UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Denise Janson Ferreira

Reserva Dinâmica e Antecipada de Recursos para Configurações Multi-Clusters Utilizando Ontologias e Lógica Difusa

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Prof. Mário Antônio Ribeiro Dantas, Dr.
(Orientador)

Florianópolis, Agosto de 2010

Reserva Dinâmica e Antecipada de Recursos para Configurações Multi-Clusters Utilizando Ontologias e Lógica Difusa

Denise Janson Ferreira

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, área de concentração Sistemas Distribuídos e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Mário Antônio Ribeiro Dantas, Dr. (Coordenador)

Banca Examinadora

Prof. Mário Antônio Ribeiro Dantas, Dr.
(Orientador)

Prof. Nelson F. F. Ebecken, Dr. (UFRJ)

Prof. Carlos Montez, Dr. (UFSC)

Prof. João Bosco M. Sobral, Dr. (UFSC)

"O que somos é consequência do que pensamos."

Siddharta Gautama

Para meus pais, Américo e Elisabeth, meus avós, Herall e Mathilde,
e minhas irmãs, Lívia e Cristiane.

Agradecimentos

Agradeço ao meu orientador, prof. Dr. Mário Dantas, pela dedicação nesses anos de trabalho, amizade e oportunidades à mim confiadas. Ao prof. Dr. Michael A. Bauer, pela oportunidade de trabalho conjunto na UWO (*University of Western Ontario*) e na Sharcnet. Agradeço ao prof. Alexandre P. C. da Silva e à Dr. Jinhui Qin pela explicação e disponibilidade dos seus trabalhos respectivamente de mestrado e doutorado, para desenvolvimento conjunto; obrigada pelas sugestões e pela amizade.

Aos membros do projeto para desenvolvimento de Middleware, pela oportunidade de produção em equipe, que me ajudaram a entender o funcionamento desses gerenciadores em ambiente real. À Juliana Eyng, pelo trabalho conjunto em seu doutorado, que me ajudou a entender as necessidades das aplicações na prática. À Sharcnet pelo ambiente de testes.

Aos membros do LaPeSD que me acompanharam, aos funcionários da secretaria, em especial Vera Sodré, pelo carinho e atenção esses anos, e Katiana Silva, agora no final. À CAPES, pelo suporte financeiro. Aos membros da banca, prof. Dr. Nelson F. F. Ebecken, prof. Dr. Carlos Montez e prof. Dr. João B. M. Sobral, pela disponibilidade e contribuição com este trabalho. Aos professores que ajudaram direta ou indiretamente.

Gostaria de agradecer imensamente à minha família por todo apoio, paciência e principalmente amor durante esse período conturbado, vocês são minha inspiração, minha motivação, minha base. Obrigada por estarem sempre presentes em todos os momentos. Agradeço também à Dani, amiga como uma irmã, que me incentivou nos momentos difíceis e compartilhou meus êxitos e alegrias do início ao fim deste trabalho!

Agradecimento especial também ao prof. Valdir e à Cris, pelos momentos alegres, pelos cafés (he! he!), pela força nos meus desânimos. Ao Jeferson, pela motivação, companhia, principalmente pelo exemplo e inspiração. Ao Matheus, pela colaboração nos trabalhos conjuntos e por mostrar apoio nos momentos difíceis. À Márcia, Nilsa, Isabel e à todos os amigos que compartilharam esse período comigo.

À Deus, por permitir mais esta realização na minha vida, que me proporcionou aprendizados, crescimento profissional e vínculos amigos!

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

A reserva antecipada de recursos é um mecanismo importante para garantir maior aproveitamento na utilização dos recursos disponíveis em ambientes multi-clusters distribuídos. Este mecanismo permite, por exemplo, que um usuário forneça parâmetros com o objetivo de satisfazer determinados requisitos no momento da execução de uma aplicação. Esta previsibilidade permite que o sistema alcance maiores níveis de QoS. Entretanto, a complexidade das configurações de larga escala e as mudanças dinâmicas verificadas nesses sistemas limitam o suporte à reserva antecipada. O presente trabalho de pesquisa caracteriza-se pela proposta de uma abordagem de reserva antecipada utilizando os paradigmas de ontologias e lógica difusa. Esta proposta permite que uma aplicação reserve mais de um cluster por tarefa, e, também, que requisite uma grande diversidade de recursos. Em adição, a disponibilidade local dos recursos é verificada localmente de forma dinâmica, evitando conflitos futuros no momento da alocação. Foram realizadas comparações relativas a outros trabalhos e os resultados experimentais, utilizando simulações de configurações de multi-clusters, indicam que o mecanismo proposto alcançou com sucesso flexibilidade para tarefas que requisitaram alto processamento, permitindo um balanceamento adequado de processos entre clusters.

Palavras chave: Reserva antecipada de recursos, Alocação de Recursos, Grades.

