuem nós com arquiteturas diferentes e executam sistemas operacionais diferentes.

Entretanto, muitas vezes, um único *cluster* não possui capacidade de resolver problemas computacionais que demandam grande poder computacional. Neste caso, múltiplos *clusters* interconectados, formam um ambiente *multi-cluster* e, consequentemente, podem aumentar o poder computacional para a resolução desses problemas. A seção seguinte apresenta uma descrição mais detalhada desses ambientes.

2.3 MULTI-CLUSTER

Conforme visto na seção anterior, os *clusters* possuem a localização restrita a um único domínio. No entanto, é possível, e muitas vezes necessário, que *clusters* de múltiplos domínios se interconectem para formar um sistema distribuído único de larga escala. Estes sistemas são conhecidos

como *multi-cluster* (FERREIRA, 2010).

Diferentemente dos *clusters*, que são formados por um conjunto independente de estações de trabalho interconectadas por uma rede local (LAN), os *multi-clusters* são formados por um conjunto de *clusters* interconectados por uma rede WAN (*World-Area Network*).

Os sistemas *multi-cluster* podem ser classificados como super-*cluster* e *cluster*-de-*cluster* (JAVADI; AKBARI; ABAWAJY, 2006). O primeiro pode ser caracterizado por possuir um grande número de processadores homogêneos e heterogeneidade nas redes de comunicação. Por outro lado, o *cluster*-de-*cluster* é construído pela interconexão de múltiplos *clusters*, porém com heterogeneidade tanto nas redes de comunicação quanto nos processadores.

Existe um alto grau de complexidade nos ambientes *multi-cluster* quanto aos problemas de escalonamento de tarefas, pois seus recursos são: heterogêneos, distribuídos e altamente compartilhados no tempo e no espaço. Diante disso, o recebimento contínuo de tarefas e as mudanças dinâmicas da disponibilidade da capacidade de CPU dificultam a utilização de algoritmos tradicionalmente utilizados em sistemas de *cluster* (ABAWAJY; DANDAMUDI, 2003).

A configuração de sistemas *multi-cluster* forma um sistema de imagem única (SSI - *Single System Image*), ou seja, fornece ao usuário uma visão unificada do compartilhamento de recursos e serviços do ambiente (DANTAS, 2005).

A Figura 3 apresenta um exemplo de ambientes *multi-cluster*.

2.4 GRADES COMPUTACIONAIS

Na década de 90, com a maior disponibilidade de conexões mais rápidas e com aplicações necessitando cada vez mais de poder computacional, pesquisadores começaram a imaginar uma infraestrutura computacional que conectasse computadores e *clusters* geograficamente distribuídos para atender essa demanda. Diante disso, surgiu a proposta da chamada grade computacional, do inglês *computational grid* (FOSTER; KESSELMAN, 2011). Essa nomenclatura é uma analogia à rede elétrica (*power grid*) que disponibiliza a energia sem se preocupar em explicar onde esta foi gerada e como está sendo destinada. A ideia de uma grade computacional é criar um ambiente computacional distribuído que possua mecanismos que permitam o processamento, armazenamento e uso dos recursos de forma transparente para o usuário (DANTAS, 2005).

De acordo com Foster e Kesselman (2004) grade computacional é

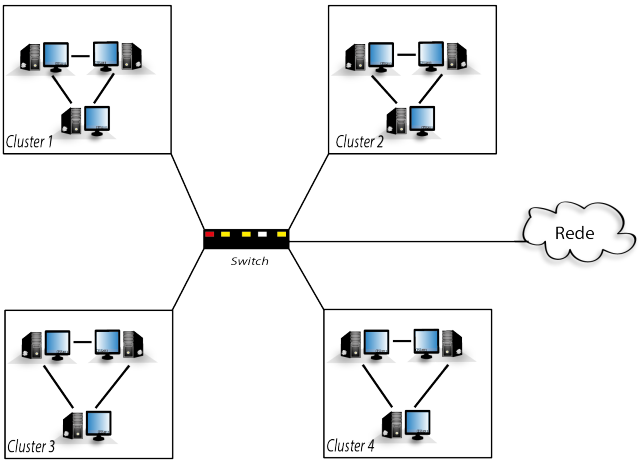

Figura 3 – Exemplo de ambiente *multi-cluster*

uma infraestrutura de *hardware* e *software* que provê acesso confiável, abrangente e barato a capacidades computacionais. Por outro lado, segundo Dantas (2005) grade computacional pode ser entendida como uma plataforma heterogênea de computadores geograficamente distribuídos, onde usuários acessam seus recursos através de uma interface única.

Krauter, Buyya e Maheswaran (2002) propõem uma taxonomia para identificar os tipos de sistemas de grades existentes. Para tanto, os sistemas podem ser divididos nas seguintes categorias:

- **Grade Computacional** (*Computacional Grid*): são sistemas que unem recursos geograficamente distribuídos para obter alta capacidade de processamento. Dependendo de como esta capacidade é utilizada, esse sistema pode ser subdividido em: supercomputação distribuída e grade de alta taxa de transferência.

  Uma grade de supercomputação distribuída executa a aplicação paralelamente entre múltiplas máquinas para reduzir o tempo de conclusão da tarefa. Já uma grade de alta taxa de transferência aumenta a taxa de conclusão de um fluxo de tarefas.

- **Grade de Dado** (*Data Grid*): consiste de uma infraestrutura para acesso, pesquisa e processamento de informações a partir de repo-

sitórios de dados, que estão geograficamente distribuídos.

- **Grade de Serviço** (*Service Grid*): são sistemas que proveem serviços que não são oferecidos por máquinas simples. Normalmente esse tipo de infraestrutura é administrada por grandes instituições e reúne alta performance, recursos computacionais dedicados, como *clusters*, supercomputadores e grandes sistemas de armazenamento de dados (BRASILEIRO et al., 2008). Como exemplo desse tipo de grade pode-se citar: SINAPAD no Brasil (SINAPAD, 2013) e NGS no Reino Unido (NGS, 2013).

Uma configuração de grade é composta por organizações virtuais (OV) (do inglês *Virtual Organizations*), que podem ser indivíduos ou entidades que compartilham seus recursos, porém apresentam determinadas políticas quanto ao acesso e uso e algumas restrições quanto à disponibilidade destes. Diante disso, tem-se proposto arquiteturas que permitam a interoperabilidade entre diferentes organizações virtuais. Camargo et al. (2006) apresentam uma arquitetura genérica implementada pela maioria dos sistemas de grades. Essa arquitetura é ilustrada na Fig. 4 e possui cinco serviços, que serão descritos a seguir.

- **Agente de Acesso**: é a primeira ponte de acesso para os usuários, permitindo a interação entre usuário e grade. Está presente em cada nó que solicite a execução de aplicações.

- **Serviço Local de Oferta de Recursos**: executado em cada máquina que disponibiliza seus recursos para a grade. É responsável por iniciar a execução da aplicação, reportar erros e retornar resultados.

- **Serviço Global de Gerenciamento de Recursos**: é responsável por monitorar o estado dos recursos compartilhados e por responder a solicitações de uso destes recursos, a fim de combinar as requisições com os recursos oferecidos.

- **Serviço de Escalonamento**: é responsável por escalonar as aplicações para recursos disponíveis. Para isso, recebe as solicitações de execução da aplicação, obtém informações quanto à disponibilidade dos recursos junto ao Serviço Global de Gerenciamento de Recursos e determina onde cada aplicação será executada.

- **Serviço de Segurança**: é responsável por proteger os recursos compartilhados para que o nó que compartilha seus recursos não sofra ataques de aplicações maliciosas. Além disso, é responsável por autenticar os usuários de modo que se saiba quem é o responsável pelas aplicações

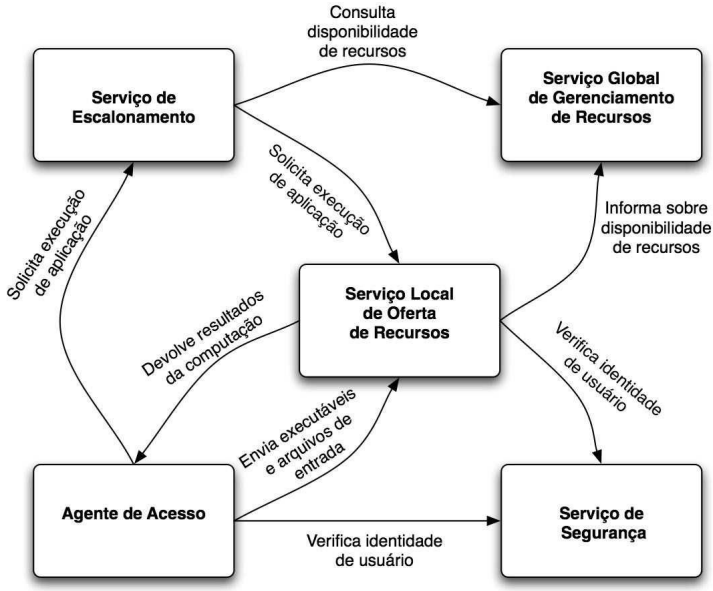

Figura 4 – Arquitetura Genérica de um sistema de grade (KON; GOLDMAN, 2008)

submetidas, bem como a segurança das comunicações da grade, mantendo assim, a confiabilidade e integridade dos dados.

2.5 NUVEM COMPUTACIONAL

Nuvem computacional, do inglês *cloud computing*, tem sido muito utilizada quando deseja-se obter recursos computacionais sob demanda. Apesar da ascensão de seu uso nos dias de hoje, a ideia básica de nuvens computacionais não é nova, tendo sido prevista nos anos 60 pelo cientista da computação John McCarthy, quando mencionou que a computação podia um dia ser organizada como de utilidade pública exatamente como o sistema de telefonia que é de utilidade pública (GARFINKEL, 1999).

O conceito de nuvem computacional pode ter diferentes percepções, já que não é uma tecnologia nova e sim uma junção de tecnologias existentes executadas de forma diferenciada.

Segundo Foster et al. (2008) nuvem computacional é um paradigma

de computação distribuída de larga escala que é impulsionado pela economia, na qual um conjunto de recursos abstratos, virtualizados, dinamicamente escaláveis, com poder computacional gerenciável e armazenável, além de plataformas e serviços, são proporcionados para usuários externos através da Internet. Por outro lado, segundo a NIST (MELL; GRANCE, 2011), responsável por desenvolver padrões e diretrizes, nuvem computacional é um modelo para permitir acesso à rede sob demanda de forma ubíqua e conveniente para o compartilhamento de recursos computacionais configuráveis (por exemplo, rede, servidores, armazenamento, aplicações e serviços) que pode ser rapidamente fornecido e liberado com o mínimo de esforço de gerenciamento ou de interação com o provedor de serviço.

O objetivo da nuvem computacional é fazer melhor uso dos recursos distribuídos, juntando-os, de modo a aumentar a taxa de transferência, além de ser capaz de resolver problemas computacionais de larga escala (RIMAL; CHOI; LUMB, 2009).

Existem muitas definições para o modelo de arquitetura de nuvem computacional, no entanto, de acordo com Zhang, Cheng e Boutaba (2010) a arquitetura está dividida em modelo de níveis, modelo de serviços e tipos de nuvens. A seguir será detalhada cada uma das divisões da arquitetura proposta.

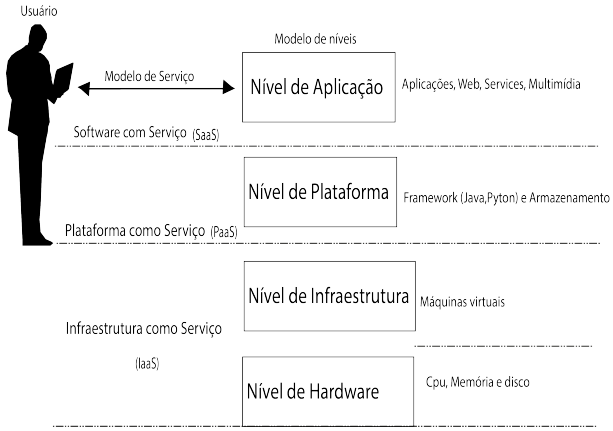

Figura 5 – Modelo de níveis da nuvem computacional

- **Modelo de Níveis**: a arquitetura de nuvem computacional pode, de modo geral, ser subdividida em quatro níveis, conforme apresentados na Fig. 5: nível de hardware, nível de infraestrutura, nível de plata-

forma e nível de aplicação.

  - Nível de Hardware: este nível é responsável por gerenciar recursos físicos como: servidores, roteadores e *switches*.
  - Nível de Infraestrutura: também conhecido como nível de virtualização, é um componente essencial da nuvem, pois cria um *pool* de armazenamento e recursos computacionais através do particionamento lógico de recursos físicos utilizando tecnologias de virtualização.
  - Nível de Plataforma: adiciona uma coleção de ferramentas, *middleware* e serviços especializados para proporcionar uma plataforma de desenvolvimento (FOSTER et al., 2008).
  - Nível de Aplicação: contém a aplicação que será executada na nuvem.

Cada nível da arquitetura é fracamente acoplado com os níveis acima e abaixo, permitindo que cada um se desenvolva separadamente. A arquitetura modular possibilita que a nuvem computacional suporte grandes quantidades de requisições das aplicações, reduzindo a sobrecarga de gerenciamento e manutenção.

- **Modelo de Serviços**: nesse modelo, todos os níveis da arquitetura da nuvem computacional podem ser implementados como um serviço para o nível acima. Consequentemente, todos os níveis podem ser percebidos como clientes do nível abaixo (ZHANG; CHENG; BOUTABA, 2010). Entretanto, em geral, as nuvens computacionais oferecem serviços que podem ser agrupados em três categorias, conforme apresenta a Fig. 5: software como serviço (SaaS - *Software as a Service*), plataforma como serviço (PaaS - *Platform as a Service*) e infraestrutura como serviço (IaaS - *Infrastructure as a Service*).

  - Infraestrutura como Serviço (IaaS): refere-se ao fornecimento de processamento, armazenamento, rede e outros serviços computacionais fundamentais capazes de desenvolver e executar software, incluindo sistemas operacionais e aplicativos. No entanto, o usuário não gerencia ou controla a infraestrutura da nuvem, o que lhe impede de controlar os sistemas operacionais, armazenamento e desenvolvimento de aplicações (MELL; GRANCE, 2011).
  - Plataforma como Serviço (PaaS): oferece ao usuário um ambiente para desenvolvimento de aplicações personalizadas. Geralmente os desenvolvedores precisam respeitar algumas restrições como

o tipo de software que eles podem utilizar. Nessa categoria, o usuário não gerencia ou controla a infraestrutura da nuvem, o que inclui rede, servidores, sistemas operacionais e armazenamento.

  - Software como Serviço (SaaS): refere-se ao fornecimento de aplicações acessadas remotamente pelos usuários através da Internet.

- **Tipos de Nuvens**: de acordo com a NIST (MELL; GRANCE, 2011), existem quatro tipos de nuvens computacionais:

  - Nuvens Públicas: uma nuvem na qual provedores de serviço oferecem, baseado em um modelo de preço, seus recursos computacionais para o público em geral.
  - Nuvens Privadas: são projetadas para uso exclusivo de uma única organização, podendo ser construídas e gerenciadas pela organização ou por provedores externos.
  - Nuvens de Comunidades: a infraestrutura da nuvem é compartilhada por várias organizações e suporta uma comunidade específica que possui as mesmas preocupações, tais como: missão, requisições seguras, políticas.
  - Nuvens Híbridas: a infraestrutura da nuvem é composta por duas ou mais nuvens (pública, privada e de comunidade) unidas por tecnologia padronizada ou proprietária que permite a portabilidade de dados e aplicações.

2.6 COMPUTAÇÃO OPORTUNISTA

O conceito de oportunismo tem sido utilizado em diversas áreas, tais como as áreas de redes e ambientes computacionais distribuídos.

Em redes oportunistas, como em redes móveis ad hoc (MANET), os dispositivos distribuídos pelo ambiente formam a rede (CONTI et al., 2010). Os nós têm escasso ou nenhum conhecimento quanto à topologia da rede. As rotas são construídas de forma dinâmica, ou seja, quando as mensagens são enviadas entre o remetente e o destinatário qualquer nó pode ser utilizado de forma oportunista para que leve a mensagem para mais perto do destinatário. As estruturas sociais humanas são a essência de soluções das redes oportunistas. Os seres humanos carregam dispositivos móveis e a mobilidade humana gera oportunidades de comunicação quando dois dispositivos, ou mais, entram em contato.

Entretanto, quando os dispositivos entram em contato, ainda que de forma oportunista, torna-se possível, não somente o uso oportunista da rede, mas também o compartilhamento de seus recursos para, por exemplo, executar tarefas remotamente. Com isso, se desenvolve um novo conceito de computação, a computação oportunista. Esta, por sua vez, explora a comunicação oportunista entre pares de dispositivos para compartilhar o conteúdo, os recursos e os serviços de cada um. A computação oportunista generaliza o conceito de rede oportunista, considerando o uso oportunista de qualquer recurso disponível na rede.

Para Conti e Kumar (2010) a computação oportunista explora os recursos disponíveis em um ambiente oportunista para fornecer uma plataforma de computação distribuída para a execução de tarefas. Entretanto, faz-se necessário o uso de serviços de *middleware* para mascarar desconexões e atrasos, gerenciar recursos de computação heterogêneos, serviços e dados, bem como para fornecer uma visão uniforme do sistema para as aplicações e para o usuário.

Devido à instabilidade apresentada por ambientes de computação oportunista, sua criação pode oferecer alguns desafios. Um deles é a interrupção da conectividade, causada pela falta de conhecimento prévio sobre a localização, tempo e largura de banda de cada contato. Implementações bem sucedidas de aplicações em redes tolerantes a atrasos, DTN (*Delay-Tolerance Network*) , têm demonstrado a utilidade das redes oportunistas. A tolerância a atraso é outro desafio importante e a chave da computação oportunista. Por fim, a heterogeneidade tem se mostrado um desafio importante para a computação oportunista, tendo em vista a diversidade de dispositivos que entram em contato oportunisticamente, dentre eles, os telefones celulares, os computadores de mão e notebooks, os sensores, as câmeras e os objetos com identificação por radiofrequência, RFID (*Radio-Frequency Identification*) . Estes dispositivos podem ser suportados por diversas frequências de rádio nas quais se comunicam.

Por fim, a computação oportunista faz uso de recursos de dispositivos móveis e da rede sem fio de modo a utilizá-los, oportunisticamente, para troca de informações e execução de tarefas remotamente.