Abstract

Advance reservation is an important mechanism for a successful utilization of available resources in distributed multi-cluster environments. This mechanism allows, for example, a user to provide parameters aiming to satisfy requirements related to applications' execution time and temporal dependence. This predictability can lead the system to reach higher levels of QoS. However, the support for advance reservation has been restricted due to the complexity of large scale configurations and also dynamic changes verified in these systems. In this research work it is proposed an advance reservation method, based on a fuzzy-ontology approach. It allows a user to reserve a wide variety of resources and enable large jobs to be reserved among different nodes. In addition, it dynamically verifies the possibility of reservation with the local RMS, avoiding future allocation conflicts. Comparisons were made with other works and the experimental results of the proposal, through simulation, indicate that the proposed mechanism reached a successful level of flexibility for large jobs and more appropriated distribution of resources in a distributed multi-cluster configuration.

Keywords: Advance reservation, Resources allocation, Grids.

Capítulo 1

Introdução

1.1 Motivação

Configurações multi-clusters, quando bem orquestradas como um ambiente de grade, podem representar um poder computacional significante para execução de diferentes classes de aplicações, inclusive tarefas paralelas [Pugliese, Talia e Yahyapour 2008]. Um grande desafio neste contexto é atingir um nível de alocação de recursos eficiente tanto para aplicações locais quanto para remotas. Em uma configuração de cluster, o sistema de gerenciamento de recursos (SGR) local oferece, naturalmente, maior prioridade às aplicações locais.

Grande parte dos escalonadores para os grades mantém os jobs em uma fila, quando não há recursos disponíveis no ambiente no momento da submissão. Ou seja, não há cálculo exato sobre quando esses jobs poderão executar. Entretanto, este comportamento pode causar problemas em algumas aplicações que necessitem de garantias de execução e qualidade de serviço, tal como aplicações de colaboração em tempo real (video conferências), video sob demanda [Sulistio e Buyya 2004]. O uso de reservas antecipadas ajuda a evitar que os jobs submetidos esperem por recursos, visto que os pedidos são realizados antecipadamente. Desta forma, aumentam consideravelmente as probabilidades de admissão do job no momento de execução [Burchard 2005].

Reservas antecipadas são especialmente úteis em ambientes de grades, os quais geralmente possuem quantidade razoável de usuários competindo por recursos. Permite acesso concorrente para que as aplicações executem em paralelo e, ao mesmo tempo, garantindo a disponibilidade dos recursos em um determinado momento no futuro [Wu et al. 2005]. Mas há também uma variedade de aplicações que requerem qualidade de serviço na rede, tal como redes de fornecimento de conteúdo e até mesmo clientes móveis, que podem precisar de reserva antecipada para, por exemplo, transferência de vídeo [Liu, Dai e Chuang 2006].

Alguns SGRs suportam reserva antecipada (RA), como o PBS Pro-

fessional [PBS], o LSF [LSF Product Suite] e o MAUI/MOAB [Manager/-Moab Workload Manager]. Entretanto, geralmente possuem formatos privados de interface para execução de reservas. SGRs que possuem componente de RA embutido são comumente chamados de meta-escalonadores. Meta-escalonadores se caracterizam por receber requisições de usuários e escalonar suas tarefas para clusters locais do sistema, podendo alterar tanto filas remotas quanto locais. Essa funcionalidade de RA em meta-escalonadores requer também um modelo eficiente de co-reserva e co-alocação de recursos. Vários projetos de meta-escalonadores para grades computacionais, baseadas em ambientes de multi-clusters, já foram propostos. Exemplos de tais meta-escalonadores são o Calana [Dalheimer, Pfreundt e Merz 2005], Deco [Agarwal et al. 2006], Gridway [GridWay Metascheduler], GridARS [Takefusa et al. 2008] e Viola [Eickermann et al. 2007].

Uma grade computacional, composta por configurações multi-cluster, pode existir em um domínio privado ou em múltiplos domínios, envolvendo organizações distintas. Cada organização virtual (OV) normalmente possui políticas próprias, em termos de software e hardware. A descrição dos recursos pode diferir de uma OV para outra, e isto faz com que o processo de compartilhamento se torne complexo. Desta forma, é necessário combinar recursos com características similares, mas descrições diferentes.

O paradigma de ontologias tem sido considerado como solução para descrição formal de recursos em grades (como ilustrado em [Ejarque et al. 2008] e [Silva e Dantas 2007]). Em adição, serve como base para a combinação semântica.

O trabalho apresentado em [Silva e Dantas 2007] oferece uma ferramenta de integração semântica para múltiplas ontologias de diferentes OVs. Este permite que um usuário comum selecione os recursos necessários no ambiente para execução de uma aplicação, tal como o sistema operacional, a arquitetura do processador e a quantidade de memória. A ferramenta retorna as OVs que podem atender a estes requisitos, avaliando informações estáticas sobre o sistema.

No trabalho desenvolvido em [Janson et al. 2009], foi realizada seleção de recursos em grades considerando a dinamicidade do sistema na seleção dos recursos. Além disso, permite que os processos de uma tarefa sejam distribuídos e balanceados entre clusters, baseado em um controle adaptativo para co-alocação de tarefas, utilizando lógica difusa.

Esta dissertação apresenta o GRADI (**G***erenciador para* **R***eserva,* **A***locação,* **D***escrição e* **I***ntegração de recursos em grades*), uma proposta diferenciada para reserva antecipada de recursos, baseada nas pesquisas de

seleção e alocação apresentadas em [Janson et al. 2009], [Silva e Dantas 2007] e [Qin e Bauer 2007]. Mecanismos de co-reserva e co-alocação são importantes para obtenção de um melhor uso e aproveitamento dos recursos em grades. Entretanto, podem surgir conflitos quando a dinamicidade do sistema não é considerada (exemplos são ilustrados em [Foster et al. 1999] e [Czajkowski, Foster e Kesselman 1999]). Com resultado, uma reserva antecipada pode garantir alguns parâmetros de QoS às aplicações.

1.2 Objetivos

1.2.1 Objetivos Gerais

Este trabalho de mestrado tem por objetivo geral propor uma abordagem diferenciada de reservas antecipadas no gerenciamento de recursos distribuídos em grades, para maximizar o nível de utilização desses recursos. Para alcançar este objetivo, foi desenvolvida uma arquitetura de middleware para grades computacionais, o GRADI, com a implementação de um protótipo eficaz para co-reserva, e com experimentos em ambiente simulado, comparando com outros trabalhos de pesquisa na área.

1.2.2 Objetivos Específicos

Dentre os principais objetivos específicos do middleware, estão:

- Oferecer e dar suporte a heterogeneidade de recursos na grade;
- Permitir integração de diferentes organizações virtuais, com descrições de recursos diferenciadas;
- Oferecer algumas garantias de execução e disponibilidade dos recursos no momento da execução;
- Permitir alguns parâmetros de QoS nos pedidos dos clientes e sobre garantias de disponibilidade dos recursos;
- Alcançar alto nível de utilização de recursos, por meio do método de reserva utilizado.
- Permitir execução de jobs com necessidade de alto processamento, entre clusters.

1.3 Estrutura

Esta dissertação está subdividida em 6 capítulos, os quais serão brevemente descritos a seguir.

No capítulo 2 é realizado um estudo geral sobre sistemas distribuídos de alto desempenho, mostrando os diferentes tipos de arquitetura. É

dado um enfoque maior em clusters e grades, sendo apresentados alguns middlewares para esses tipos de ambientes.

Conceitos fundamentais sobre meta-escalonamento para o entendimento deste trabalho são tratados no capítulo 3. São definidas e explicadas algumas definições na literatura a respeito de alocação, reserva e escalonamento de recursos, destacando trabalhos correlatos a área.

A proposta do trabalho desta dissertação é apresentada no capítulo 4, assim como a arquitetura desenvolvida para reserva de recursos. Além disso, são mostrados também detalhes da implementação e abordagem adotada.

No capítulo 5 são analisados experimentos no middleware, focando na utilização dos recursos com os resultados obtidos. Também é realizada uma comparação desta ferramenta com outros trabalhos na literatura.

Por fim, no capítulo 6 é feito um resumo do trabalho, apresentando algumas considerações, dificuldades e conclusões obtidas com os resultados. São destacadas as contribuições e mostrados futuros encaminhamentos para este estudo.

Capítulo 2

Sistemas Distribuídos de Alto Desempenho

Nas últimas décadas, nota-se um significativo avanço tecnológico em micro-processamento e redes de alta velocidade. De acordo com [Yeo et al. 2006], essas técnicas, somadas ao desenvolvimento de ferramentas padrões para computação distribuída de alto desempenho, têm favorecido o uso de sistemas distribuídos em aplicações que exigem grande quantidade processamento e armazenamento.

Cálculos complexos e repetitivos, com pequenas modificações apenas na configuração de parâmetros, compõem aplicações de diferentes áreas de conhecimento, tal como meteorologia, astronomia e genética. Essas aplicações requerem um aumento crescente de poder computacional, armazenamento, e, consequentemente, exigem infraestrutura de produção apropriada. Neste contexto, nota-se na literatura que os benefícios dos ambientes distribuídos de alto desempenho têm sido aproveitados nessas áreas (e.g. [Wang et al. 2007], [Mirin e Worley 2008] e [Andrade et al. 2007]).

No conceito de [Coulouris, Dollimore e Kindberg 2005], a computação distribuída tem como objetivo integrar computadores de arquiteturas heterogêneas em redes com largura de banda considerável, somando o poder computacional destas máquinas de forma que trabalhem colaborativamente para realizar as tarefas. Ou seja, os computadores do sistema podem ser independentes entre si, mas interligados por meio de software que permita o compartilhamento de recursos e da execução de tarefas. Além disso, essa colaboração deve ser transparente e oferecer suporte a diferentes modelos de aplicações. É importante salientar que considera-se recurso como qualquer entidade do sistema, tal como software, hardware ou dados.

Para alcançar os objetivos de desempenho, pode-se adotar as seguintes técnicas, como mostrado em [Buyya 2003]:

1. Utilizar hardwares mais eficientes e com maior capacidade;

2. Otimizar os algoritmos usados nas soluções das tarefas;

3. Fazer uso de múltiplos computadores para solucionar a tarefa em particular.

Entretanto, a primeira alternativa muitas vezes não é viável financeiramente. Além disso, nem sempre o avanço tecnológico acompanha as necessidades das aplicações. Desta forma, o uso de agregados de computadores, juntamente com a otimização do código utilizando técnicas paralelas distribuídas é uma solução de menor custo, e que pode representar um ganho significativo no desempenho.