2.7 MIDDLEWARE PARA AMBIENTES DISTRIBUÍDOS

A interconexão de máquinas com diferentes características de hardware ou de software, para a construção de um ambiente distribuído, pode prejudicar a utilização integrada de seus recursos para execução de aplicações. Diante disto, têm se desenvolvido nos últimos anos sistemas conhecidos como

middlewares, que são responsáveis pela ocultação das particularidades apresentadas pelas máquinas as quais opera.

Para Dantas (2005), um *middleware* pode ser entendido como um pacote de software com interface de programação própria responsável por permitir que desenvolvedores de aplicação não se preocupem com as características intrínsecas do sistema operacional.

Existem vários *middlewares* para *clusters* e grades computacionais, dentre eles pode-se citar: HTCondor (HTCONDOR, 2013), XtremWeb (XTREMWEB, 2013) e BOINC (BOINC, 2013). A seguir uma breve descrição de cada um deles.

2.7.1 HTCondor

O HTCondor, conhecido anteriormente como Condor, é um software que cria um ambiente de computação de alta taxa de transferência (*High-Throughput Computing* - HTC). HTC, por sua vez, pode ser entendido como um ambiente que proporciona uma grande quantidade de poder computacional por um longo período de tempo.

Desenvolvido pela Universidade de Wisconsin-Madison, o HTCondor é um sistema de lote especializado em gerenciar tarefas que requeiram uso computacional intensivo. O sistema possui um mecanismo de enfileiramento, políticas de escalonamento, esquema de prioridade e classificação de recursos.

Diferentemente dos demais sistemas de lote, o HTCondor pode utilizar tanto máquinas dedicadas quanto não dedicadas para executar as tarefas. Uma de suas principais características é a capacidade de detectar que a máquina que está executando a tarefa não está mais disponível, fazer *checkpoint* da tarefa e migrar esta tarefa para que uma máquina ociosa continue a execução exatamente de onde parou. Esse processo maximiza o número de máquinas que podem executar uma tarefa.

2.7.2 XtremWeb

XtremWeb é uma plataforma de código aberto, desenvolvida pelo *Laboratoire de Recherche en Informatique da Université Paris-Sud*, na França. O principal objetivo esse projeto é a formação de ambientes distribuídos, tais como grades computacionais (seção 2.4), computação global, computação voluntária e sistemas *Peer-to-Peer*.

Assim como em outros sistemas distribuídos de larga escala, o

XtremWeb utiliza recursos ociosos (CPU, armazenamento e rede) de computadores remotos, conectados à Internet, ou de computadores pertencentes a uma LAN.

A arquitetura do XtremWeb é composta pelo Cliente, Coordenador e *Workers*. Durante a execução, os *workers* contactam o servidor para obter os *jobs*. Em resposta, o servidor envia um conjunto de parâmetros e uma aplicação, caso o *worker* ainda não a tenha armazenado. Quando o *worker* finaliza a execução de seu *job*, ele contacta o coordenador para enviar seus resultados. Nesta arquitetura o cliente e o servidor são o mesmo computador.

O XtremWeb também pode ser utilizado para construir sistemas *Peer-to-Peer* centralizados. Neste caso, os *jobs* submetidos pelo cliente são registrados no servidor e escalonados para os *workers*.

2.7.3 BOINC

BOINC (*Berkeley Open Infrastructure for Network Computing*) é uma plataforma de *software* para grades *Desktop*. Em outras palavras, é um *middleware* que utiliza recursos computacionais ociosos de máquinas que disponibilizam voluntariamente seus recursos para executar aplicações que exigem alto poder computacional.

Desenvolvido pela Universidade da Califórnia em Berkeley, o BOINC foi projetado para suportar aplicações que necessitam de grandes exigências computacionais e de armazenamento. Dentre suas principais características pode-se citar a autonomia dos projetos pois cada um opera com o seu próprio servidor e base de dados; a flexibilidade oferecida para os voluntários, que podem controlar para quais projetos vão disponibilizar seus recursos e como esses serão divididos entre os projetos escolhidos; e o suporte dado à aplicações que produzem ou consomem grandes quantidades de dados ou que utilizam grandes quantidades de memória.

2.8 CONSIDERAÇÕES SOBRE O CAPÍTULO

Nesse capítulo foram abordados conceitos sobre as infraestruturas de sistemas distribuídos existentes. *Cluster* é um conjunto de máquinas dedicadas ou não, geralmente homogêneas, de um único domínio e, normalmente, interligadas por redes de alta performance. Já o *Multi-Cluster* é um conjunto de vários *clusters* interligados por rede WAN. Com isso, possui vários domínios. A grade computacional é composta por máquinas heterogêneas geograficamente distribuídas e, normalmente, não dedicadas. A nuvem com-

putacional pode ser considerada uma junção de tecnologias existentes, porém utilizadas de maneira diferenciada. Permite o acesso sob demanda à recursos e serviços através da Internet. Por fim, a computação oportunista explora os recursos computacionais e de rede ociosos para execução de tarefas.

Cada ambiente descrito acima, possui o escalonamento distribuído diferenciado, de acordo com sua característica.

3 ESCALONAMENTO EM AMBIENTES OPORTUNISTAS

Neste capítulo é apresentado primeiramente o conceito de escalonamento e mostra o processo de execução de uma aplicação em ambientes distribuídos. Em seguida, são apresentados os conceitos relativos à reserva antecipada de recursos, meta-escalonamento, grade oportunista e detalhado o ambiente utilizado para a implementação da proposta deste trabalho de pesquisa. Por fim, são apresentados trabalhos correlatos à proposta desta dissertação, bem como uma análise comparativa entre eles.

3.1 INTRODUÇÃO

Em um ambiente típico de grade uma aplicação pode gerar vários *jobs*, que, por sua vez, podem ser compostos por tarefas. O sistema de grade é responsável por enviar cada tarefa para um recurso para ser executada (XHAFA; ABRAHAM, 2010). Em um cenário de grade simples, o usuário é quem seleciona a máquina mais adequada para executar sua aplicação ou tarefa. Contudo, em geral, sistemas de grade dispõem de escalonadores que procuram de maneira automática e eficiente as máquinas mais adequadas para executar os *jobs*.

Escalonadores são componentes de *software* que destinam as tarefas para recursos da grade sob vários critérios e de acordo com a configuração do ambiente.

O processo de escalonamento de um escalonador consiste em determinar quando e onde os *jobs* serão executados e quantos recursos serão alocados. O processo de alocação, por sua vez, destina recursos para os *jobs* de acordo com seus requisitos e com o estado do sistema. O escalonamento em um ambiente de grade possui quatro etapas, que serão descrita a seguir.

1. **Descoberta de recursos**: o escalonador tem acesso às informações sobre os recursos disponíveis e as tarefas a serem executadas através do GIS (*Grid Information Service*);

2. **Seleção de Recursos**: nem todos os recursos são candidatos para a alocação da tarefa. Por isso, nessa fase os recursos são selecionados com base nos requisitos do *job* e nas características do recurso;

3. **Alocação da Tarefa**: nessa fase a tarefa é alocada no recurso selecionado;

4. **Monitoramento da Execução da Tarefa**: essa fase consiste em informar sobre o andamento da execução, bem como as possíveis falhas do *job*, o que, dependendo da política de escalonamento será reescalonado para outro recurso.

3.2 RESERVA ANTECIPADA DE RECURSOS

Reserva antecipada pode ser definida como a cessão da capacidade de recursos de particulares de forma restrita ou limitada por intervalo de tempo previamente definido (MACLAREN, 2003). Em outras palavras, é o processo onde os usuários podem reservar recursos computacionais para serem utilizados em um determinado período de tempo. Conforme definido em TeraGrid (2013), reserva antecipada dedica um conjunto de recursos para um usuário ou um grupo de usuários para que sejam utilizados durante um intervalo de tempo específico no futuro. Comumente, são reservados recursos como quantidade de CPU e memória, espaço em disco e largura de banda.

Um dos principais objetivos da reserva antecipada, conforme abordado nos trabalhos de Sulistio e Buyya (2004), Castillo, Rouskas e Harfoush (2011) e Tomás et al. (2011), é proporcionar qualidade de serviço através da garantia da disponibilidade dos recursos computacionais no tempo reservado pelo usuário. Além disso, o uso da reserva antecipada permite que os usuários tenham acesso simultâneo e concorrente a recursos adequados para a execução de aplicações paralelas, bem como o aumento da previsibilidade do sistema.

Entretanto, ambientes distribuídos altamente dinâmicos e instáveis como ambientes de grade, ambientes oportunistas e ambientes de *cluster* não dedicados, tornam, muitas vezes, os objetivos da reserva antecipada, descritos acima, inalcançáveis. Diante disto, Foster et al. (1999) menciona que o mecanismo de reserva antecipada proporciona um aumento na expectativa de que os recursos podem ser alocados quando solicitado. Fazendo uma analogia exemplificativa, pode-se comparar a definição de Foster et al. (1999) de reserva antecipada com o serviço oferecido pelo Correio Aéreo Nacional (CAN) às pessoas comuns. Neste serviço qualquer pessoa pode se inscrever para viajar gratuitamente nas aeronaves da Força Aérea Brasileira (FAB), no entanto, as viagens estão condicionadas à disponibilidade de voos, e do tipo de missão da FAB para o destino desejado, bem como o número de vagas disponibilizadas pelo CAN (CAN, 2013). Em outras palavras, é realizada a reserva antecipada, porém não há garantias de que a pessoa realize a viagem.

A reserva antecipada de recursos apresenta como parâmetro principal dois tempos, o tempo inicial (*start time*) e o tempo final (*end time*) da

reserva. O primeiro tempo representa o tempo escolhido pelo usuário para que a execução da aplicação inicie. Por se tratar de reserva antecipada, este tempo é diferente do tempo corrente. Por outro lado, o segundo tempo determina o tempo de término da execução ou de liberação dos recursos. Este tempo também pode ser representado pela duração da execução. Neste caso, o tempo final é calculado através da soma do tempo inicial com a duração.

Apesar da praticidade e usabilidade gerada pela reserva antecipada de recursos aos usuários finais, este mecanismo apresenta diversos desafios. Um dos desafios é lidar com o pouco ou nenhum conhecimento do usuário em relação ao ambiente e às características de sua aplicação. Além disso, é complexo lidar com a dinamicidade dos ambientes distribuídos que apresentam diferentes domínios e políticas de controle. Um terceiro desafio é lidar com a complexidade das solicitações por recursos realizadas pelos usuários, que pode acarretar sucessivas negações por parte do gerenciador, o que pode causar a degradação no desempenho do sistema.

Para se obter uma reserva antecipada de maneira eficiente, o gerenciador deve ter competência para coordenar a descoberta e a seleção de recursos, bem como a alocação e reserva de vários recursos simultaneamente. Estas funções são desenvolvidas pelo meta-escalonador, que será visto na próxima seção.

3.3 META-ESCALONAMENTO

Meta-escalonamento, do inglês *meta-scheduling*, provê uma interface de acesso a recursos virtualizados para os usuários finais, aplicando políticas globais tanto para provedores de recursos quanto para consumidores (XIAOHUI et al., 2006). Seu objetivo principal é controlar e facilitar o acesso e manter o gerenciamento das tarefas entre os diversos gerenciadores de recursos.

Meta-escalonadores (do inglês *meta-scheduler*) e sistemas de escalonamento local implementam as funcionalidades de gerenciamento dos recursos (ADZIGOGOV; SOLDATOS; POLYMENAKOS, 2005). No entanto, sistemas de escalonamento local têm uma visão apenas local dos recursos que controlam, não tendo acesso a toda gama de recursos da grade. Por outro lado, meta-escalonadores possuem uma visão mais ampla dos recursos da grade, o que lhes permite coordenar os recursos em maior escala. A Figura 6 apresenta um exemplo de um ambiente com meta-escalonador.

Os meta-escalonadores são componentes de *middleware* que se compromete a distribuir automaticamente os *jobs* através dos recursos heterogêneos dispersos em uma grade (ADZIGOGOV; SOLDATOS; POLYME-

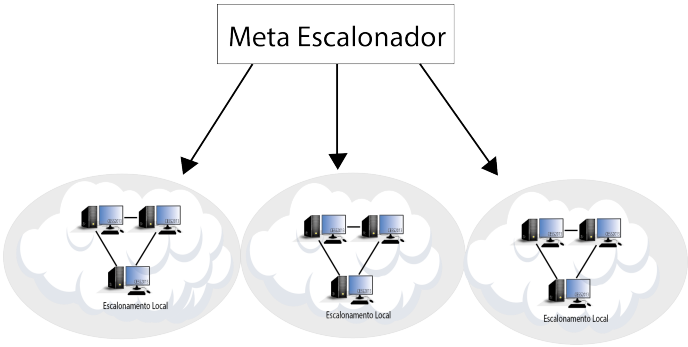

Figura 6 – Exemplo de ambiente com meta-escalonador

NAKOS, 2005). Seu principal componente é um algoritmo que seleciona o sistema que executará o *job*. Após esta seleção, o meta-escalonador torna-se responsável pelos processos de submissão, execução do *job* no sistema destino e monitoramento da execução.

Segundo Qin, Dantas e Bauer (2013) meta-escalonadores são caracterizados por receberem requisições dos usuários e então escalonar uma aplicação para um *cluster* local ou sistemas de escalonamento local, podendo ainda orquestrar configurações *multi-clusters*. A funcionalidade de reserva de recursos em um meta-escalonador requer mecanismos eficientes de co-escalonamento, co-reserva e co-alocação. Estes conceitos são descritos nas subseções seguintes.

3.3.1 Co-Alocação

Co-alocação pode ser definida como a alocação simultânea de múltiplos recursos em diferentes *clusters* (SINAGA, 2004). Segundo Qin e Bauer (2009) este procedimento oferece a possibilidade de um uso mais eficiente dos recursos do computador e redução no tempo de resposta.

Em um ambiente de grade a co-alocação oferece maiores problemas e desafios tais como independência de localização de acesso e gerenciamento de recursos, heterogeneidade de recursos no sentido de capacidade e de políticas e localização de recursos geograficamente distribuídos.

A co-alocação pode ser alcançada fazendo um conjunto de reservas dos recursos necessários com os respectivos provedores de recursos (MACLAREN; KEOWN; PICKLES, 2006).

3.3.2 Co-Escalonamento

Co-escalonamento, do inglês *Co-Scheduling*, pode ser entendido como a habilidade de escalonar *jobs* para serem executados ao mesmo tempo, porém em diferentes nós de processamento. O co-escalonamento pode ser de dois tipos, confirme descritos na pesquisa de Sodan (2005):

- *Gang*: este tipo de escalonamento significa escalonar todas as tarefas de um *job* ao mesmo tempo, ou seja, escalonar simultaneamente em diferentes nós ou processadores alocados para este *job*. Fatias de tempo são globalmente coordenadas e todas as tarefas do *job* são interrompidos caso outro *job* tente executar. Escalonamento *Gang* pode ser suportado por *hardware* podendo também ser executado inteiramente por *software*. Não é muito flexível, sendo as decisões de escalonamento estritamente executadas.

- *Loosely Coordinated*: originário dos ambientes NOW (*Network of Workstation*) e do problema de escalonar juntamente cargas de trabalho serial e paralela. É baseado no escalonamento local e na organização de comunicação dos nós envolvidos. O objetivo é continuar a comunicação de tarefas de um mesmo *job* no mesmo espaço de tempo. É mais flexível pois permite alterações dinâmicas de acordo com as necessidades no momento da alocação.

3.3.3 Co-Reserva

Segundo Manteescu (MATEESCU, 2003) co-reserva pode ser entendida como a reserva simultânea de múltiplos recursos em múltiplos gerenciadores de recursos locais. Na co-reserva duas relações são utilizadas, a relação espacial e temporal (ROBLITZ; REINEFELD, 2005).

- **Relação Espacial**: refere-se a localização dos recursos reservados. Em outras palavras, reserva-se o recurso tendo como base o horário de uso de um determinado recurso;

- **Relação Temporal**: ocorre quando o horário de uso de um recurso coincide ou tem relação com o horário de uso de outro recurso. Não significa que as reservas serão executadas simultaneamente, pois o horário de uso do recurso pode iniciar quando terminar o uso de outro recurso. Em outras palavras, reserva-se o recurso tendo como base o horário de uma outra reserva.

Usualmente são utilizadas tabelas de alocação (*timeslot*) que representam a porcentagem de recursos disponíveis e alocados durante um determinado período de tempo (FOSTER et al., 1999). A Figura 7 apresenta um modelo de tabela de alocação com alocações imediatas e reservas de recursos para execuções futuras, representados pelos ”As” e pelos ”Rs”, respectivamente.

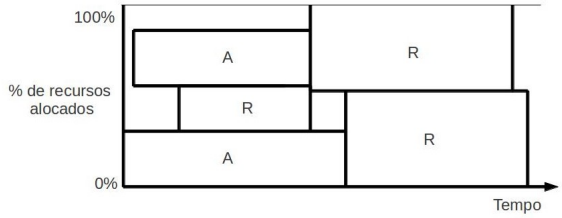

Figura 7 – Exemplo de tabela *timeslot*

3.3.4 Balanceamento de Carga

Escalonamento dinâmico baseia-se na redistribuição de processos entre os processadores durante o tempo de execução. Essa redistribuição é realizada com a transferência das tarefas dos processadores muito carregados para os processadores com pouca carga de trabalho, este procedimento é nomeado como balanceamento de carga (SHIRAZI; KAVI; HURSON, 1995). Um algoritmo de balanceamento de carga, do inglês *load balancing*, é definido sob três políticas:

1. **Política de Informação**: especifica a quantidade de informações de carga disponível para alocação do *job*;

2. **Política de Transferência**: determina as condições sob as quais um *job* deve ser transferido, ou seja, a carga atual da máquina e o tamanho do *job* em questão;

3. **Política de Colocação**: identifica o elemento de processamento no qual um *job* que deve ser transferido.

As técnicas de balanceamento e compartilhamento de carga possuem pouco ou nenhum conhecimento do tempo de compilação com base nos

parâmetros de execução de uma aplicação. Diante disto, estas técnicas dependem da redistribuição do tempo de execução dos processos entre os nós para atingir os objetivos de desempenho definidos.