Considerando a variedade de características dos sistemas paralelos distribuídos, são utilizados na literatura diferentes enfoques para classificação da arquitetura desses ambientes, como mostrado a seguir.

2.1 Classificação de Arquiteturas Paralelas

A Taxonomia de Flynn, que surgiu a mais de 30 anos é a mais conhecida para classificação de ambientes de hardware. Ainda é muito utilizada hoje em dia, apesar de ter um pouco de dificuldade em abranger todas as arquiteturas atuais. Esta abordagem foca no número de instruções executadas em paralelo versus o conjunto de dados para os quais as instruções submetidas. Está subdividida em SISD, SIMD, MISD e MIMD, detalhados abaixo de acordo com [Dantas 2005]:

- SISD (*Single Instruction Single Data*): realiza uma única instrução por vez, ou seja, utilizando um único processador. O SISD representa a grande parte dos computadores convencionais com instruções sendo executadas sequencialmente. Possui um único fluxo de instruções para um único fluxo de dados.

- SIMD (*Single Instruction Multiple Data*): realiza uma única instrução por vez, mas sobre múltiplos dados. Isso ocorre devido a ocorrência de facilidades de hardware para armazenamento, podendo ser um vetor, ou um *array*. Os processadores recebem uma mesma instrução de uma unidade de controle, mas operam sobre diferentes conjuntos de dados. Dessa forma, a mesma instrução e processada sob diferentes itens de dados. Ocorre um fluxo único de instruções com um fluxo múltiplo de dados.

- MISD (*Multiple Instruction Single Data*): seria o fluxo de múltiplas instruções sob o fluxo de um único dado. Neste caso várias unidades de processamento recebem diferentes instruções sob um mesmo conjunto de dados e derivados. Há controvérsias sobre o MISD, muitos autores não o consideram, por não haver conhecimento de arquitetura de máquinas deste tipo.

- MIMD (*Multiple Instruction Multiple Data*): fluxo de múltiplas instruções com o fluxo de múltiplos dados. Nesta classificação, a arquitetura possui vários processadores e estes podem executar instruções independentes entre si sob diferentes fluxos de dados. Esta estrutura e própria para o desenvolvimento de algoritmos paralelos. Nesta categoria estão grande parte dos sistemas multi-computadores.

Para dar suporte à comunicação entre processadores e garantir alto desempenho nestas configurações é necessário o uso de redes de interconexão, como descrito na próxima subseção.

2.1.1 Redes de Interconexão

Sistemas distribuídos de alta desempenho devem incorporar tecnologias de interconexão eficientes para garantir largura de banda suficiente, altas taxas de transferência e latência baixa na comunicação entre os processadores. Caso não seja utilizada uma tecnologia adequada, esta comunicação pode representar um gargalo no desempenho do sistema como um todo.

A escolha sobre qual tecnologia de interconexão utilizar no sistema depende de diversos fatores, dentre estes: compatibilidade com o hardware e os sistemas operacionais; custo; desempenho da rede com relação a largura de banda e latência. Por outro lado, o tipo de aplicação também deve ser analisada. A quantidade de processamento comparada à quantidade de comunicação deve ser observada. Isto porque a granularidade da aplicação apresenta apresenta diferentes desempenhos nos diferentes tipos de rede e influencia diretamente em qual tipo de configuração utilizar na rede, como demonstrado em [Pinto, Mendonça e Dantas 2008].

Encontram-se na literatura diferentes tecnologias para auxiliar na construção eficiente de sistemas de alto desempenho mais eficientes. Dentre estas tecnologias estão: Gigabit Ethernet [Seifert 1998], Giganet [Vogels et al. 2000], SCI [Pinto, Mendonça e Dantas 2008], Myrinet [Boden et al. 1995], Infinitband [Shanley 2002], Quadrics [Petrini et al. 2003], e outros.

No trabalho de [Yeo et al. 2006] é feita comparação entre algumas destas tecnologias, na qual são observados fatores como largura de banda, latência, compatibilidade de hardware, sistema operacional, implementação do protocolo. Neste trabalho é mostrada uma comparação entre as principais tecnologias, na tabela 2.1.

A rede de interconexão é um fator importante na configuração de sistemas distribuídos paralelos tal como clusters. Nas próximas seções serão detalhadas configurações para sistemas distribuídos paralelos de alto desempenho: clusters, multi-clusters e grades computacionais.

Tabela 2.1: Tecnologias de rede de interconexão, de [Yeo et al. 2006]

Comparison Criteria	Gigabit Ethernet	Giganet cLAN	Infiniband
Bandwidth (MBytes/s)	< 100	< 125	850
Latency (µs)	< 100	7-10	< 7
Hardware Availability	Now	Now	Now
Linux Support	Now	Now	Now
Max. No. of Nodes	1000s	1000s	> 1000s
Protocol Implementation	Hardware	Firmware on adaptor	Hardware
Virtual Interface Architecture (VIA) Support	NT/Linux	NT/Linux	Software
Message Passing Interface (MPI) Support	MVICH [12] over M-VIA [13], TCP	3rd Party	MPI/Pro [14]

Comparison Criteria	Myrinet	QsNet II	Scalable Coherent Interface (SCI)
Bandwidth (MBytes/s)	230	1064	< 320
Latency (µs)	10	< 3	1–2
Hardware Availability	Now	Now	Now
Linux Support	Now	Now	Now
Max. No. of Nodes	1000s	4096	1000s
Protocol Implementation	Firmware on adaptor	Firmware on adaptor	Firmware on adaptor
Virtual Interface Architecture (VIA) Support	Linux	None	Software
Message Passing Interface (MPI) Support	3rd Party	Quadrics	3rd Party

2.2 Clusters

Os clusters, também chamados de agregados computacionais, são definidos no trabalho de [Buyya 2003] como configurações de sistema distribuído paralelo formadas por um conjunto de computadores convencionais, independentes, mas interconectados por redes - muitas vezes de alto desempenho - e trabalhando juntos, como um único recurso integrado, para solução de problemas computacionais. Observando que um nó do cluster pode ser composto por uma máquina com um único processador, ou por um sistema multi-processado. Desta forma, é possível alcançar processamento semelhantes aos super-computadores, contudo, por um custo mais baixo.

Segundo [Colvero, Dantas e Cunha 2005], as principais características de uma configuração de cluster, as quais a diferenciam de outras configurações são:

- Alocação de recursos dentro de um único domínio, o qual é centra-

lizado;

- A segurança de processamento e recursos é dispensável, visto se tratar de um único domínio;
- Permitem milhares de nós na rede (quantidade limitada se for comparada às outras configurações detalhadas nas próximas seções);
- Objetiva solução de problemas de granularidade grossa;
- Homogeneidade com relação ao sistema operacional.

Alguns dos principais objetivos desses sistemas são:

- Realizar tarefas complexas em paralelo pela rede, oferecendo alto desempenho;
- Diminuir custos por meio do uso de componentes de fácil disponibilidade e heterogêneos, ou seja, independentes de fornecedor;
- Permitir diferentes configurações, adaptando a disponibilidade do sistema as requisições das aplicações e ao orçamento do cliente.
- Oferecer escalabilidade, ou seja, facilidade em adicionar novos recursos ao sistema;
- Obter alta disponibilidade de recursos para a execução das tarefas por meio de tolerância a falhas. A independência das máquinas escravas impedem que falhas afetem a o sistema como um todo.
- Oferecer transparência das complexidades do gerenciamento destas máquinas para o usuário final, formando um sistema de imagem única (SSI). Ou seja, permitir o manuseio dos recursos de forma global e transparente, criando a ilusão de estar sendo usada uma única máquina, em vez da rede.

O cluster pode estar configurado de forma dedicada ou não dedicada, como observado por [Dantas 2005]:

Configurações não dedicadas: os computadores são conectados a rede sem que haja uma interoperabilidade dos recursos. Para que os computadores interajam é necessário programação.

Configurações dedicadas: utilizam de pacotes de software centrais que são responsáveis pelo controle lógico do ambiente agregado. Desta forma o sistema sabe previamente parâmetros da rede como capacidade de armazenamento dos discos, tipo de processadores, etc. O

uso destes pacotes permite melhor escolha na distribuição das aplicações e tarefas.

Como visto nesta seção, clusters referem-se a ambientes em um único domínio. Por outro lado, quando há necessidade de agregar clusters de domínios diferentes, usa-se o conceito de multi-clusters.

2.2.1 Multi-Clusters e Gerenciamento de Recursos

Uma configuração multi-cluster permite o compartilhamento de recursos em diferentes domínios, conectando múltiplos ambientes de cluster geograficamente dispersos. Esta configuração forma um sistema integrado de imagem única, ou seja, o usuário tem uma visão global dos recursos e os utiliza como se estivessem em uma única máquina, independente de onde estão fisicamente associados [Yeo et al. 2006].

Associar clusters de diferentes domínios permite um melhor aproveitamento dos recursos do sistema como um todo. Isto porque máquinas ociosas de um domínio podem ser utilizadas por usuários de diferentes domínios, e não só por usuários do domínio local. Desta forma, usuários que ficariam em fila em seus domínios, aproveitam os recursos de outros domínios e, além disso, usuários que precisam de recursos específicos obtém uma maior quantidade de opções dentro do sistema. Entretanto, a administração destes recursos em um ambiente multi-cluster é limitada a um único tipo de gerenciador, ou seja, o gerenciador de cada cluster local deve ser o mesmo para todos os domínios do sistema.

O sistema de gerenciamento de clusters baseia-se na coordenação das requisições dos usuários, dos recursos compartilhados, e na alocação de forma eficiente dos recursos disponíveis no sistema. Desta forma, os principais componentes deste sistema são: um gerenciador dos recursos compartilhados, e um escalonador de tarefas. Este escalonador precisa de informações sobre as filas do sistema, a atual carga de processamento em cada nó e a disponibilidade dos mesmos para que tome uma decisão eficiente sobre onde alocar as tarefas, de acordo com suas estratégias de escalonamento. Todas essas informações ele consegue por meio do módulo de gerenciamento de recursos compartilhados.

Para a execução de tarefas paralelas nos sistemas locais, é necessário o desenvolvimento de camadas de software responsáveis pela comunicação entre processos, e gerenciamento dos recursos no cluster. Existem middlewares e APIs com objetivo de diminuir a complexidade destes processos. Nas próximas seções serão detalhados os ambientes paralelos e middlewares de gerenciamento de recursos para clusters.