O compartilhamento de carga, do inglês *Load-sharing*, tem como objetivo maximizar a taxa com que os sistemas distribuídos realizam seus trabalhos, quando estes trabalhos estão disponíveis. Além disso, se esforça para evitar estados não compartilhados, ou seja, estados em que há computadores ociosos enquanto tarefas disputam serviços com outro computador (SHIVARATRI; KRUEGER; SINGHAL, 1992). Por outro lado, balanceamento de carga, também tem o objetivo de evitar estados não compartilhados, porém com um diferencial que tenta equalizar as cargas em todos os computadores. Além disso, o balanceamento de carga pode potencialmente reduzir a média e o desvio padrão do tempo de resposta das tarefas.

3.4 GRADE OPORTUNISTA

Partindo do mesmo conceito da computação oportunista, apresentado no capítulo 2, de utilizar recursos ociosos, porém em ambiente diferente, tem-se a grade oportunista.

Grade oportunista, segundo Goldchleger (2004), pode ser entendida como um subtipo de grades computacionais que utilizam recursos computacionais ociosos de máquinas não dedicadas e geograficamente distribuídas, para formarem dinamicamente uma grade.

Com a rápida difusão de seu uso, devido a sua economicidade e praticidade, pesquisadores têm estudado a estrutura das grades oportunistas de modo a sugerir taxonomias e classificações aceitáveis para o desenvolvimento destas grades. Esta dissertação assume que as grades oportunistas estão divididas e classificadas em duas categorias, conforme apresentadas nos trabalhos de Choi et al. (2007) e de Ponciano e Brasileiro (2012): Grades *Desktop* e *Peer-to-Peer* Oportunista.

As subseções a seguir apresentam mais detalhadamente cadas uma delas.

3.4.1 Grades *Desktop*

As grades *Desktop* são caracterizadas de acordo com a organização, a plataforma, a escala e o provedor de recursos, conforme pode ser visto na Fig. 8.

- **Organização**: quanto à organização de seus componentes, as grades

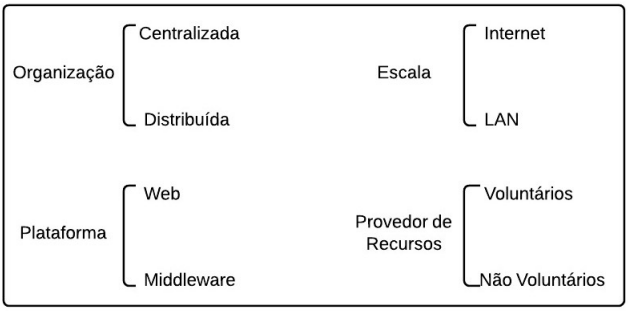

Figura 8 – Taxonomia das grades *Desktop*

Desktop podem ser caracterizadas como centralizada ou distribuída.

A organização centralizada consiste de nós clientes que são responsáveis pela submissão de tarefas a serem executadas, de nós provedores de recursos que disponibilizam seus recursos computacionais ociosos para a execução de aplicações e de um nó servidor responsável pelo gerenciamento de toda a grade. Em outras palavras, o servidor controla a submissão das aplicações, gerencia o escalonamento destas aplicações para os provedores de recursos além disso, verifica o resultado da execução e retorna-o para o cliente. Já a organização distribuída consiste de nós clientes e nós provedores de recursos, onde os nós provedores de recursos obtêm informações parciais de outros provedores. Diante disso, os nós provedores de recursos são responsáveis também pela construção dinâmica da rede, bem como o escalonamento distribuído das tarefas.

- **Plataforma**: quanto à plataforma, as grades *Desktop* podem ser subdivididas em: plataforma baseada na *Web* e baseada em *middleware*.

  Na plataforma baseada na *Web* os clientes escrevem suas aplicações e as postam na *Web*. Os provedores voluntários, por sua vez, realizam o *download* das aplicações e as executam. Em contrapartida, nas plataformas baseadas em *middleware*, os voluntários precisam instalar e executar um *middleware* específico para que a execução da aplicação ocorra.

- **Escala**: de acordo com a escala, estas grades podem ser categorizadas como baseada na Internet e baseada na LAN .

Na categoria baseada na Internet, a grade *Desktop* é formada por provedores de recursos anônimos, podendo ter a presença de *Firewall*, NAT, endereçamento dinâmico, baixa largura de banda e conexão não confiável. Por outro lado, na categoria baseada em LAN, os provedores de recursos estão dentro de um mesmo domínio, ou seja, de uma mesma corporação ou instituição.

- **Provedor de Recursos**: quanto ao provimento dos recursos, as grades *Desktop* podem ser classificadas como voluntárias e não voluntárias.

  No sistema voluntário, os usuários disponibilizam seus recursos computacionais ociosos para contribuir com a resolução de problemas de pesquisas avançadas, frequentemente de amplo interesse social (FOSTER; KESSELMAN, 2011). Este tipo de sistema tem que ser capaz de lidar com problemas como a não confiabilidade dos computadores e com a baixa conectividade. Já no sistema não voluntário, as implementações são realizadas em um ambiente mais controlado no qual participantes formam parte de uma única organização real ou de uma organização virtual.

A Tabela 1 apresenta algumas características básicas que permitem a diferenciação entre grades computacionais e grades *Desktop*.

Tabela 1 – Comparação entre grades computacionais e grades *Desktop* (CHOI et al., 2007)

Item	**Grades *Desktop***		**Grades Computacionais**
	Baseada na Internet	Baseada na LAN	
Recurso	*Desktop* provedores de recursos anônimos	*Desktop* provedores de recursos de OV	Supercomputadores, *cluster* base de dados, armazenamento
Conexão	Largura de banda pobre Não dedicada Conectividade imediata Considera *Firewall*, NAT e endereçamento dinâmico	Largura de banda intermediária Não dedicada Conexão mais constante	Dedicada de alta velocidade
Heterogeneidade	Altamente heterogêneo	Heterogeneidade intermediária	Baixa heterogeneidade
Dedicação	Não dedicado Altamente volátil	Não dedicado Volatilidade baixa	Dedicado
Confiabilidade	Pode apresentar provedores maliciosos	Baixa confiabilidade	Alta confiabilidade
Falta	Instável	Instável	Mais estável
Gerenciabilidade	Administração individual totalmente distribuído e de difícil gerenciamento	Administração individual mais controlável	Administrado por profissional
Aplicação	Independente Computação intensiva Alta vazão	Independente Computação intensiva Possibilidade de dados intensivos Alta vazão	Independente ou dependente Computação ou dados intensivos Alta performance

3.4.2 Grade *Peer-to-Peer*

Nas grades *Peer-to-Peer* os recursos computacionais são disponibilizados pelos usuários, porém com o diferencial de que estes esperam um

retorno de favores, como por exemplo, maior prioridade quando requisita recursos (PONCIANO; BRASILEIRO, 2012).

Outro ponto importante que permite a diferenciação destas grades em relação as grades *Desktop* é que a conexão das máquinas por meio de uma rede local abrange recursos que são gerenciados pelos próprios *Peers* da grade e a demanda por recursos é tipicamente por rajadas. Demanda por rajadas pode ser entendido como breves períodos de alta demanda por recursos, seguido por períodos de inatividade possivelmente longos. Pode-se citar como exemplo de grades *Peer-to-Peer* oportunista o projeto OurGrid (OURGRID, 2013).

A seguir será detalhada a arquitetura do ambiente de grade oportunista utilizado para o desenvolvimento do protótipo.

3.5 INTEGRADE

InteGrade é um projeto de *middleware* para computação oportunista desenvolvido em parceria entre universidades brasileiras (GOLDCHLEGER et al., 2004). Seu principal objetivo é utilizar recursos computacionais ociosos (CPU e memória) de computadores pessoais, estações de trabalho, para executar aplicações que requerem alto poder computacional.

A unidade estrutural básica do InteGrade é de um *cluster*, ou seja, uma coleção de máquinas conectadas, tipicamente, por uma rede local (SILVA et al., 2010). Os *clusters* InteGrade podem ser interligados eficientemente, de maneira a permitir a criação de um sistema de grade altamente escalável com até milhares de máquinas (GOLDCHLEGER, 2004). Cada *cluster* possui os seguintes componentes:

- **Gerenciador do *Cluster***: responsável pelo gerenciamento do *cluster*. Dentre suas funções, pode-se citar o escalonamento das aplicações, o registro dos provedores de recursos no *cluster*, bem como a comunicação com gerenciadores pertencentes a outros *clusters*.
- **Provedores de Recursos**: responsáveis por disponibilizar seus recursos ociosos para o *cluster*, a fim de executar aplicações a eles submetidas.
- **Nó Usuário**: contém a interface do usuário. Responsável por submeter aplicações a serem executadas pelos Provedores de Recursos.

Atualmente, o InteGrade executa três tipos de aplicações:

- **Aplicações Sequenciais**: são aplicações simples, com apenas um binário executável, que requerem apenas um nó para execução;

- **Aplicações Paramétricas ou *Bag-og-Tasks***: várias cópias de uma mesma aplicação são distribuídas para diferentes nós. Cada uma delas processa diferentes conjuntos de parâmetros ou arquivos de entrada de maneira independente, ou seja, não há comunicação entre os nós;
- **Aplicações Paralelas**: aplicações paralelas do tipo único programa com múltiplos dados (SPMD - *Simple Program, Multiple Data*), com comunicação entre os nós. Estas aplicações são baseadas nos modelos BSP (*Bulk Synchronous Parallelism*) e MPI (*Message Passing Interface*) .

A arquitetura do InteGrade, apresentada na Fig. 9, é composta por módulos responsáveis pelo funcionamento do *cluster*. A comunicação entre esses módulos é realizada por meio de chamadas remotas. O InteGrade utiliza o CORBA (CORBA, 2013) para a realização da comunicação remota. CORBA é um projeto de *middleware* que permite que os módulos se comuniquem independente de linguagens de programação, plataforma de *software* e *hardware* e rede de comunicação (COULOURIS; DOLLIMORE; KINDBERG, 2007).

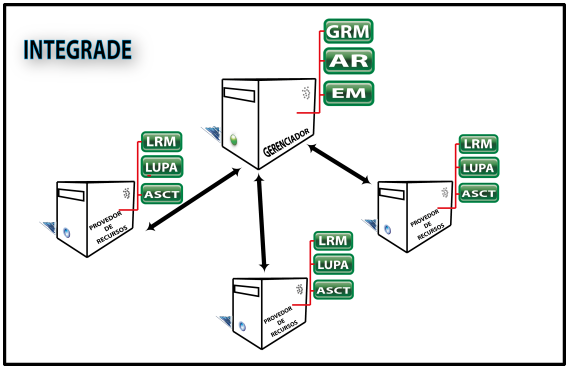

Figura 9 – Arquitetura do InteGrade

O **LRM** (*Local Resource Manager*) está presente em todos os nós Provedores de Recursos. É responsável por coletar e distribuir informações sobre a disponibilidade dos recursos do nó, além de instanciar e executar aplicações a ele submetidas.

O **GRM** (*Global Resource Manager*) atua como gerenciador do *cluster*. Recebe as informações coletadas pelos LRMs sobre a quantidade e disponibilidade dos recursos pertencentes ao *cluster*, e as utiliza para o escalonamento das aplicações. Também é responsável pela comunicação inter-*cluster*.

O **ASCT** (*Application Submission and Control Tool*) é uma interface gráfica amigável que permite que o usuário submeta aplicações para serem executadas no *cluster*. Através dela, o usuário pode estabelecer requisitos como, plataforma de *software*, quantidade máxima de CPU em uso e quantidade mínima de memória disponível, quantidade de nós necessários para a execução da aplicação e forneça os arquivos de entrada, se houver. Além disso, o usuário pode monitorar a execução da aplicação, bem como recuperar arquivos de saída e de eventuais erros da aplicação.

O **AR** (*Application Repository*) armazena as aplicações que serão executadas no *cluster*. Através do ASCT o usuário registra a aplicação no repositório para posteriormente ser executada.

O **EM** (*Execution Manager*) mantém informações sobre a execução de cada aplicação no *cluster*, como o estado da execução, em que nó a aplicação está executando e os nomes dos arquivos de entrada e saída. Também é responsável por reiniciar o processo de execução de aplicações de máquinas que se tornaram indisponíveis.

O **LUPA** (*Local Usage Pattern Analyzer*) estabelece padrões de uso a partir da análise do uso dos recursos das máquinas (CONDE, 2008). Estes padrões são utilizados para prever a disponibilidade dos recursos pertencentes ao *cluster*, e assim melhorar o escalonamento das aplicações.

Fazendo uma análise comparativa entre as características do InteGrade e a taxonomia das grades oportunistas apresentada anteriormente, pode-se classificá-lo como uma grade oportunista do tipo grade *Desktop*. O InteGrade possui organização centralizada, plataforma de *middleware* (com implementação da plataforma *Web* em andamento), escala de Internet para InteGrade inter-*cluster* e de LAN para intra-*cluster* e provedores de recursos voluntários.

A próxima subseção apresenta maiores detalhes quanto ao funcionamento do LUPA pois, tal módulo possui um papel importante na proposta desta dissertação.

3.5.1 Local Usage Pattern Analyzer - LUPA

O LUPA é um módulo do InteGrade responsável por monitorar o uso do computador do usuário, de modo a estabelecer padrões de uso e gerar predições de uso e de disponibilidade dos recursos (CONDE, 2008). Atualmente, o LUPA monitora o uso de CPU, medido em porcentagem utilizada (calculada de acordo com a média de utilização desde a última coleta dos dados) e a disponibilidade de memória RAM, medida em kilobytes.

A arquitetura do LUPA é divida em três módulos: módulo de coleta,

módulo de cálculo dos padrões e módulo de predição.

O módulo de coleta é responsável por coletar informações sobre o uso dos recursos a cada 5 minutos e armazenar em arquivos de *log*. O LUPA obtém esses dados através de sua interação com o sistema operacional e, dessa forma, se torna dependente do sistema operacional do nó ao qual pertence. Os arquivos de *log* são arquivos de texto com colunas separadas por tabulação, onde a primeira é um número inteiro que corresponde ao instante em que os dados foram coletados e as demais à utilização dos recursos. Quando, por algum motivo, não for possível coletar os dados de utilização dos recursos, são utilizadas estimativas para o preenchimento da lacuna. É considerado que a estimativa da utilização em um instante que não houve coleta é igual à utilização da última coleta realizada.

O módulo de cálculo dos padrões identifica os padrões de uso dos recursos computacionais compartilhados na grade. Para isso, os dados coletados durante um período de 48 horas são utilizados como entrada para o algoritmo de análise de agrupamento, que é executado uma vez ao dia. Esse algoritmo tem a função de separar em grupos os recursos com comportamentos semelhantes. Cada grupo possui uma lista dos recursos pertencentes a ele e um elemento representativo que é a média dos comportamentos de seus recursos. É com esse elemento que a utilização das últimas horas será comparado.

O módulo de predição é responsável por prever a utilização dos recursos. Para isso, são comparados os dados das últimas 24 horas de monitoramento com os elementos representativos de cada grupo. Ao encontrar o elemento representativo mais próximo, assume-se que o uso futuro serão os valores contidos no grupo do elemento representativo selecionado. Caso o LUPA esteja executando há pouco tempo e não tenha todos os dados das últimas 24 horas monitoradas, o algoritmo faz a média do uso das últimas horas e assume que os valores irão se manter.

3.5.2 Implementação do InteGrade

Conforme mencionado anteriormente, o InteGrade utiliza a especificação CORBA para a comunicação remota entre os módulos a ele pertencentes.

Os módulos do servidor (GRM, EM, AR), bem como o ASCT foram implementados utilizando a linguagem Java e o ORB JacORB. Já os módulos dos provedores de recursos (LRM e LUPA) foram implementados utilizando a linguagem C++. A escolha da linguagem C++ para a implementação desses módulos se deu pois pode-se aliar essa linguagem à orientação a objetos e ao

baixo consumo de recursos. Para a comunicação remota do LRM foi utilizado o ORB OiL (ORB in Lua) (OIL, 2013). O OiL é um ORB compacto desenvolvido na linguagem Lua (LUA, 2013), uma linguagem de *script* que combina programação procedural com poderosas construções para descrição de dados baseadas em tabelas associativas e semântica extensível. Por outro lado, o LUPA não realiza comunicação remota, ou seja, suas informações são acessadas diretamente pelo LRM.

3.6 TRABALHOS CORRELATOS

Esta seção tem como objetivo apresentar algumas abordagens e pesquisas relacionadas à ambientes oportunistas e à reserva antecipada em ambientes distribuídos. Por fim é apresentada uma análise comparativa entre os trabalhos.

3.6.1 GarQ

Para Sulistio et al. (2009) a escolha de uma estrutura de dados pode minimizar significativamente o tempo gasto para procurar nós disponíveis, adicionar novas requisições e excluir reservas existentes.

Diante disso, este trabalho apresenta três contribuições. Primeiramente foram modificadas as estruturas da dados Lista Encadeada e Árvore de Segmentos para suportar operações como adicionar, excluir, pesquisar e buscar por intervalos capazes de lidar com reservas antecipadas em grades computacionais. Para isso foi desenvolvido um algoritmo para a estrutura de dados Árvore Segmentada que procura por intervalos próximos do solicitado para reserva caso tenha sido inicialmente rejeitado.

Na segunda contribuição foi introduzida e adaptada a estrutura de dados *Calendar Queue* para o gerenciamento das reserva. Esta estrutura de dados é uma fila de prioridade para futuro conjunto de eventos problemas na simulação de eventos discretos. O futuro conjunto de eventos partilha de características similares à reserva antecipada em grades, onde são registrados eventos futuros, e horários em ordem cronológica.

Por fim, os autores propõem uma nova estrutura de dados, chamada de *Grid advanced reservation Queue* (GarQ), que é a combinação das estruturas de dados *Calendar Queue* e Árvore de Segmentos, para que as reservas tenham o gerenciamento eficiente das operações de adicionar, excluir, pesquisar e buscar por intervalos livres igualmente implementado na Árvore de Segmentos. A estrutura de dados proposta tem baldes, do inglês *buckets*, (δ)

com valor fixo, que representa um *slot* de pequena duração, assim como na estrutura de dados *Calendar Queue*. Cada balde contém o número de nós já reservados (*rv*) e uma lista encadeada (ordenada ou não) contendo as reservas que iniciam no tempo deste balde. A Tabela 2 apresenta as reservas antecipadas já realizadas e o pedido de reserva do *User5*. Já a Figura 10 apresenta um exemplo de como as reservas são armazenadas no GarQ com lista encadeada ordenada, com o intervalo de tempo $\delta = 1$. Nesse exemplo, o máximo de nós disponíveis é três.