2.2.2 Ambientes Paralelos

O ambiente paralelo representa uma camada intermediária na configuração de clusters e caracteriza-se por pacotes de software e bibliotecas que permitem o desenvolvimento de comunicação entre processos e programação a paralela distribuída. Com estas ferramentas, é possível utilizar diversos processadores da rede e suas memórias locais. Além disso, estes pacotes padronizam a execução de tarefas paralelas, garantindo a portabilidade do código da aplicação.

Algumas das tecnologias utilizadas nesses ambientes são o PVM (*Parallel Virtual Machine*) e o MPI (*Message Passing Interface*). Apesar de serem eficientes e convergirem no objetivo principal de processamento paralelo, há consideráveis diferenças de implementação entre estes pacotes, como pode ser observado no trabalho [Hussain e Ahmed 2006]. Desta forma, as características e funcionalidades diversas que cada um apresenta podem ser diferenciais, dependendo do tipo de aplicação, influenciando no desempenho geral.

Abaixo estão as definições do PVM e do MPI, respectivamente de acordo com os projetos [PVM project] e [MPI Project].

PVM: conjunto integrado de ferramentas de softwares e bibliotecas, os quais emulam um framework concorrente em computadores heterogêneos interconectados. Baseia-se na criação de uma máquina virtual única para um conjunto de computadores ligados na rede e que trabalham cooperativamente, possibilitando a visão do sistema como um único recurso computacional.

MPI: biblioteca de troca de mensagens para programação paralela distribuída. Os processos utilizam de chamadas às bibliotecas para trocar informações com outros processos. não utiliza o conceito de máquina virtual única, pois é baseado em troca de mensagens entre processos em diferentes processadores.

O principal objetivo destes pacotes é permitir a programação paralela no ambiente, entretanto, não cobrem grande parte das tarefas de gerenciamento de recursos. Para esta função existem middlewares específicos, como descrito a seguir.

2.2.3 Middlewares de Gerenciamento de Recursos para Clusters

Como mencionado na seção anterior, o gerenciamento de recursos em clusters envolve uma série de funções que não apenas a paralelização dos processos. Algumas destas tarefas são: descrição e seleção de

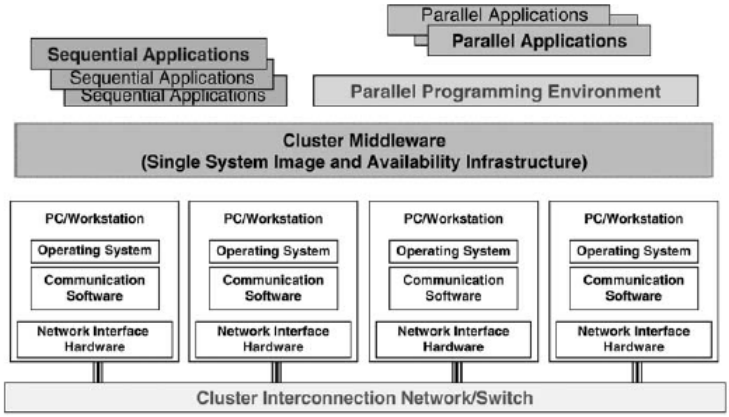

Figura 2.1: Arquitetura típica de um cluster, de [Buyya 2003]

recursos; alocação; escalonamento; coordenação; monitoramento dos recursos e dos processos; tolerância a falhas; segurança; armazenamento distribuído.

Existem alguns middlewares para gerenciamento de recursos em clusters. De acordo com [Dantas 2005], middleware pode ser definido como uma camada de software intermediária entre a aplicação e o ambiente de execução (hardware, sistema operacional), a qual permite a integração trasparente em sistemas distribuídos heterogêneos. O middleware tem a função de abstrair a programação distribuída para que fique o mais próximo possível da centralizada, formando um sistema de imagem única.

Resumindo, dentro do contexto de sistemas distribuídos de alto desempenho, uma configuração típica da arquitetura de um cluster, ou multicluster, pode ser definida como mostrado na figura 2.1, de [Buyya 2003]. Nesta, um switch ou uma rede de interconexão realiza a comunicação de nós, que podem ser formados de máquinas ou clusters. Cada um desses nós tem sua independência local, mas faz parte de um sistema global, gerenciado por um middleware que os integra como um sistema de imagem única. Este middleware recebe aplicações seriais ou paralelas, e as encaminha para as filas mais adequadas. Caso seja uma aplicação paralela, é utilizado um ambiente paralelo para execução.

Existem vários projetos de middlewares para clusters, entre estes: LSF (Load Sharing Facility) [LSF 2010], SGE (Sun Grid Engine) [SGE

2010], Torque (antigo PBS - Portable Batch System) [Torque 2010] e Condor [Condor 2010]. São gerenciadores competitivos no mercado, eficientes com relação às principais funções de um gerenciador de recursos (escalonamento e políticas de alocação em sistemas distribuídos heterogêneos). Entretanto, cada um possui algumas peculiaridades, o que os fazem mais adequados ou não, dependendo do contexto a ser aplicado. A seguir uma breve descrição sobre cada um destes.

2.2.3.1 LSF

O LSF é um gerenciador de recursos comercial que cria um sistema de imagem única em uma rede de computadores heterogêneos, para que todos os recursos possam ser manipulados e utilizados. O sistema base do LSF oferece os principais serviços para compartilhamento de carga em uma rede com diferentes sistemas computacionais, como mostrado na figura 2.2 da arquitetura abaixo:

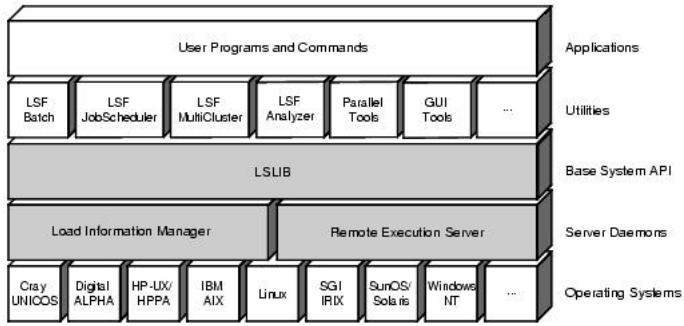

Figura 2.2: Arquitetura do LSF, em [LSF guide 2002]

Esta arquitetura é detalhadamente descrita em [LSF guide 2002], onde é mostrado que o LSF possui uma API para facilitar a interface com o usuário (LSLIB); um módulo para monitorar a carga de cada máquina (LIM); um outro módulo para armazenar essas informações de carga coletadas (LIM mestre), possuindo um sistema de tolerância a falhas para este módulo. Outros componentes são o RES, para execução remota; e o PIM, que monitora cada tarefa ou processo criados localmente no cluster.

O módulo MBD (*Master Batch Daemon*), é responsável por decidir as políticas de escalonamento para as tarefas no ambiente. Para realizar essa decisão o MBD utiliza vários parâmetros, entre estes: requisitos de recursos da tarefa; disponibilidades destes recursos no cluster; limitações

dos recursos para execução da tarefa; dependências da tarefa; restrições de compartilhamento; carga atual do recurso.

Depois decidido o escalonamento, as tarefas enviadas permanecem em fila até serem escalonadas para execução em um host. Os tipos de filas podem ser controladas e configuradas pelo administrador, assim como os tipos e prioridades das tarefas.

O LSF se destaca por possuir muitas características e funções complementares que auxiliam o usuário e que nos outros gerenciadores ainda estão limitadas. Entretanto, cada funcionalidade extra deve ser paga. Apesar disso, alguns usuários preferem pagar para ter essas características a mais. Algumas dessas características relevantes são:

- APIs para programadores java, C e de serviços web que queiram escrever códigos cientes do ambiente de cluster, e que tenham um certo controle.

- Arquitetura em camadas que permite adicionar funções/características extras ao produto. Por exemplo, integração com FlexNet Publisher (gerenciador de licença de softwares); função multi-cluster.

2.2.3.2 SGE

O SGE é um gerenciador de recursos desenvolvido para maximizar a utilização dos recursos em ambientes distribuídos. A figura 2.3 abaixo mostra a arquitetura do SGE, segundo [Beginner's guide to sun tm Grid Engine 2009]. Os principais componentes da arquitetura são: o Qmaster,

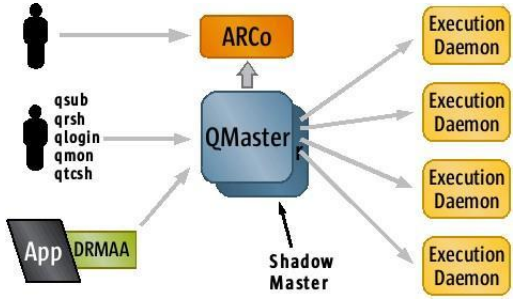

Figura 2.3: Arquitetura do SGE, em [Beginner's guide to sun tm Grid Engine 2009]

o qual é executado em uma única máquina e possui funções de alocação de recursos, escalonamento, administração, monitoramento; Shadow Master,

para oferecer tolerância a falhas; Execution Daemon, para execução utilizando recursos locais. Não há limites na quantidade de jobs que um deamon pode rodar. Mas geralmente esse número é limitado a quantidade de processadores da máquina em execução; o DRMAA, que é uma interface de programação que permite às aplicações submeter, monitorar e controlar as tarefas no cluster. Permite programação em C e Java; Qmon, interface gráfica; e o ARCo, ferramenta web para acessar histórico de informações armazenadas do sistema, por padrão em SQL. A arquitetura é explicada mais detalhadamente em [Beginner's guide to sun tm Grid Engine 2009].

Para execução do escalonamento, cada tarefa no SGE possui uma descrição com um conjunto de propriedades e definições que descrevem como deve ser executada. A prioridade dessas tarefas é definida por uma série de fatores: prioridade do usuário que submeteu; quais e quantos recursos necessita; deadline; quanto tempo está aguardando para executar; e outros. Além disso, também são considerados os tipos de fila e a carga atual dos recursos.