Tabela 2 – Exemplos de alocação das reservas antecipadas

User1	reserva 2 nós no período entre 10 e 11
User2	reserva 1 nó no período entre 10 e 13
User3	reserva 1 nó no período entre 11 e 14
User4	reserva 1 nó no período entre 14 e 15
User5	solicita reserva de 2 nós no período entre 11 e 13

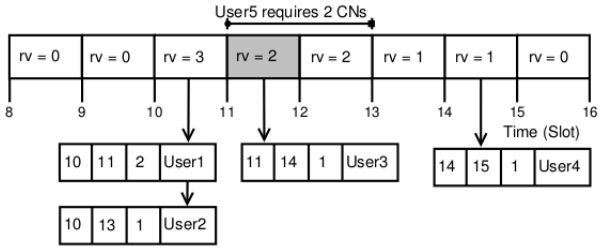

Figura 10 – Armazenamento de reservas no GarQ com fila ordenada e $\delta = 1$ (SULISTIO et al., 2009)

Como resultado experimental, o GarQ obteve melhor performance em cargas de trabalho se comparado com a Lista Encadeada, Árvore Segmentada e *Calendar Queue*. Entretanto, para pequenos *jobs*, a Árvore Segmentada obteve melhor performance no tempo de execução. Além disso, o GarQ obteve melhor taxa em relação ao baixo consumo de memória e em relação a alta performance, se comparada com as outras estruturas de dados.

3.6.2 Proposta de Ferreira, Dantas e Bauer

Neste trabalho é proposto um método de reserva antecipada dinâmica que possibilita que usuários reservem recursos de diferentes *clusters* (FERREIRA; DANTAS; BAUER, 2010). Este método permite que tarefas compostas por um número de processos, que não poderiam ser executados em um único *cluster*, tenham suas reservas distribuídas entre diferentes *clusters*. Isso é possível uma vez que os autores assumem que uma tarefa é composta por processos e cada processo contém diferentes sub-tarefas. Além disso, a proposta faz uso de lógica *fuzzy* para controlar a distribuição das reservas e de ontologias para combinar as requisições descritas pelos usuários com as configurações do sistema.

A Figura 11 apresenta os principais componentes da proposta. Do lado do cliente há uma interface interativa (*Interative Interface*) que recebe do usuário restrições quanto ao tipo e quantidade de recursos necessários para executar a aplicação. Estas restrições são enviadas para o servidor, que possui informações estáticas dos recursos de cada *cluster* do ambiente. O *Selector* compara a requisição feita pelo usuário com as informações estáticas armazenadas, procura todos os possíveis cenários para responder a requisição do usuário e as envia para o *Advance Reservation Manager*.

O *Advance Reservation Manager* acessa as informações das reservas já realizadas e as compara com as recebidas pelo *Selector*, procurando eliminar resultados conflituosos. Então, é determinado o ambiente que será reservado para a execução da aplicação. No entanto, dependendo da quantidade de processadores solicitados pelo usuário, pode não haver um *cluster* que atenda à requisição do usuário. Nesse caso, a reserva é distribuída entre os *clusters* do ambiente e esta distribuição é controlada por um algoritmo baseado na lógica *fuzzy*.

Para que não haja conflitos, o *Advance Reservation Manager* requisita dinamicamente a confirmação da disponibilidade do recurso com o gerenciador de recursos de cada *cluster* local.

Para validar a proposta, o protótipo foi testado através de simulações realizadas com o simulador GridSim onde pôde-se atingir os objetivos de selecionar e reservar *clusters* para atender grandes tarefas de maneira eficaz.

3.6.3 SA-layer

Tomás et al. (2011) propõem um *framework* para melhorar o meta-escalonamento antecipado e um sistema autônomo capaz de prever o tempo de execução de uma tarefa para melhorar a qualidade de serviço oferecida aos

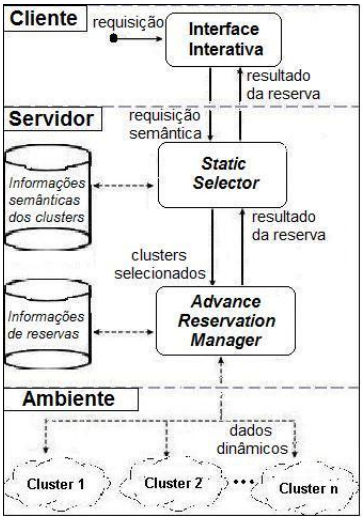

Figura 11 – Principais componentes da proposta de Ferreira, Dantas e Bauer (2010)

usuários da grade.

No meta-escalonamento antecipado proposto são selecionados os recursos para execução das tarefas e o período de tempo que a execução será realizada, porém não há reserva física, uma vez que, segundo os autores, não é possível a reserva antecipada de recursos em ambientes reais de grade computacional. O processo de escalonamento antecipado segue os seguintes passos:

1. O usuário envia um pedido de execução para o meta-escalonador do domínio local. Cada pedido contém informações como arquivos de entrada, aplicação, tempo de início e tempo que a execução deverá terminar. Os dois últimos parâmetros consistem nos parâmetros de QoS;

2. O meta-escalonador se comunica com o *Gap Management* que verifica a disponibilidade de recursos que atendam aos parâmetros de QoS especificados pelo usuário;

3. Se não houver recursos que atendam aos requisitos de QoS exigidos pelo usuário, o meta-escalonador verifica com meta-escalonadores de outros domínios a disponibilidade destes recursos;

4. Se ainda assim não for possível atender aos requisitos de QoS especificados pelo usuário, se inicia um processo de negociação entre o usuário

e o meta-escalonador para que se redefina novos requisitos de QoS.

Para a implementação deste *framework* foi desenvolvida uma extensão do meta-escalonador GridWay, chamado de *Scheduler in Advance Layer* (SA-layer). O SA-layer é um componente modular que utiliza funções provenientes do GridWay, tais como descoberta e monitoramento de recursos, submissão de tarefas e monitoramento de execução, além disso, permite o reconhecimento da rede em meta-escalonamento antecipado. Por ser modular, o SA-layer é composto pelos seguintes componentes, conforme apresentado na Fig. 12.

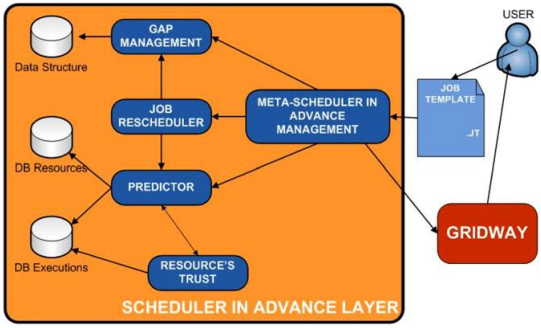

Figura 12 – Principais componentes da SA-layer (TOMÁS et al., 2011)

- *JOB_Information*: este novo parâmetro adicionado ao GridWay possibilita que o usuário indique algumas informações sobre a tarefa, como por exemplo, o tamanho do arquivo a ser transferido, início e fim da execução.

- *DB Executions*: armazena informações sobre aplicações previamente executadas.

- *DB Resources*: armazena informações sobre o estado dos recursos e da rede ao longo do tempo.

- *Data Structure*: estrutura de dados responsável pelo registro de *slots* de tempo livres. A estrutura utilizada neste trabalho é a *Red-Black Trees*, cujo objetivo é desenvolver técnicas eficientes para identificar períodos ociosos de tempo para cada tarefa submetida sem a necessidade de analisar todos os períodos ociosos.

- *Gap Management*: responsável pelo gerenciamento da estrutura de dados, de forma a manter atualizadas as informações armazenadas.
- *Resource Trust*: responsável por calcular a confiabilidade das previsões a fim de ajustá-las conforme a precisão da última estimativa de execução.
- *Predictor*: responsável por estimar o tempo de execução da tarefa, levando em conta as características desta tarefa, o poder de processamento dos recursos e o estado futuro da rede. Para isso, os autores implementaram três algoritmos: o primeiro, chamado de Tempo Total de Completude (TCT - *Total Completion Time*), calcula o tempo total de completude das tarefas em cada recurso computacional baseado em arquivos de *log* de execuções passadas. O segundo algoritmo, chamado de Tempos de Transferência e Execução Separadamente (ETTS - *Execution and Transfer Times Separately*), estima o tempo necessário para execução desta tarefa e o tempo de transferência de arquivos separadamente. Por fim, o terceiro algoritmo, chamado de RT, é baseado no algoritmo ETTS, porém utilizando o *Resource Trust* para complementar a predição.
- *Job Rescheduler*: responsável pelo monitoramento dos recursos ativos e por verificar os *slots* todo o tempo. Quando este módulo detecta que o recurso não estará mais disponível, ele realoca a tarefa para outro recurso.

Como validação da pesquisa, foram realizados quatro testes em dois ambientes de grade reais, onde o objetivo foi realizar comparações entre os três algoritmos (TCT, ETTC e RT) propostos pelos autores e o proposto por Castillo, Rouskas e Harfoush (2008). Os testes consistem no escalonamento das tarefas de forma a atender e não atender a QoS solicitada, bem como na sobreposição e perda de tempo no cálculo da estimativa de tempo de completude da tarefa. Os experimentos mostraram que o algoritmo RT apresentou melhores resultados do ponto de vista do sistema.

3.6.4 Proposta de Martins et. al.

Nesta pesquisa, Martins et al. (2012) propõem um mecanismo de gerenciamento para ambientes de grade computacional oportunista baseado no escalonamento dinâmico e re-escalonamento, cujo objetivo é gerenciar a execução das aplicações com restrições de tempo de execução estabelecidas pelo usuário no momento de sua submissão.

De acordo com a abordagem deste trabalho, o usuário pode especificar em qual das classes, *nice* ou *soft-deadline*, a aplicação submetida pertencerá. Na classe *soft-deadline* o usuário deve fornecer o prazo para a execução da aplicação. O objetivo desta classe é completar a execução da aplicação dentro do prazo determinado. Por outro lado, o objetivo da classe *nice* é apenas completar com sucesso a execução da aplicação.

O mecanismo de gerenciamento proposto neste trabalho é composto pelos seguintes componentes:

- Um mecanismo de predição do tempo de execução das aplicações de acordo com a capacidade de processamento dos recursos da grade;
- Um escalonamento *online* que mapeia as tarefas da aplicação para nós que possam cumprir o prazo de execução especificado pelo usuário;
- Um mecanismo que monitora o progresso da execução das tarefas em cada nó, verificando se é necessário ou não movê-las para que outro nó satisfaça o prazo de execução.
- Um mecanismo de tolerância a falhas baseado em *checkpoint* cujo objetivo é garantir o sucesso da execução das aplicações diante de um evento de falha.

Para o funcionamento do mecanismo, um componente chamado de *Local Resource Manager* (LRM), que é executado em cada nó da grade, recebe regularmente relatórios sobre o progresso da execução das tarefas. Em cada notificação recebida, o LRM estima se a execução da aplicação, no caso de aplicações pertencentes à classe *soft-deadline*, será completada dentro do limite de tempo. Se não, o LRM notifica a aplicação, através de um método de *callback*, forçando sua suspensão e salvando seu estado em um armazenamento estável (*checkpoint*). A requisição de execução das tarefas é encaminhada para o escalonador, que irá re-escalonar as tarefas para um nó da grade que satisfaça as exigências de prazo.

Caso não haja nós disponíveis que atendam as exigências de prazo estabelecidas pelo usuário, o mecanismo procura por nós ocupados que satisfaçam as seguintes condições:

- Se a tarefa pertence a classe *nice*, o mecanismo suspende a execução da tarefa e salva seu estado, permitindo que a nova tarefa continue sua execução;
- Por outro lado, se a tarefa for da classe *soft-deadline*, o mecanismo verifica se o aumento do prazo da atual execução, com a inserção da execução da nova tarefa, será suficiente para cumprir o prazo previsto.

Caso não, o mecanismo reserva o recurso para que a nova tarefa execute quando a atual termine.

Finalmente, caso não existam recursos que atendam às exigências, a submissão da aplicação é rejeitada.

A validação da proposta foi realizada em um ambiente simulado e mostrou resultados experimentais significativamente melhores quando comparado com um escalonamento tradicional utilizado em computação em grade.

3.6.5 OurGrid

O OurGrid é um *middleware* de grade de código aberto, baseado na arquitetura *Peer-to-Peer*, desenvolvido pela Universidade de Campina Grande. Pode ser conceituado como uma cooperativa de grade na qual laboratórios doam seus recursos computacionais ociosos para em troca receberem acesso a recursos ociosos de outros laboratórios quando necessitarem (BRASILEIRO et al., 2008). O objetivo do OurGrid é aumentar a capacidade computacional de laboratórios de pesquisa em todo o mundo sem ser invasivo, ou seja, o usuário local tem prioridade no acesso a recursos locais. Para isso, a submissão de uma tarefa local "mata" qualquer tarefa remota que estiver executando localmente, evitando assim, a queda da performance local.

Para que o OurGrid seja bem sucedido ele deve ser rápido, simples, escalável e seguro (CIRNE et al., 2006). Para ser considerado rápido, o tempo de resposta da tarefa deve ser melhor do que se for utilizado somente os recursos locais, caso contrário, o OurGrid não será útil para o usuário e esse abandonará o sistema. Quanto à simplicidade, laboratórios não querem dispender muito esforço com os computadores que irão resolver seus problemas. O OurGrid deve ser altamente escalável para poder administrar a grande quantidade de recursos computacionais ociosos de laboratórios ao redor do mundo. Por fim, o OurGrid precisa ser seguro pois sua concessão automática de acesso permitirá que códigos desconhecidos acessem as máquinas da grade. Por isso, cada máquina deve se manter segura.

Sistemas *Peer-to-Peer* podem ter sua performance comprometida por *Peers* que consomem recursos e nunca contribuem para a comunidade. Para resolver esse problema, foi desenvolvido para o OurGrid o mecanismo *Network of Favors*. Com esse mecanismo, o *Peer* que doa uma maior quantidade de recursos tem melhor reputação e com isso é priorizado quando requisita recursos. Este mecanismo tem como objetivo incentivar o máximo de contribuição possível para o sistema.

O OurGrid é composto por quatro componentes responsáveis pelo seu

funcionamento, conforme mostrado na Fig. 13.

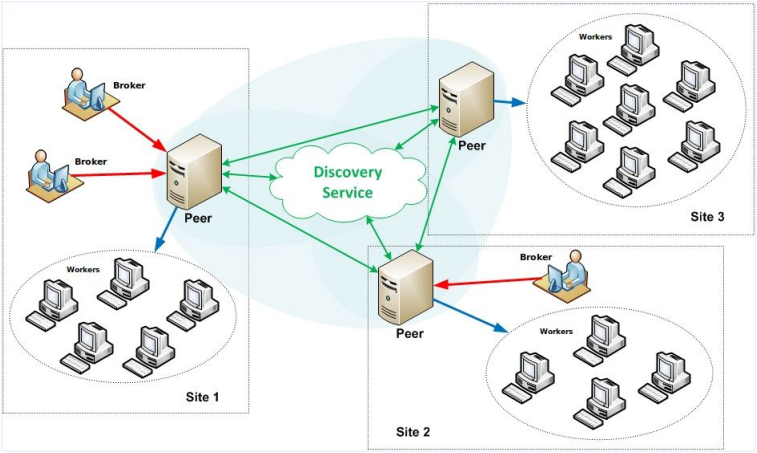

Figura 13 – Arquitetura do OurGrid (OURGRID, 2013)

- *Worker*: é executado nas máquinas da grade e é responsável por implementar um ambiente seguro para a execução das aplicações da grade. Além disso, executa um algoritmo de detecção de ociosidade que detecta a disponibilidade da máquina quando os recursos estão ociosos e interrompe qualquer execução se os recursos são considerados não ociosos.

- *Peer*: é o componente que gerencia, em cada *site*, o conjunto de máquinas que são disponibilizadas para a grade (ou seja, o *Worker*). Em geral, cada *site* contém um *Peer* instalado. O *Peer* junta-se à grade por meio de uma notificação de um serviço de descoberta de *Peers* e é imediatamente informado sobre a presença de outros *Peers* na grade.

- *Discovery Service*: é responsável por conectar múltiplos OurGrid, de modo que vários *Peers* podem interagir e trocar recursos computacionais.

- *Broker*: é responsável por fornecer aos usuários uma interface gráfica para submissão de suas aplicações. Além disso, o *Broker* escalona as aplicações para as máquinas da grade e gerencia sua execução.

Quando uma aplicação é submetida para execução, o *Broker* contacta o *Peer* local que verifica o número de máquinas que atendam às requisições

do usuário para execução da aplicação (exemplo, sistema operacional, quantidade mínima de memória, etc). Quando o *Peer* recebe cada requisição, esse tenta encontrar máquinas adequadas para executarem a aplicação. Se não houverem máquinas na grade local, o *Peer* tenta obter máquinas de outros *Peers* e espera pela resposta. Sempre que novas máquinas (local ou remota) estão disponíveis, o *Peer* local entrega estas máquinas para o *Broker* solicitante. Se uma ou mais máquinas são entregues para o *Broker*, esse escalona as tarefas e inicia o gerenciamento de suas execuções

3.6.6 Considerações sobre os trabalhos correlatos

Esta subseção apresenta uma análise comparativa entre os trabalhos relacionados descritos nas subseções anteriores, conforme pode ser vista na Tab. 3. As características a serem comparadas são descritas a seguir.

- **C1**: Ambiente de Grade Oportunista;
- **C2**: Reserva Antecipada de Recursos;
- **C3**: Reserva Antecipada a Pedido da Usuário;
- **C4**: Pesquisa por Intervalos Livres.

Tabela 3 – Comparação entre os trabalhos correlatos

Trabalhos de Pesquisa	C1	C2	C3	C4
Sulistio et. al., 2009	Não	Sim	Sim	Sim
Ferreira, Dantas e Bauer, 2010	Não	Sim	Sim	Não
Tomás et. al., 2011	Não	Sim	Sim	Não
Martins et. al., 2012	Sim	Sim	Não	Não
OurGrid, 2013	Sim	Não	Não	Não

O primeiro trabalho (SULISTIO et al., 2009) apresenta como principal proposta uma nova estrutura de dados, chamada de *Grid advanced reservation Queue* (GarQ), que é a combinação das estruturas de dados *Calendar Queue* e Árvore de Segmentos, anteriormente modificadas pelos autores. O principal objetivo da GarQ é o gerenciamento eficiente das operações de adicionar, excluir e pesquisar por reservas em um ambiente de grade computacional. Além disso, é proposto um novo mecanismo que busca por intervalos livres muito próximos do escolhido pelo usuário, caso já tenha sido rejeitado.

A pesquisa de Ferreira, Dantas e Bauer (2010) propõe um mecanismo de reserva antecipada de recursos que possibilita que usuários reservem recursos de diferentes *clusters*. No entanto, não possui o mecanismo de busca por intervalos livres. Entretanto, por se tratar de um ambiente *multi-cluster*, a busca por intervalos livres torna-se pouco necessária pois, a chance de reservar recursos no período solicitado pelo usuário é maior, uma vez que é possível pesquisar por nós disponíveis pertencentes a outros *clusters*.

O trabalho de Tomás et al. (2011) apresenta um *framework* para melhorar o meta-escalonamento antecipado em ambientes de grades computacionais e um sistema autônomo capaz de prever o tempo de execução de uma aplicação. Possui o mecanismo de reserva antecipada onde o usuário especifica parâmetros como tempo de início e tempo de completude da execução da aplicação. Não possui o mecanismo de busca por intervalos livres, porém permite a comunicação entre grades para a alocação da aplicação, caso a grade do usuário não atenda aos requisitos exigidos por este.

A pesquisa de Martins et al. (2012) propõe um mecanismo de gerenciamento, para ambientes de grade oportunista, baseada no escalonamento dinâmico e re-escalonamento. O trabalho apresenta o mecanismo de reserva antecipada, porém se diferencia dos outros trabalhos pois a reserva é realizada pelo próprio sistema. Em outras palavras, o usuário não tem a possibilidade de escolher pelo tempo de início e fim da execução, a reserva é feita caso o único recurso que atende às especificações do usuário esteja executando uma aplicação.

Por fim, o OurGrid (OURGRID, 2013) é uma infraestrutura *Peer-to-Peer* para grades oportunistas, onde laboratórios de todo o mundo disponibilizam seus recursos computacionais ociosos e utilizam recursos ociosos de outros laboratórios. No entanto, não possui o mecanismo de reserva antecipada em sua arquitetura.

3.7 CONSIDERAÇÕES SOBRE O CAPÍTULO

Neste capítulo foram apresentados conceitos que fundamentam a proposta de pesquisa. Primeiramente foram apresentados dois pontos de vista quanto ao conceito de reserva antecipada de recursos. O primeiro, e mais difundido, defende que a reserva antecipada é um mecanismo que oferece qualidade de serviço para o usuário pois, para utiliza-lo é preciso garantir a disponibilidade dos recursos no momento da execução. Por outro lado, o segundo ponto de vista, utilizado para fundamentar essa pesquisa, assume que o mecanismo de reserva antecipada aumenta a expectativa de que a execução ocorra. Com isso, entende-se que não há garantias de disponibilidade de re-

cursos e, consequentemente, de execução.

Foram apresentados também, conceitos sobre meta-escalonador, grade oportunista e a arquitetura do *middleware* InteGrade utilizado para o desenvolvimento da arquitetura proposta e do protótipo. E, por fim, foram apresentados alguns trabalhos correlatos e uma tabela comparando suas características.

4 RESERVA ANTECIPADA OPORTUNISTA

Este capítulo apresenta a proposta de um modelo e sua respectiva implementação no tocante a reserva antecipada de recursos para ambientes de grade oportunista. Esta proposta consiste na implementação de um módulo adicional a arquitetura do InteGrade, *middleware* para grades oportunistas, que permita que seus usuários reservem recursos para que sejam utilizados em um tempo futuro.

Inicialmente é apresentada uma introdução do trabalho proposto. Em seguida, é detalhado o modelo proposto, a arquitetura proposta, bem como o detalhe de sua implementação e o processo de execução de uma aplicação. Por fim, são apresentadas algumas considerações da proposta apresentada.

4.1 INTRODUÇÃO

Grade oportunista é um interessante paradigma quando se deseja obter computação de alto desempenho com baixo custo de investimento. Entretanto, as solicitações de execução nesse tipo de ambiente são realizadas de forma aleatória e desordenada. Essas solicitações desordenadas podem ocasionar sucessivas negações de execução, o que poderá se tornar um processo trabalhoso para o usuário, além de gerar uma possível sobrecarga no sistema devido à necessidade de sucessivas solicitações. O mecanismo de reserva antecipada de recursos é visto como uma maneira eficiente de ordenar as solicitações de execução, uma vez que essas são armazenadas em filas esperando que o horário que a execução ocorra.

No entanto, trabalhos como os de Sulistio e Buyya (2004), Castillo, Rouskas e Harfoush (2011) e Tomás et al. (2011) associam o uso do mecanismo de reserva antecipada com a garantia de disponibilidade dos recursos reservados e, consequentemente, com o oferecimento de qualidade de serviço para o usuário. Diante desse conceito, não é convencional o uso de reserva antecipada em grades oportunistas, devido a instabilidade desses ambientes e, consequentemente, a não garantia de disponibilidade de seus recursos.

Contudo, o trabalho de Foster et al. (1999) menciona que o mecanismo de reserva antecipada proporciona uma maior expectativa de que os recursos podem ser alocados quando exigidos. O autor ainda exemplifica esse conceito com a compra de um bilhete de avião ou concerto, que fornece uma maior expectativa de obter o assento.

Com base no conceito apresentado por Foster, este trabalho de pesquisa propõe um modelo e uma arquitetura que permite que usuários realizem

reservas antecipadas de recursos em ambientes de grades oportunistas. Conforme apresentados nos trabalhos de Gomes e Dantas (2013a) e (GOMES; DANTAS, 2013b), o objetivo principal da proposta é oferecer ao usuário um melhor esforço, uma vez que não é possível oferecer qualidade de serviço devido à instabilidade do ambiente. Esta proposta também busca aumentar a usabilidade do sistema, organizar as execuções e melhorar o escalonamento das aplicação pois, segundo Tomás et al. (2011), a reserva antecipada aumenta a previsibilidade do sistema. Além disso, é implementado um mecanismo de busca por intervalos livres, onde a reserva é realocada em um intervalo de tempo mais próximo do escolhido pelo usuário, uma vez que sua solicitação tenha sido previamente rejeitada. Este mecanismo baseia-se no conceito de busca por intervalos livres apresentado na pesquisa de Sulistio e Buyya (2004). É proposto ainda, um serviço de sugestões que é responsável por apresentar para o usuário a quantidade de recursos já reservados durante um determinado período de tempo. O objetivo desse serviço é auxiliar o usuário na escolha do período de tempo para sua reserva, possibilitando que essa escolha seja feita para um período mais ocioso, ou seja, com menos reservas cadastradas.

O ambiente utilizado para o desenvolvimento desta proposta foi o InteGrade. O InteGrade, conforme foi visto na seção 3.5, é um projeto de *middleware* para grades oportunistas que utiliza a capacidade computacional ociosa de estações de trabalho, laboratórios acadêmicos e computadores pessoais para executar aplicações que requerem alto poder computacional (GOLDCHLEGER et al., 2004). Seu principal objetivo é oferecer para os usuários da grade alto poder computacional sem degradar a performance de suas máquinas, priorizando sempre a execução local. O InteGrade foi escolhido para o desenvolvimento desta pesquisa pois não possui reserva antecipada em sua arquitetura básica e é um projeto brasileiro desenvolvido em parceria entre 5 universidades. Além disso, possui código aberto e de fácil acesso, bem como a disponibilidade de um suporte eficiente.

Adicionalmente à proposta descrita, é utilizado no serviço de sugestões um mecanismo de predição de uso de recursos, o LUPA, originalmente implementando no InteGrade. O objetivo de se utilizar o LUPA é para reforçar o auxílio ao usuário na escolha do período de tempo mais apropriado para a realização de sua reserva. O LUPA foi detalhado na seção 3.5.1.

O desenvolvimento desta proposta passou por algumas etapas. Primeiramente, foram desenvolvidos os métodos de reserva antecipada de recursos e de busca por intervalos livres para realocação da reserva previamente rejeitada. Posteriormente, os métodos propostos foram inseridos e adaptados para o InteGrade.

Em uma segunda etapa, foi desenvolvido o módulo responsável por

apresentar ao usuário um gráfico com as predições de disponibilidade de CPU e memória de todo o *cluster*, bem como a quantidade de recursos reservados durante um período de tempo. Com a visualização deste gráfico, o usuário poderá identificar o melhor período de tempo, ou seja, o período mais ocioso para solicitar sua reserva.

Por fim, em uma terceira etapa foram realizados testes e estudos de caso para visualizar o comportamento do sistema quanto às solicitações de reservas. Além disso, foram monitoradas algumas máquinas pertencentes ao Laboratório de Pesquisa em Sistemas Distribuídos (LaPeSD) (LAPESD, 2013), de forma a preencher o gráfico de predições de uso. Essa etapa será detalhada no próximo capítulo.

4.2 MODELO PROPOSTO

Uma visão geral do modelo proposto é apresentada na Fig. 14, o qual é composto por módulos responsáveis por permitir que usuários reservem recursos de forma oportunista para que sejam utilizados no futuro. Para tanto, o modelo possui como ambiente base o de grade oportunista com configuração de *cluster*, porém sua modelagem genérica se encaixa em qualquer tipo de ambiente distribuído. Assim, considera-se que o ambiente é composto por máquinas pessoais interconectadas e em constante uso, ou seja, não dedicadas e conectadas de maneira hierárquica por uma rede local.

É através da interface gráfica que o usuário interage com o sistema de reserva. Essa, por sua vez, se comunica com os módulos *Suggestion Service* e *Reservation Service* pertencentes ao meta-escalonador. O primeiro módulo é responsável por apresentar para o usuário informações que poderão auxiliá-lo na escolha de um horário para sua reserva, tais como, a quantidade de recursos reservados em um determinado período de tempo. Essas informações são apresentadas para o usuário na interface por meio de um gráfico.

Já o módulo *Reservation Service* é responsável pelo gerenciamento dos pedidos de reserva. Após receber as informações referentes à reserva, o módulo verifica na base de dados a disponibilidade do horário solicitado. Caso não haja reserva no período, a reserva é realizada com sucesso. Caso contrário, o módulo verifica o período de tempo disponível mais próximo do escolhido e realiza a reserva.

Conforme pode ser visualizado na Fig. 14, o meta-escalonador possui ainda os módulos *Execution Service* e *Notification Service*. O primeiro módulo tem a função de recuperar as reservas cadastradas na base de dados e verificar a existência de reservas a serem executadas no horário atual. Caso haja, é responsável também, por iniciar o processo de execução. Por fim,

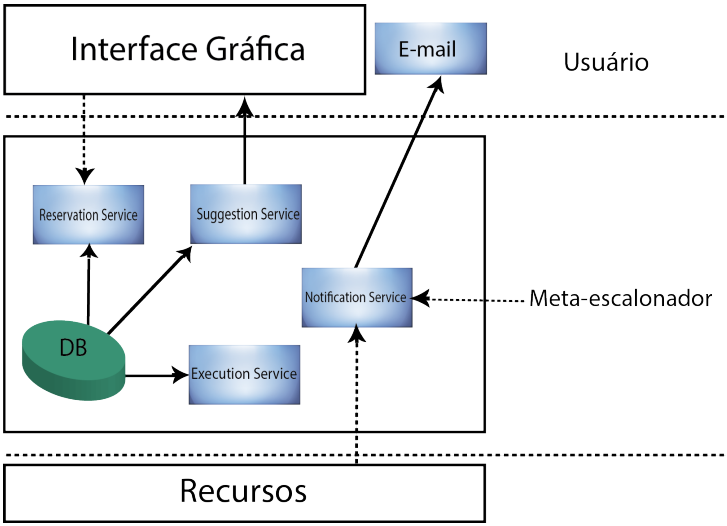

Figura 14 – Módulos pertencentes à arquitetura proposta

o módulo *Notification Service* é responsável por encaminhar aos usuários *e-mails* que os notifique sobre o estado da execução de sua reserva, ou seja, execução aceita, rejeitada ou completada.

A próxima subseção apresenta detalhes sobre a arquitetura proposta como complemento da infraestrutura do InteGrade.

4.3 ARQUITETURA PROPOSTA

Em complemento à arquitetura do InteGrade, propõe-se o módulo *Advance Reservation of Resources* (ARR). Conforme descrito anteriormente, este módulo possibilita que o usuário reserve antecipadamente recursos do *cluster* para que sejam utilizados futuramente. Conforme pode ser visto na Fig. 15, o ARR foi inserido como módulo adicional do servidor do InteGrade, ou seja, no Gerenciador do *Cluster*.

Por se tratar de um módulo adicional à arquitetura do servidor, a implementação do ARR foi realizada utilizando o JacORB para a comunicação remota com os outros módulos do InteGrade e a linguagem de programação Java. O objetivo foi utilizar a mesma tecnologia utilizada no InteGrade, de maneira a desenvolver uma extensão do que já foi implementado.

Aproveitou-se a interface implementada no InteGrade para o desen-

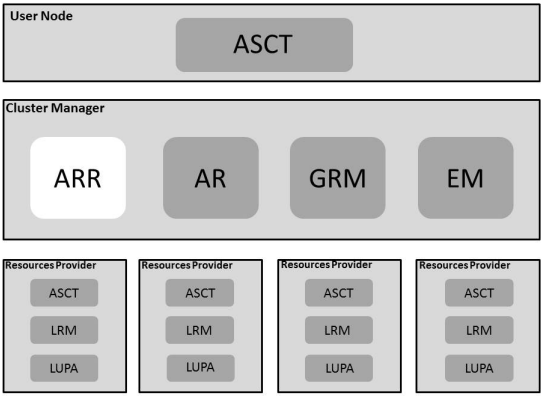

Figura 15 – Arquitetura proposta inserida no InteGrade

volvimento do módulo, apenas inserindo campos para que os requisitos da reserva fossem informados, bem como a possibilidade de visualização do gráfico de sugestões. Com isso, as especificações que eram informadas pelo usuário na arquitetura original do InteGrade (tipo de aplicação, quantidade de nós, quantidade de CPU e quantidade de memória livre) não sofreram mudanças.

A Figura 16 apresenta as principais classes e métodos que compõem o módulo ARR e que representam os quatro componentes descritos na seção 4.2. A classe `AdvancedReservationImpl` contém a implementação da classe `AdvancedReservationSkeleton` e, foi subtraída do diagrama de classes devido à falta de espaço. Esta classe é responsável por receber os dados remotos e encaminhá-los para as classes que farão sua manipulação.

4.3.1 Reservation Service

Esse módulo é responsável pela realização e gerenciamento das reservas e é representado pela classe `ReservationService`. Para realizar uma reserva, o usuário deve informar os seguintes requisitos:

- *start time*: tempo de início da execução;
- *end time*: tempo de término da execução;
- *e-mail*: e-mail do usuário para que receba as notificações sobre o estado

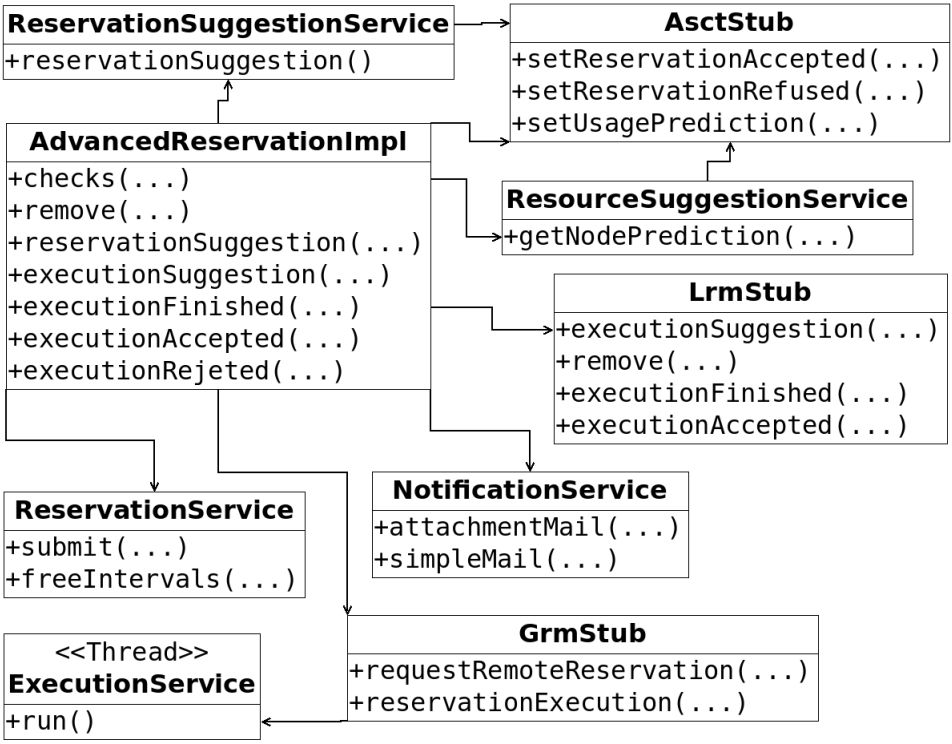

Figura 16 – Diagrama de classe do módulo ARR

da reserva (execução da reserva rejeitada por falta de nós disponíveis) ou da execução (execução aceita e iniciando ou execução concluída).

Em posse dessas informações, é verificado, inicialmente, se o período de tempo solicitado conflita com os cadastrados na base de dados. Caso haja conflitos, é verificado se no horário conflitante o nó reservado é o mesmo do escolhido pelo meta-escalonador. Caso seja o mesmo, é realizada uma busca por um intervalo de tempo mais próximo do escolhido pelo usuário para realizar a reserva. Então, após a reserva ser realizada com sucesso, o usuário recebe confirmação da reserva e é informado, através do `AsctStub`, sobre o *start time* e o *end time* no qual seu pedido de reserva foi cadastrado.

Conforme apresentado anteriormente, no momento da solicitação de execução, o usuário pode especificar quantidade de recursos e a busca por nós é realizada com base nessa especificação. No entanto, este módulo realiza a reserva de todo o nó e não dos recursos requisitados pelo usuário.

A reserva antecipada proposta possui alguns estados em seu ciclo de vida, conforme apresentados na Fig. 17, podendo esta estar em qualquer um deles.

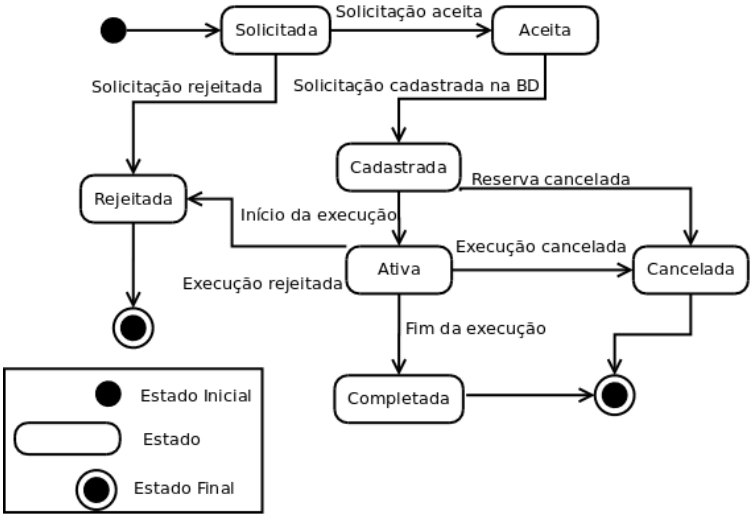

Figura 17 – Estados da reserva antecipada

As definições de cada estado são descritas as seguir:

- **Solicitada**: primeiro estado da reserva, quando o usuário solicita por

uma reserva antecipada e informa os requisitos;

- **Rejeitada**: a solicitação pode ser rejeitada caso não haja recursos durante a solicitação, no início da execução, ou intervalo livre que atendam às especificações do usuário;
- **Aceita**: a solicitação foi aceita, ou seja, o período de tempo ou os recursos estão disponíveis para o usuário;
- **Cadastrada**: o pedido de reserva é cadastrado na base de dados;
- **Cancelada**: a reserva pode ser cancelada enquanto cadastrada na base de dados ou durante sua execução;
- **Ativa**: inicia o processo de execução da aplicação;
- **Completada**: o tempo final da reserva é alcançado.

Durante a implementação, viu-se a necessidade de determinar duas regras para a realização da reserva, que foram:

- Só pode ser realizada uma reserva por usuário. Esse verificação é realizada com base no *e-mail* cadastrado. O objetivo dessa regra é impossibilitar a monopolização do ambiente;
- As reservas só poderão ser feitas durante 24 horas a partir da hora da solicitação. O objetivo dessa regra é evitar reservas para períodos muito futuros.

4.3.1.1 Aspectos da Implementação do Módulo Reservation Service

O algoritmo 1 apresenta o primeiro processo para o pedido de reserva antecipada. Abaixo são descritas as variáveis citadas no algoritmo.

- *List*: lista de reservas cadastradas na base de dados;
- *tempList*: lista temporária que armazenará as reservas com horários conflitantes com o solicitado;
- $start_{new}$: tempo de início solicitado;
- end_{new}: tempo de término solicitado;
- $start_i$: tempo de início na posição i da *List*;

Algorithm 1 Submissão da Reserva

```
 1: if List estiver vazia then
 2:     reserva aceita e salva na base de dados
 3: else
 4:     for i = 0 até o tamanho da List do
 5:         if as seguintes propriedades forem verdadeiras then
 6:             coloque i na tempList
 7:             - start_new ≤ start_i e end_new ≥ end_i
 8:             - start_new ≤ end_i
 9:             - end_new ≤ end_i
10:         end if
11:     end for
12: end if
13: if tempList estiver vazia then
14:     salve a reserva na base de dados
15: else
16:     for i = 0 até o tamanho da tempList do
17:         if as seguintes propriedades não forem verdadeiras then
18:             remova i da tempList
19:             - start_new ≤ start_i e end_new ≥ start_i
20:             - start_new ≤ start_i e end_new ≥ end_i
21:             - start_new ≥ start_i e start_new ≤ end_i
22:             - start_new ≥ start_i e end_new ≤ end_i
23:         end if
24:     end for
25: end if
26: if tempList estiver vazia then
27:     salve a reserva na base de dados
28: else if o nó não estiver reservado para o mesmo start e end time then
29:     salve a reserva na base de dados
30: else
31:     chame o algoritmo Intervalos Livres
32: end if
```

- end_i: tempo de término na posição i da *List*.

As linhas (1) e (2) verificam a existência de reservas cadastradas na base de dados. Se *List* estiver vazia, a reserva é cadastrada na base de dados. Caso contrário, as linhas (4) a (9) procuram superficialmente por conflitos entre as reservas cadastradas e a nova reserva. Em outras palavras, o algoritmo verifica a existência de reservas que iniciam no mesmo horário do pedido de reserva. Existindo conflitos nessa fase, os dados da reserva conflituosa são armazenados na *tempList*. Essa primeira etapa do algoritmo foi baseada no algoritmo de Sulistio e Buyya (2004). Se *tempList* estiver vazia a reserva é salva na base de dados, linhas (13) e (14). Caso contrário, é realizada uma busca mais minuciosa por conflitos, linhas (16) a (22). Então, é verificado se as propriedades descritas nas linhas (19) a (22) são falsas. Se sim, a reserva

que foi comparada com a nova reserva é retirada da *tempList*. Caso a *tempList* se torne vazia, a nova reserva é cadastrada na base de dados. Caso contrário, se o nó escolhido pelo meta-escalonador para a nova reserva não for o mesmo da reserva conflituosa, a reserva é realizada. Senão é chamado o algoritmo 2.

O algoritmo 2, em complemento ao algoritmo anterior, tem como objetivo buscar por intervalos livres mais próximo do período solicitado pelo usuário. As variáveis de mesmo nome do algoritmo anterior, possuem as mesmas descrições e funcionalidades.

Algorithm 2 Intervalos Livres

```
 1: for i = 0 até o tamanho da List do
 2:     if tempList_id for igual id_i then
 3:         duration = end_new − start_new
 4:         if i + 1 == List then
 5:             start_new = end_i + 5 minutos
 6:             end_new = start_new + duration
 7:         else
 8:             durationAux = start_i(i + 1) − end_i
 9:             if durationAux >= duration + 10 minutos then
10:                 start_new = end_i + 5 minutos
11:                 end_new = start_new + duration
12:             end if
13:         end if
14:     end if
15: end for
```

Na linha (2) é verificado, dentro de um bloco de repetição, se o *id* (número de identificação da reserva na base de dados) da reserva conflituosa é igual ao *id* da reserva que está na posição *i* da *List*. Caso seja, é calculada a duração da nova reserva, linha (3). Então, é verificado entre as linhas (4) e (6) se a posição *i* corresponde ao último elemento da *List*. Se sim, o tempo de início da nova reserva é modificado para o tempo de término da última reserva mais cinco minutos (esse tempo é acrescido para que o tempo de início da nova reserva não seja igual ao tempo de término da reserva em questão), linha (5), e o tempo de término da nova reserva o tempo de início da mesma reserva mais a duração, linha (6). Se não, *durationAux* recebe o intervalo de tempo entre a reserva na posição *i* e a próxima reserva, linha (8). A linha (9) verifica se a nova reserva se encaixa, em termos de duração, nesse intervalo. É acrescentado dez minutos ao intervalo para que seja possível acrescentar cinco minutos no tempo de início e cinco no de término. Se for possível incluir a nova reserva neste intervalo, os procedimentos das linhas (5) e (6) descritos acima são repetidos.

4.3.2 Suggestion Service

O módulo *Suggestion Service* é responsável por apresentar para o usuário a quantidade de recursos reservados, bem como as predições de uso do *cluster*. Para isso, esse módulo é dividido em duas classes principais: `ReservationSuggestionService` e `ResourceSuggestionService`. A primeira classe recupera da base de dados uma lista contendo todas as reservas cadastradas e verifica o tempo de duração de cada reserva. Com isso, é possível verificar o intervalo de tempo que uma certa quantidade de recursos são utilizados. O cálculo realizado para estimar a quantidade de recursos reservados é de média simples. Em outras palavras, cada máquina reservada representa uma porcentagem de recursos utilizados, dependendo do número de máquinas cadastradas no *cluster*. Por exemplo, se houvesse três máquinas cadastradas no *cluster* e o usuário reservasse uma máquina, então estaria utilizando 33% dos recursos do *cluster* durante o intervalo de tempo reservado.

Por outro lado, a classe `ResourceSuggestionService` recebe as informações sobre as predições de uso de recursos e a quantidade total de memória do nó do `LrmStub`. Para a realização das estimativas de uso da CPU, também é realizado o cálculo de média simples. As predições encaminhadas se referem a porcentagem de uso de CPU em intervalos de cinco minutos porém, são consideradas apenas as predições relativas a cada hora, para diminuir a quantidade de informações apresentadas no gráfico. Essas predições são, então, somadas e divididas pela quantidade de máquinas pertencentes ao *cluster*. Já na predição de uso de memória utiliza-se um cálculo diferente. As informações encaminhadas são de predições de memória livre em intervalos de cinco minutos. Com isso, são somadas as predições de cada máquina em cada hora e subtraídas da quantidade total de memórias do *cluster* de modo a estabelecer a predição de uso de memória. Após isso são convertidos em porcentagem para melhor visualização e entendimento do gráfico.

Após isso, as informações de predição de uso e quantidade de recursos reservados são encaminhadas para o `AsctStub`, que será responsável por apresentá-las graficamente para o usuário.

4.3.2.1 Aspetos da Implementação do Módulo Suggestion Service

Esta subseção apresenta o algoritmo 3, que permite a visualização da quantidade de recursos reservados, do módulo *Suggestion Service*. Abaixo são descritas as variáveis utilizadas nesse algoritmo.

- *listReservation*: lista contendo as reservas cadastradas na base de da-

dos;

- *reservationTime*: lista de inteiros para armazenar os horários que possuem recursos reservados;
- *reservationTotal*: lista para armazenar a quantidade de recursos reservados em porcentagem;
- $startTime_i$: tempo de início da *listReservation* na posição *i*;
- $differentNode_i$: variável booleana que informa a necessidade de executar a aplicação em diferentes nós da *listReservation* na posição *i*;
- *date*: recebe a hora convertida para o formato HH da *listReservation* na posição *i*;
- *numMachine*: número de nós solicitados pelo usuário;
- *reservation*: variável do tipo TreeMap que armazena *date* como chave e como valor a variável *aux*, *numMachine* ou 1.

A linha (1) verifica se a lista de reservas recuperadas da base de dados está vazia. Se sim, é inserido 0 em cada posição da lista para que se apresente um valor no gráfico, linhas (2) a (5). Se não, dentro de um bloco de repetição é determinado a duração da reserva da posição *i* na linha (8). Na linha (10) a duração é transformada em horas, uma vez que os tempos são armazenados em milissegundos na base de dados. Dentro de um bloco de repetição é verificado, na linha (16), se o usuário exigiu que as execuções fossem executadas em máquinas diferentes. Caso tenha exigido, é somada a quantidade de máquinas solicitadas com as já reservadas no determinado período de tempo, linhas (20) a (23). Caso contrário, é somado 1 na quantidade de máquinas já reservadas, linhas (29) a (31). Por fim, nas linhas (38) a (41) quantidade de máquinas reservadas é transformada em porcentagem e a chave e os valores da *reservation* são repassados para as listas *reservationTime* e *reservationTotal*, respectivamente.

4.3.3 Execution Service

O módulo *Execution Service* responsável por verificar a existência de execuções para a hora atual e por iniciar o processo de execução é representado pela classe `ExecutionService`. Essa classe recupera as informações da base de dados e executa uma *thread* a cada um minuto para verificar

Algorithm 3 Uso de recursos do *cluster* pelas reservas cadastradas na base de dados

```
1: if listReservation estiver vazia then
2:     for i = 0 até 24 do
3:         reservationTime = i
4:         reservationTotal = 0
5:     end for
6: else
7:     for i = 0 até o tamanho de listReservation do
8:         duration = endTime_i − startTime_i
9:         date = startTime_i
10:        time = duration/3600000 milissegundos
11:        mod = duration módulo 3600000 milissegundos
12:        if mod ≠ 0 then
13:            time = time + 1
14:        end if
15:        for j = 0 até time + 1 do
16:            if differentNode_i then
17:                if não tiver date em reservation then
18:                    coloque date e numMachine em reservation
19:                else
20:                    aux = valor de reservation_date
21:                    aux = aux + numMachine
22:                    coloque date e aux em reservation
23:                    date = date + 1
24:                end if
25:            else
26:                if não tiver date em reservation then
27:                    coloque date e 1 em reservation
28:                else
29:                    aux = valor de reservation_date
30:                    aux = aux + 1
31:                    coloque date e aux em reservation
32:                end if
33:                date = date + 1
34:            end if
35:        end for
36:    end for
37: end if
38: for até o tamanho de reservation do
39:     coloque date de reservation em reservationTime
40:     sum = (valor de reservation/qtdMaquinas) * 100
41:     reservationTotal = sum
42: end for
```

a existência de execuções. Existindo, esta encaminha as informações necessárias para a realização da execução diretamente para o `GrmStub`, que dará prosseguimento ao processo de execução.

A lista de execuções é ordenada primeiramente pelo *start time*. Caso haja *start time* iguais, é utilizada a política de o primeiro a chegar é o primeiro a ser servido (*First-come, First Served* - FCFS). Em outras palavras, a segunda ordenação é realizada de acordo a ordem de pedido, ou seja, o primeiro pedido tem prioridade sobre o segundo, e assim sucessivamente.

4.3.4 Notification Service

Esse módulo é representado pela classe `NotificationService` e é responsável por informar o usuário sobre o estado da execução (aceite, rejeite ou completude). Para isso, recebe informações vindas tanto da classe `GrmStub` quanto da classe `LrmStub`. As informações são vindas da primeira classe quando a execução é rejeitada por falta de máquinas disponíveis no *cluster* que atendam as especificações do usuário para a execução da aplicação. Já as vindas da classe `LrmStub` se referem ao aceite e à completude da execução. Na notificação de completude da execução são enviados também os arquivos de saída, para que o usuário tenha acesso aos seus resultados no momento da ciência do estado da execução.

4.4 PROCESSO DE EXECUÇÃO DE UMA APLICAÇÃO

A Figura 18 apresenta o processo de execução de uma aplicação no InteGrade utilizando o mecanismo de reserva antecipada, o qual será detalhado a seguir.

O processo de execução se inicia com o registro da aplicação no *Application Repository*. Em seguida, o usuário poderá escolher entre a execução imediata e a execução reservada. Com a escolha da opção de execução reservada o usuário pode optar por visualizar o gráfico com as informações de predição de uso do *cluster* e a quantidade de recursos reservados, para auxiliar na escolha dos requisitos da reserva (*start time, end time*). Em seguida, o usuário pode especificar os requisitos de recursos para a execução da aplicação, conforme descrito na seção 3.5, e, adicionalmente, informar os requisitos para reserva.

Com as exigências de recursos e reservas realizadas, o usuário submete sua solicitação de execução reservada. A partir deste momento, o ASCT envia as informações referentes às especificações de recursos para o GRM.

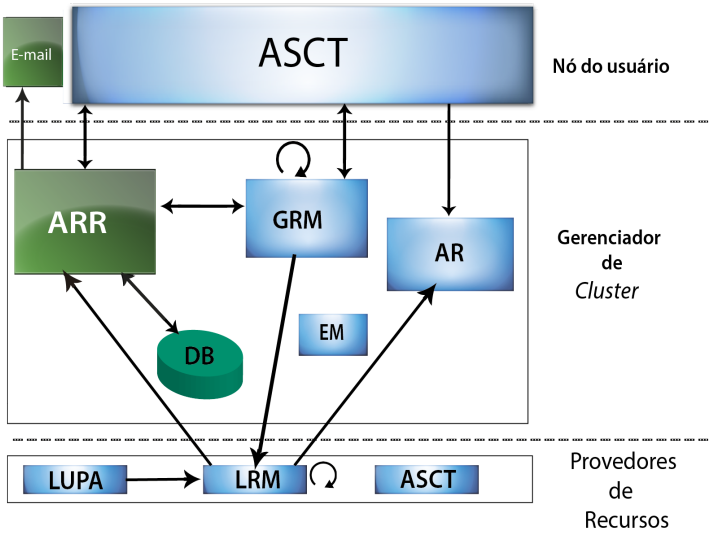

Figura 18 – Processo de execução de uma aplicação no InteGrade com reserva antecipada

Este, por sua vez, verifica se existem nós no *cluster* que atendam às exigências solicitadas pelo usuário. Caso não exista, o usuário é notificado pelo ASCT. Caso contrário, o ASCT envia as informações referentes à reserva para o ARR. Então, são realizadas as seguintes verificações:

- Verifica se a data final é posterior a data inicial;
- Verifica se a data inicial não é maior do que a hora atual mais 24 horas;
- Verifica se existem reservas armazenadas na base de dados com o mesmo *e-mail* da reserva solicitada;

Após estas verificações, o *Reservation Service* busca na base de dados por reservas no período solicitado. Se não houver, a reserva é realizada com sucesso. Caso contrário, o módulo ainda verifica se a máquina reservada no tempo conflitante é a mesma da escolhida pelo GRM para o novo pedido de reserva. Caso sejam diferentes, a reserva é realizada com sucesso. Se não, o módulo realiza a reserva em um período de tempo disponível mais próximo do solicitado. Com a realização da reserva, as informações referentes aos recursos, à aplicação, à reserva e possíveis arquivos de entrada são armazenadas na base de dados.

Chegada a hora da execução, o *Execution Service* recupera as informações da base de dados e as envia para o GRM, o qual verificará novamente a disponibilidade do nó previamente selecionado para a execução da aplicação. Se o nó não pertencer mais ao *cluster* ou estiver executando uma aplicação, é realizada nova busca de nós que atendam as exigências do usuário. Caso não haja, a execução não ocorrerá e a reserva falhará.

Após a escolha do nó candidato, o GRM envia uma solicitação de execução para o LRM do nó. Neste momento, o LRM verifica se, de fato, possui recursos disponíveis. Este procedimento é necessário pois o GRM mantém uma visão aproximada da disponibilidade dos recursos de cada nó do *cluster*. Caso não haja recursos disponíveis no nó, o LRM notifica o GRM, que torna a procurar por outro nó candidato. Entretanto, caso o nó possa executar a aplicação, o LRM solicita a aplicação ao *Application Repository* e a executa. O LRM, então, envia a notificação de aceite e posteriormente de completude da execução tanto para o *Notification Service*, que notifica o usuário por e-mail quanto para o ASCT, que o notifica pela interface gráfica.

Contudo, se no momento de iniciar a execução o nó se desconecta do *cluster*, o GRM busca outro nó para fazer a realocação. Caso não haja nós disponíveis a execução da reserva falhará.

4.5 CONSIDERAÇÕES SOBRE O CAPÍTULO

O modelo proposto permite que usuários reservem recursos computacionais para que sejam utilizados em um tempo no futuro. Com a abordagem proposta, é possível distribuir as solicitações de alocações de recursos durante todo o período do dia e não apenas realizá-las na hora em que o usuário solicita execução. Para a implementação da arquitetura proposta utilizou-se a infraestrutura do *middleware* InteGrade, o que possibilitou o aproveitamento e adaptação da interface gráfica, bem como do módulo de predição de uso, o LUPA.

Devido à dinamicidade do ambiente oportunista e a constante incerteza de que as máquinas continuarão conectadas ao *cluster*, torna-se difícil garantir que a reserva seja executada com sucesso. No entanto, a abordagem proposta ainda se torna válida, tendo em vista que em um ambiente oportunista sem reserva antecipada também há incertezas de continuidade de conexão e, consequentemente, possibilidade de interrupção da execução.

5 AMBIENTE E RESULTADOS EXPERIMENTAIS

Este capítulo apresenta o ambiente e os resultados experimentais obtidos por meio de estudos de casos que apresentam o comportamento do InteGrade com e sem o uso da reserva antecipada. O objetivo é mostrar os benefícios e as limitações existentes na proposta.

Primeiramente, na seção 5.1, é descrito o ambiente utilizado. Na seção 5.2 são apresentados os estudos de casos. Por fim, na seção 5.3, são apresentadas as considerações finais, bem como uma tabela que compara a abordagem proposta com os trabalhos relacionados.

5.1 AMBIENTE EXPERIMENTAL

O ambiente experimental é composto por quatro máquinas pertencentes ao Laboratório de Pesquisa em Sistemas Distribuídos (LaPeSD) (LAPESD, 2013). A Tabela 4 apresenta as características de *hardware* e as funções de cada máquina. Todas as máquinas operam com o sistema Ubuntu. O Gerenciador do *Cluster* é uma máquina totalmente dedicada, ou seja, opera apenas como servidor. Já as máquinas provedoras de recursos pertencem à pesquisadores do laboratório e estão em constante uso.

Tabela 4 – Características do ambiente experimental

Máquinas	**Qtd. de Memória (MB)**	**Qtd. de CPU**	**Funções**
athos	991	1	Gerenciador do *Cluster*
lapesd	3015	1	Provedor de Recursos Nó Usuário
lapesdrt	4024	1	Provedor de Recursos Nó Usuário
porthos	3015	1	Provedor de Recursos

Após a configuração do ambiente, foram realizadas algumas reservas de recursos para a execução futura de aplicações sequenciais e paramétricas. Foram apresentadas na Tab. 5 apenas as informações julgadas importantes para o entendimento dos estudos de casos. Na primeira coluna são apresentados nomes fictícios de usuários com o intuito de identificar a reserva na tabela de alocação (*Timeslot Table*). Nas segunda e terceira colunas são mostradas as horas de início e fim da execução da reserva. As últimas colunas apresentam, respectivamente, o tipo de aplicação e a quantidade de nós necessários e

reservados para a execução.

Conforme descrito no capítulo 4, o usuário poderá realizar a reserva antecipada durante 24 horas a partir da hora da solicitação. No entanto, optou-se por realizar as reservas durante um período de 12 horas (12h às 24h) com a finalidade de diminuir o tamanho da amostra.

Tabela 5 – Reservas cadastradas no banco de dados

Usuário	**Início**	**Fim**	**Aplicação**	**Qtd. Nó**
User A	12h	13h	Sequencial	1
User B	12h	14h	Sequencial	1
User C	13h	14h	Sequencial	1
User D	14h	15h	Paramétrica	2
User E	17h	18h	Paramétrica	2
User F	18h	20h	Sequencial	1
User G	20h	21h	Sequencial	1
User H	20h	22h	Sequencial	1
User I	21h	23h	Paramétrica	2
User J	23h	24h	Sequencial	1

A Figura 19 apresenta o gráfico de predições de uso dos recursos (CPU e memória) e quantidade de recursos reservados no intervalo entre 00h e 24h do mesmo dia. Para a obtenção dos resultados de predição, as máquinas provedoras de recursos foram monitoradas pelo LUPA durante um período de 24 dias. Por, aproximadamente, uma semana, a máquina *lapesd* ficou ociosa, apenas executando o InteGrade. O comportamento dessa máquina pode ter reduzido significativamente a média de uso do *cluster*.

Ao analisar o gráfico, pode-se visualizar que o constante uso dos recursos computacionais do *cluster* inicia-se, aproximadamente, às 07h e termina por volta das 19h. Com isso, o melhor período de tempo para realizar uma reserva seria entre 00h e 06h e 19h e 23h pois, o ambiente se mostra mais ocioso. Por outro lado, há maiores quantidades de recursos reservados nos períodos entre 13h e 14h, 18h, 20h e 21h e, 23h. Vale ressaltar que em alguns horários como às 14h, 18h, 20h, 21h e 23h, ocorrem picos de 100% de utilização pois, se refere ao tempo de término de uma execução e início de outra.

A próxima seção descreve os estudos de caso realizados com base no ambiente descrito nessa seção.

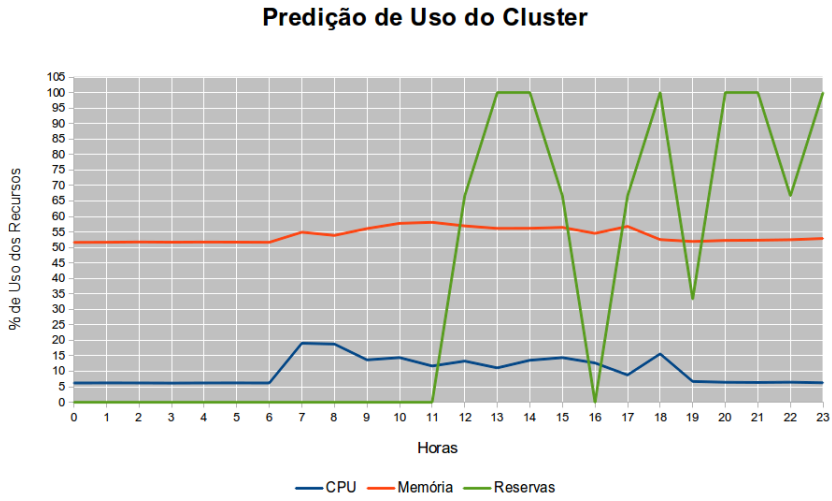

Figura 19 – Gráfico de predições e recursos reservados

5.2 ESTUDOS DE CASO

Para a obtenção dos resultados experimentais foram realizados dois estudos de caso que ilustram o comportamento do InteGrade sem e com a implementação da arquitetura proposta nesta dissertação. O primeiro estudo de caso, apresentado na subseção 5.2.1 está dividido em dois cenários. O primeiro cenário apresenta a requisição por recursos de um usuário para a execução de sua aplicação sendo atendida. Já o segundo, mostra a solicitação de um segundo usuário sendo recusada por falta de recursos disponíveis no período solicitado.

Por outro lado, o segundo estudo de caso, detalhado na subseção 5.2.2 está dividido em três cenários. No primeiro, um usuário solicita reserva que é realizada com sucesso. Já no segundo cenário, a solicitação de um segundo usuário é recusada por falta de recursos no período solicitado e esta, então, é realocada pelo meta-escalonador. Por fim, no terceiro cenário uma das máquinas provedoras de recursos é desconectada do *cluster* o que provoca a realocação das reservas, bem como a falha de execuções das reservas que não puderam ser realocadas.

Nestes experimentos, considerou-se que quando a solicitação de execução é atendida pelo provedor de recursos a execução da aplicação é

Tabela 6 – Requisições de execução imediata

Usuário	Aplicação	Qtd. Nó
User Y	sequencial	1
User Z	paramétrica	2

completada com sucesso. Além disso, não foram especificadas quantidades de recursos computacionais, apenas quantidades de nós. As alocações das reservas são feitas respeitando a heurística previamente implementada no InteGrade, que realiza um rodízio de escolha das máquinas de modo a não utilizar sempre a mesma.

5.2.1 InteGrade sem Reserva Antecipada

Nesse estudo de caso, buscou-se mostrar o comportamento do InteGrade nas execuções imediatas. Para tanto, são realizadas duas solicitações de execução, conforme apresentadas na Tab. 6. O **User Y** solicita execução imediata para um aplicação do tipo sequencial e o **User Z** para execução de uma aplicação do tipo paramétrica com necessidade de dois nós diferentes.

No entanto, no mesmo período das solicitações há uma aplicação paramétrica em execução que ocupa duas das três máquinas disponíveis no *cluster*. Conforme pode ser visto na tabela de alocação, na Fig. 20, a solicitação do **User Y** é atendida com sucesso, uma vez que ainda há recursos disponíveis.

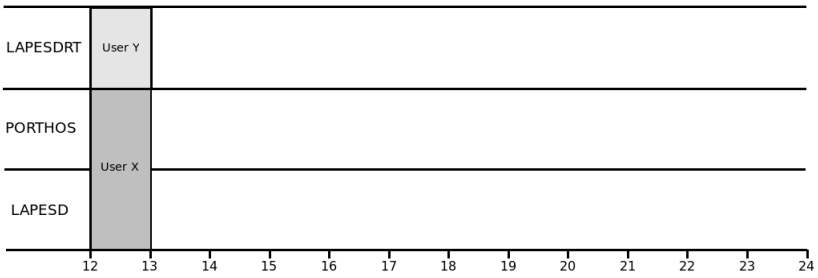

Figura 20 – Tabela de alocação com requisições para execução imediata com disponibilidade de recursos

Por outro lado, o **User Z** tem sua solicitação recursada, já que não

há recursos computacionais disponíveis para execução de sua aplicação no período de tempo requisitado. Conforme pode ser analisado na tabela de alocação na Fig. 21 há um excesso de solicitações em um mesmo período de tempo, das 12h às 13h, que poderiam ser distribuídas ao longo das 12 horas disponíveis.

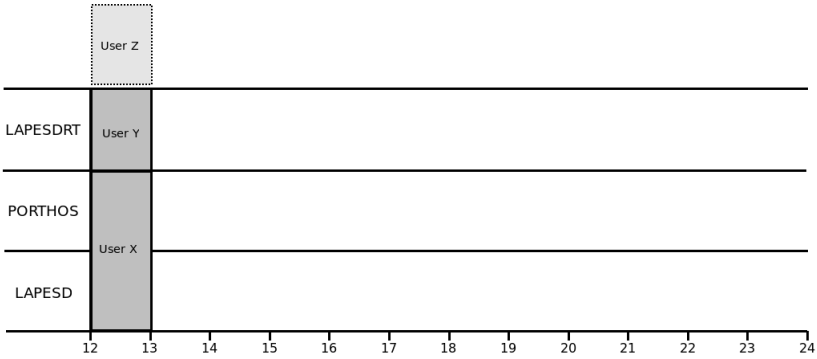

Figura 21 – Tabela de alocação com requisições para execução imediata sem disponibilidade de recursos

É importante destacar que o **User Z** terá que repetir a sua solicitação até que haja recursos disponíveis, ou seja, até que as aplicações dos usuários **User X** e **User Y** completem sua execução.

5.2.2 InteGrade com Reserva Antecipada

O objetivo desse estudo de caso é apresentar o funcionamento da arquitetura proposta. Para isso, dois usuários solicitam reservas de recursos, conforme pode ser visto na Tab. 7.

Tabela 7 – Tabela de requisições

Usuário	Início	Fim	Aplicação	Qtd. Nó
User K	12h	13h	sequencial	1
User L	13h	15h	paramétrica	2

O usuário **User K** que solicita uma máquina, para o período entre 12h e 13h tem sua reserva realizada com sucesso pois há recursos disponíveis que

atendam as exigências do usuário. A alocação da reserva pode ser vista na tabela de alocações apresentada na Fig. 22.

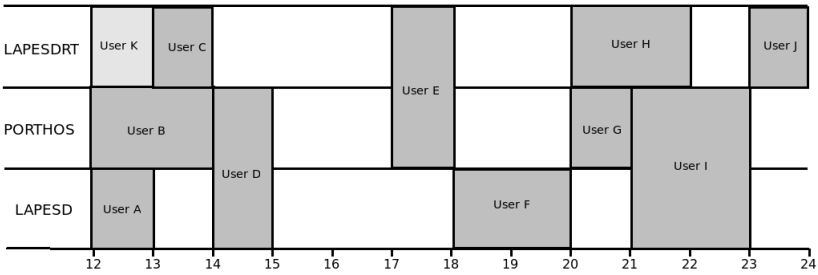

Figura 22 – Tabela de alocação com solicitação de reserva em um período disponível

Por outro lado, o usuário **User L** que solicita reserva de duas máquinas para o período entre 13h e 15h tem sua solicitação recusada. Conforme pode ser visto na tabela de alocação, na Fig. 23, há apenas um nó disponível para esse horário, o que inviabiliza o pedido de reserva. Contudo, para que o usuário não tenha que solicitar novamente, ou até, que tenha que repetir esse processo muitas vezes, o módulo proposto realoca sua reserva para um período de tempo mais próximo do escolhido. Neste caso, a reserva é realocada para o período entre 15h e 17h, conforme pode ser analisado na tabela de alocação na Fig. 24.

Em uma terceira situação, a máquina *lapesd* foi desconectada do *cluster* minutos antes do início da primeira execução, das 12h às 13h. Ao chegar o horário da execução de cada reserva, o meta-escalonador verifica a disponibilidade do(s) nó(s) reservado(s). Como a máquina *lapesd* não pertence mais ao *cluster*, o meta-escalonador pesquisa por outra máquina que satisfaça as requisições do usuário para realocar a reserva. Caso não haja máquinas disponíveis a reserva falhará.

A tabela de alocação apresentada na Fig. 25 mostra que as execuções das reservas dos usuários **User A** e **User I** falharam pois, não há recursos disponíveis para o período de tempo previamente reservado. Por outro lado, as reservas dos usuários **User D** e **User F** foram realocadas para as máquinas *porthos* e *lapesdrt* e, *porthos*, respectivamente, pois, estão disponíveis no tempo da reserva.

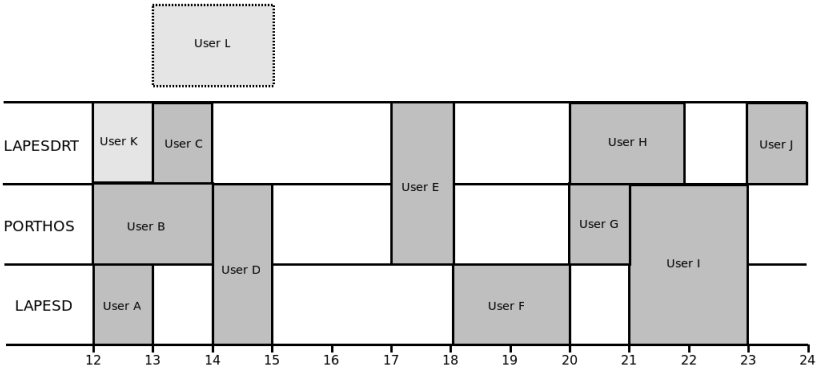

Figura 23 – Tabela de alocação com solicitação de reserva em um período indisponível

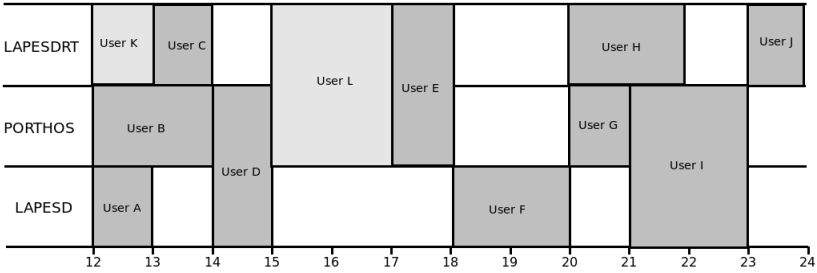

Figura 24 – Tabela de alocação com a realocação da reserva

5.3 CONSIDERAÇÕES SOBRE O CAPÍTULO

Os estudos de caso indicam que a implementação da proposta obteve um grau de sucesso satisfatório. A abordagem, por sua vez, permitiu que o usuário, por meio da reserva antecipada, utilizasse recursos no recorrer do dia e não apenas no horário da solicitação. Pôde-se verificar também, que o serviço de sugestões se mostrou eficiente pois, conflitos de horários de reservas podem ser evitados. Além disso, o serviço proporciona um aumento na probabilidade de que a execução ocorra, devido às predições de uso.

No entanto, pode-se constatar que o mecanismo de reserva antecipada pode falhar, ou seja, não há garantias de disponibilidade de recursos e, consequentemente, de execução da reserva. Com isso, fica claro a impossibili-

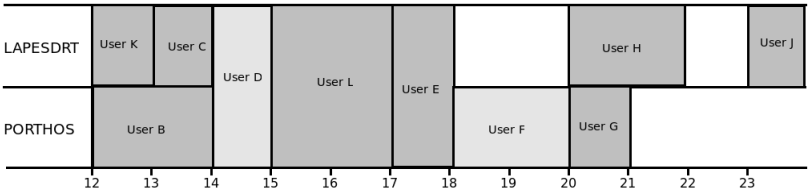

Figura 25 – Tabela de alocação com a realocação das reservas de uma máquina ausente

dade de oferecer qualidade de serviço devido à instabilidade que o ambiente oportunista apresenta. Diante disso, este trabalho de pesquisa apresenta um melhor esforço para o usuário quanto à utilização do sistema.

A Tabela 8 apresenta uma análise comparativa entre as características dos trabalhos correlatos e a abordagem proposta neste trabalho, que indicam um diferencial da proposta apresentada por esta dissertação. As características a serem comparadas são descritas a seguir.

- **C1**: Ambiente de Grade Oportunista;
- **C2**: Reserva Antecipada de Recursos;
- **C3**: Reserva Antecipada a Pedido da Usuário;
- **C4**: Pesquisa por Intervalos Livres.

Tabela 8 – Comparação entre os trabalhos correlatos

Trabalhos de Pesquisa	C1	C2	C3	C4
Sulistio et. al., 2009	Não	Sim	Sim	Sim
Ferreira, Dantas e Bauer, 2010	Não	Sim	Sim	Não
Tomás et. al., 2011	Não	Sim	Sim	Não
Martins et. al., 2012	Sim	Sim	Não	Não
OurGrid, 2013	Sim	Não	Não	Não
Este trabalho de pesquisa	Sim	Sim	Sim	Sim

6 CONCLUSÕES E TRABALHOS FUTUROS

A presente dissertação de mestrado forneceu uma revisão bibliográfica sobre ambientes distribuídos, bem como uma divisão e classificação dos ambientes oportunistas. Além disso, observou-se que, de acordo com um dos conceitos encontrados na literatura, o mecanismo de reserva antecipada pode ser utilizado em ambientes dinâmicos, como o de grade oportunista pois, não se torna necessário garantir a disponibilidade do recurso no período de que inicia sua utilização.

Com isso, neste trabalho de pesquisa foram abordados problemas de alocação de recursos em ambientes dinâmicos como os de infraestrutura oportunista. Observou-se que quando vários usuários solicitam recursos em um mesmo tempo, esgota-se a disponibilização desses recursos o que impossibilita o usuário de executar sua aplicação. Desta forma, este trabalho buscou a melhor maneira de resolver o problema descrito anteriormente. Em outras palavras, buscou-se melhorar a vazão do uso dos recursos, bem como disponibilizar um maior período para a alocação dos recursos computacionais.

Diante disso, foi proposto um modelo e uma arquitetura de reserva antecipada para grades oportunistas. A abordagem proposta possibilita que recursos sejam reservados para serem utilizados em um tempo no futuro. Com isso, é possível aumentar a quantidade de horários para alocação dos recursos, uma vez que estes poderão ser utilizados a qualquer período e não somente na hora que é solicitada a execução. Além isso, foi proposto um serviço que apresenta para o usuário a predição da quantidade de recursos que serão utilizados e a quantidade de recursos (CPU e Memória) reservados do *cluster*.

Para o desenvolvimento de uma arquitetura foi utilizado o ambiente InteGrade. Foram realizados dois estudos de caso com o objetivo de visualizar o comportamento do InteGrade sem e com a arquitetura proposta. Os resultados mostraram a eficiência da arquitetura para organizar as execuções e controlar a vazão das solicitações de alocação de recursos. A arquitetura também, apresentou resultados satisfatórios quanto à utilização do sistema por parte do usuário pois caso não existam recursos para o período solicitado, sua reserva é realocada para o período de tempo mais próximo. Além disso, a abordagem proposta auxilia o usuário na hora de escolher um período para a sua reserva, uma vez que é apresentado um gráfico contendo as predições de uso dos recursos, bem como a quantidade de recursos reservados do *cluster*. Pode-se observar que a utilização deste gráfico por parte do usuário evita que haja conflitos de reservas, uma vez que é possível visualizar os períodos de saturação de uso de recursos.

No entanto, quanto à disponibilidade dos recursos no momento da

execução, o InteGrade com a arquitetura proposta não se mostrou tão eficiente. Por fazer uso de máquinas pessoais em constante uso, não é possível garantir que as máquinas estarão sempre conectadas ao *cluster*, o que pode prejudicar a execução da aplicação e resultar em uma falha da reserva.

6.1 TRABALHOS FUTUROS

Como trabalhos futuros para a melhoria do modelo proposto pode-se citar:

- Utilizar o mecanismo de reserva antecipada com o ambiente oportunista federalizado. Em outras palavras, possibilitar que o meta-escalonador procure por recursos em outros ambientes oportunistas caso os recursos reservados não estejam disponíveis no momento da execução. Com a disponibilização de vários ambientes poderia-se oferecer qualidade de serviço para o usuário, uma vez que pode-se garantir a disponibilidade da quantidade de recurso reservado no momento da execução;
- Pesquisar um método que auxilie o usuário na escolha do *end time*. Esse método deve ser capaz de estimar o tempo de término da execução. Com isso, resolveria o problema de ter *end time* muito longos ou muito curtos, o que manteria o nó reservado mesmo depois de terminado a execução ou terminaria o tempo e não completaria a execução, respectivamente;
- Calcular o consumo energético da abordagem proposta. Com isso, poderia-se estimar, por exemplo, se é mais eficiente energeticamente reservar recursos de uma máquina com alto nível de utilização de seus recursos ou de uma máquina mais ociosa.

REFERÊNCIAS BIBLIOGRÁFICAS

ABAWAJY, J. H.; DANDAMUDI, S. P. Parallel job scheduling on multicluster computing system. In: IEEE. *Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on*. [S.l.], 2003. p. 11–18.

ADZIGOGOV, L.; SOLDATOS, J.; POLYMENAKOS, L. EMPEROR: An OGSA Grid meta-scheduler based on dynamic resource predictions. *Journal of Grid computing*, Springer, v. 3, n. 1-2, p. 19–37, 2005.

BOINC. *BOINC*. Acessado em: Maio 2013. <`http://boinc.berkeley.edu/`>.

BRASILEIRO, F. et al. An approach for the co-existence of service and opportunistic grids: The eela-2 case. In: *Latin-American Grid Workshop*. [S.l.: s.n.], 2008.

BUYYA, R. (Ed.). *High Performance Cluster Computing: Architectures and Systems*. Prentice hall. [S.l.: s.n.], 1999.

CAMARGO, R. Y. D. et al. The grid architectural pattern: Leveraging distributed processing capabilities. *Pattern Languages of Program Design*, v. 5, p. 337–356, 2006.

CAN. *Correio Aéreo Nacional*. Acessado em: Maio 2013. <`http://www.fab.mil.br/acessoainformacao/index.php/faq?showall=\&start=1`>.

CASTILLO, C.; ROUSKAS, G.; HARFOUSH, K. Online algorithms for advance resource reservations. *Journal of Parallel and Distributed Computing*, Elsevier, v. 71, n. 7, p. 963–973, 2011.

CASTILLO, C.; ROUSKAS, G. N.; HARFOUSH, K. Efficient resource management using advance reservations for heterogeneous grids. In: IEEE. *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. [S.l.], 2008. p. 1–12.

CHOI, S. et al. Characterizing and classifying desktop grid. In: IEEE. *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*. [S.l.], 2007. p. 743–748.

CIRNE, W. et al. Labs of the world, unite!!! *Journal of Grid Computing*, Springer, v. 4, n. 3, p. 225–246, 2006.

CONDE, D. M. R. *Análise de padrões de uso em grades computacionais*. Dissertação (Mestrado) — Universidade de São Paulo, Janeiro 2008.

CONTI, M. et al. From opportunistic networks to opportunistic computing. *Communications Magazine, IEEE*, IEEE, v. 48, n. 9, p. 126–139, 2010.

CONTI, M.; KUMAR, M. Opportunities in opportunistic computing. *Computer*, Institute of Electrical and Electronics Engineers, Inc., 3 Park Avenue, 17 th Fl New York NY 10016-5997 USA, v. 43, n. 1, p. 42–50, 2010.

CORBA. *CORBA*. Acessado em: Julho 2013. <`http://www.corba.org/`>.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Sistemas Distribuídos: Conceitos e Projeto*. 4ª. ed. [S.l.]: Bookman, 2007.

DANTAS, M. *Computação distribuída de alto desempenho: redes, clusters e grids computacionais*. [S.l.]: Axcel Books, 2005.

FERREIRA, D.; DANTAS, M.; BAUER, M. An Ontological-Fuzzy Approach to Advance Reservation in Multi-Cluster Grids. In: IOP PUBLISHING. *Journal of Physics: Conference Series*. [S.l.], 2010. v. 256, n. 1.

FERREIRA, D. J. *Reserva Dinâmica e Antecipada de Recursos para Configurações Multi-Clusters Utilizando Ontologias e Lógica Difusa*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, Agosto 2010.

FLYNN, M. J. Some computer organizations and their effectiveness. *Computers, IEEE Transactions on*, IEEE, v. 100, n. 9, p. 948–960, 1972.

FOSTER, I.; KESSELMAN, C. *The Grid: Blueprint for a New Computing Infrastructure*. [S.l.]: Morgan Kaufmann, 2004.

FOSTER, I.; KESSELMAN, C. The History of the grid. *High Performance Computing: From Grids and Clouds to Exascale, Advances in Parallel Computing Series.*, Vol. 20, p. 3–30, 2011. IOS Press.

FOSTER, I. et al. A distributed resource management architecture that supports advance reservations and co-allocation. In: IEEE. *Quality of Service, 1999. IWQoS'99. 1999 Seventh International Workshop on*. [S.l.], 1999. p. 27–36.

FOSTER, I. et al. Cloud computing and grid computing 360-degree compared. In: IEEE. *Grid Computing Environments Workshop, 2008. GCE'08*. [S.l.], 2008. p. 1–10.

GARFINKEL, S. L. *Architects of the Information Society: Thirty-Five Years of the Laboratory for Computer Science at MIT*. [S.l.]: The MIT Press, 1999.

GOLDCHLEGER, A. *Integrade: Um sistema de middleware para computação em grade oportunista*. Dissertação (Mestrado) — Universidade de Sao Paulo, 2004.

GOLDCHLEGER, A. et al. Integrade: object-oriented grid middleware leveraging the idle computing power of desktop machines. *Concurrency and Computation: Practice and Experience*, Wiley Online Library, v. 16, n. 5, p. 449–459, 2004.

GOMES, E.; DANTAS, M. An Enhancement of a Scheduling Approach for an Opportunistic Environment. In: *8TH International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 2013. 3PGCIC*. [S.l.: s.n.], 2013.

GOMES, E.; DANTAS, M. Towards a Resource Reservation Approach for an Opportunistic Computing Environment. In: IOP PUBLISHING. *Journal of Physics: Conference Series, JPCS*. [S.l.], 2013.

HTCONDOR. *HTCondor Version 7.8.7 Manual*. Acessado em: Março 2013. <`http://research.cs.wisc.edu/htcondor/`>.

INTEGRADE. *InteGrade*. Acessado em: Junho 2012. <`http://www.integrade.org.br`>.

JAVADI, B.; AKBARI, M. K.; ABAWAJY, J. H. A performance model for analysis of heterogeneous multi-cluster systems. *Parallel computing*, Elsevier, v. 32, n. 11, p. 831–851, 2006.

KON, F.; GOLDMAN, A. Grades Computacionais: Conceitos Fundamentais e Casos Concretos. *Proceedings of the Tomasz Kowaltowski e Karin Breitman (Org.) Atualizações em Informática*, p. 55–104, 2008.

KRAUTER, K.; BUYYA, R.; MAHESWARAN, M. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, Wiley Online Library, v. 32, n. 2, p. 135–164, 2002.

LAPESD. *Laboratório de Pesquisa em Sistemas Distribuídos*. Acessado em: Setembro 2013. <`http://www.lapesd.inf.ufsc.br/`>.

LUA. *Lua*. Acessado em: Julho 2013. <http://www.lua.org/>.

MACLAREN, J. Advance reservations: State of the Art. In: *Global Grid Forum*. [S.l.: s.n.], 2003. v. 9.

MACLAREN, J.; KEOWN, M. M.; PICKLES, S. Co-allocation, fault tolerance and grid computing. In: *Proceedings of the UK e-Science All Hands Meeting*. [S.l.: s.n.], 2006. p. 155–162.

MARTINS, M. M. et al. Execution Management of Applications with Runtime Restrictions on Opportunistic Grids Environments. In: *PESARO 2012, The Second International Conference on Performance, Safety and Robustness in Complex Systems and Applications*. [S.l.: s.n.], 2012. p. 1–7.

MATEESCU, G. Quality of service on the grid via metascheduling with resource co-scheduling and co-reservation. *International Journal of High Performance Computing Applications*, SAGE Publications, v. 17, n. 3, p. 209–218, 2003.

MEFFE, C.; MUSSI, E.; MELLO, L. Guia de Estruturação e Administração do Ambiente de Cluster e Grid. *Brasilia*, 2006.

MELL, P.; GRANCE, T. The NIST definition of cloud computing (draft). *NIST special publication*, v. 800, p. 145, 2011.

NGS. *National Grid Service*. Acessado em: Agosto 2013. <http://www.ngs.ac.uk/>.

OIL. *ORB in Lua*. Acessado em: Julho 2013. <http://www.tecgraf.puc-rio.br/~maia/oil/>.

OURGRID. *OurGrid*. Acessado em: Março 2013. <http://www.ourgrid.org/index.php>.

PONCIANO, L.; BRASILEIRO, F. Assessing green strategies in peer-to-peer opportunistic grids. *Journal of Grid Computing*, Springer, p. 1–20, 2012.

QIN, J.; BAUER, M. A. Job co-allocation strategies for multiple high performance computing clusters. *Cluster Computing*, Springer, v. 12, n. 3, p. 323–340, 2009.

QIN, J.; DANTAS, M.; BAUER, M. Application Programmers Interactions Enhancing a Meta-Scheduler in Multi-Clusters Environments. Artigo submetido. 2013.

RIMAL, B.; CHOI, E.; LUMB, I. A taxonomy and survey of cloud computing systems. In: IEEE. *INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on*. [S.l.], 2009. p. 44–51.

ROBLITZ, T.; REINEFELD, A. Co-reservation with the concept of virtual resources. In: IEEE. *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*. [S.l.], 2005. v. 1, p. 398–406.

SETI@HOME. *SETI@home*. Acessado em: Agosto 2013. <http://setiathome.berkeley.edu/>.

SHIRAZI, B. A.; KAVI, K. M.; HURSON, A. R. *Scheduling and load balancing in parallel and distributed systems*. [S.l.]: IEEE Computer Society Press, 1995.

SHIVARATRI, N. G.; KRUEGER, P.; SINGHAL, M. Load distributing for locally distributed systems. *Computer*, IEEE, v. 25, n. 12, p. 33–44, 1992.

SILVA, F. J. da Silva e et al. Application execution management on the InteGrade opportunistic grid middleware. *Journal of Parallel and Distributed Computing*, Elsevier, v. 70, n. 5, p. 573–583, 2010.

SINAGA, J. M. P. *A Dynamic0 Co-allocation Service in Multi-Cluster Systems*. Dissertação (Mestrado) — Delf University of Technology, April 2004.

SINAPAD. *Sistema National de Processamento de Alto Desempenho*. Acessado em: Agosto 2013. <`https://www.lncc.br/sinapad/index.php`>.

SODAN, A. C. Loosely coordinated coscheduling in the context of other approaches for dynamic job scheduling: a survey. *Concurrency and Computation: Practice and Experience*, Wiley Online Library, v. 17, n. 15, p. 1725–1781, 2005.

SULISTIO, A.; BUYYA, R. A grid simulation infrastructure supporting advance reservation. In: *16th International Conference on Parallel and Distributed Computing and Systems (PDCS 2004)*. [S.l.: s.n.], 2004. p. 9–11.

SULISTIO, A. et al. GarQ: An efficient scheduling data structure for advance reservations of grid resources. *International Journal of Parallel, Emergent and Distributed Systems*, Taylor & Francis, v. 24, n. 1, p. 1–19, 2009.

TANENBAUM, A. S.; STEEN, M. V. *Distributed systems*. [S.l.]: Prentice Hall, 2002.

TERAGRID. *Advance Reservation and Co-Scheduling Report*. Acessado em: Maio 2013. <http://www.teragridforum.org/mediawiki/images/c/cd/Schedwg_RsrvCoschedReport.pdf>.

TOMÁS, L. et al. Network-aware meta-scheduling in advance with autonomous self-tuning system. *Future Generation Computer Systems*, Elsevier, v. 27, n. 5, p. 486–497, 2011.

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.

XIAOHUI, W. et al. CSF4: A WSRF compliant meta-scheduler. In: CITESEER. *The 2006 World Congress in Computer Science, Computer Engineering, and Applied Computing, GCA*. [S.l.], 2006. v. 6, p. 61–67.

XTREMWEB. *XtremWeb: the Open Source Platform for Desktop Grids*. Acessado em: Abril 2013. <http://www.xtremweb.net/>.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, Springer, v. 1, n. 1, p. 7–18, 2010.

APÊNDICE A – Detalhes da Implementação

Neste apêndice será detalhado a *Interface Definition Language* (IDL) desenvolvida para o modelo.

A.1 INTERFACE DEFINITION LANGUAGE - IDL

A interface IDL do módulo ARR é apresentada na Lista A.1. Abaixo serão detalhadas algumas variáveis descritas na lista.

- reserve: contém as informações necessárias para realizar a reserva (*start time, end time* e *e-mail*);
- execInfo: contém informações necessárias para a reserva da máquina;
- oFile: arquivos de saída;
- acInfo: dados para informar o usuário quanto ao aceite da reserva;
- memTot: quantidade total de memória;
- asctIor: identificação do ASCT solicitante.

Listing A.1 – Interface IDL do Advanced Reservation of Resource

```
interface AdvancedReservation {

  void alterReservation(in types::RInfo reserve);

  void remove(in long id);

  long reservationData(in types::ARInfo reserve,
                       in string lrmIor,
                       in types::ExecInfo execInfo);

  boolean checks(in types::ARInfo reserve);

  void executionFinished(in string email,
                         in types::OutFile oFile);

  void executionAccepted(in types::Info acInfo,
                         in string email);

  void executionSuggestion(in string predictionCPU,
                           in string predictionMem,
                           in unsigned long memTot);

  void reservationSuggestion(in string asctIor);

 };
```

APÊNDICE B – Trabalhos Publicados

Este apêndice apresenta os trabalhos publicados ao longo da pesquisa deste trabalho de dissertação.

B.1 PUBLICAÇÕES

B.1.1 HPCS 2013 - High Performance Computing Symposium

- **Título:** Towards a Resource Reservation Approach for an Opportunistic Computing Environment
- **Evento:** High Performance Computing Symposium
- **Local:** Ottawa, Canadá
- **Data:** 2-6 de Junho
- **Autores:** Eliza Gomes e M.A.R. Dantas
- **Premiação:** Best Student Paper
- **Publicação:** Journal of Physics: Conference Series (JPCS)
- **Estrato Qualis/CAPES:** B2

B.1.2 3PGCIC 2013 - International Conference on P2P, Parallel, Grid, Cloud and Internet Computing

- **Título:** An Enhancement of a Scheduling Approach for an Opportunistic Environment
- **Evento:** 8TH International Conference on P2P, Parallel, Grid, Cloud and Internet Computing
- **Local:** Compiegne, França
- **Data:** 28-30 Outubro
- **Autores:** Eliza Gomes e M.A.R. Dantas
- **Estrato Qualis/CAPES:** B4