O esquema de filas do SGE é composto por uma série de instâncias de filas, uma por máquina. Diferentes filas podem ser criadas para diferentes configurações dentro do cluster, como mostrado na figura 2.4, de [Beginner's guide to sun tm Grid Engine 2009]. Há suporte para reserva antecipada de recursos, para acesso exclusivo a hosts, e controle de submissões. Além disso, permite gerenciamento multi-cluster e acordos de nível de serviço (SLA). A grande vantagem do SGE, é o custo, pois possui

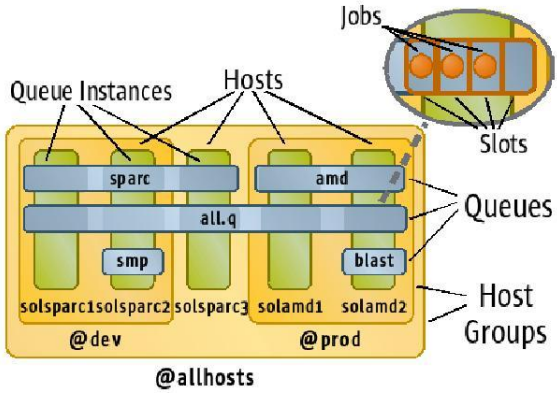

Figura 2.4: Exemplo de hosts e filas, de acordo com o manual [Beginner's guide to sun tm Grid Engine 2009]

uma versão *open source*, e com vasta documentação. Entretanto, possui ainda módulos importantes em desenvolvimento inicial, como o DRMAA [DRMAA], uma especificação para padronizar submissão de jobs, monitoramento, e controle em sistemas distribuídos. Está na versão 2.0, com muitas necessidades de implementação ainda em aberto. Desta forma, é mais trabalhoso usar a API e o código interno do SGE. Além disso, algumas funcionalidades extras que o LSF apresenta, ainda precisam de implementação no SGE, tal como o FlexNet Publisher (gerenciador de licença de softwares).

2.2.3.3 PBS/Torque

O Torque é um bom gerenciador de recursos, mas não tem a mesma velocidade de desenvolvimento como o SGE, ou o LSF. Utiliza o Moab como escalonador externo. o Torque controla as filas e os recursos, o Moab requisita ao servidor do Torque informações de dados dos jobs e sobre o estado dos nós do cluster, além de descrições. Com essa informação o Moab consegue decidir como gerenciar os jobs, de acordo com políticas, prioridades e reservas. Então o Moab direciona o Torque sobre onde e como deve executar os jobs. Com esta integração, o desenvolvimento de novas funcionalidades tem se concentrado muito mais no Moab que no Torque, o qual fica responsável apenas pela execução.

No trabalho de [Gaj et al. 2002] há uma comparação de desempenho entre o PBS e o LSF. Neste, foi mostrado que o LSF possui maior throughput que o PBS. Por outro lado, o PBS possui um tempo de turn around menor.

O LSF despacha jobs para host independentemente da velocidade relativa desses hosts, o que aumenta o throughput do sistema. Mas aumenta também o tempo de turn around, visto que máquinas lentas estarão executando e demoram mais para terminar as tarefas. Além disso, os jobs são executados um em seguida do outro no LSF, o que também ajuda no throughput. Mas para isso são utilizados algoritmos complexos.

Já o PBS distribui o job apenas em uma quantidade limitada de hosts rápidos e por isso apresenta um turn around mais rápido que o LSF. Entretanto, o tempo de execução médio é baixo, já que o throughput é menor. Além disso, a utilização dos recursos é mais limitada, visto que muitos recursos permanecem ociosos.

Desta forma, o LSF apresenta melhor desempenho principalmente no caso de jobs curtos, com taxa de submissão alta. Se a taxa for baixa, pode ser que o PBS fique na vantagem. Isso porque o PBS é melhor em tempo de turn around do que LSF, para qualquer tipo de job, principalmente os pequenos.

Ter um alto throughput é importante em casos de jobs grandes, nos quais o usuário não faz nenhum outro processamento enquanto o job não completar sua execução. Neste caso, a quantidade de jobs completados por unidade de tempo muito importante. Já o tempo de turnaround é mais interessante quando o usuário envia tarefas interativas, de modo que espera resultados rápidos para continuação do processamento.

2.2.3.4 Condor

O Condor é uma ferramenta que cria uma ambiente de alto desempenho, podendo gerenciar estações de clusters dedicados heterogêneos. A sua grande vantagem com relação aos outros gerenciadores por ter um sistema de checkpoint muito eficiente. Além disso, é o gerenciador de sistemas mais antigo, e permite a formação de grades, enquanto que a maioria dos gerenciadores coordenam no máximo ambientes multi-clusters. Entretanto, possui algumas restrições, oferece um suporte limitado a programação paralela, e tem suporte técnico não tão eficaz como os outros gerenciadores, com pouca documentação.

Sua característica de grade se baseia no cycle-scavenge. Ou seja, quando um segundo gerenciador escalona seus jobs em um processador que está sendo utilizado pelo Condor, ele faz checkpoint do seu job e o move para outra máquina. Desta forma, máquinas que estão reservadas por outros gerenciadores podem ser utilizadas enquanto ociosas.

2.3 Grades Computacionais

Grades computacionais se referem a ambientes geograficamente distribuídos e paralelos e colaborativos, voltados ao compartilhamento, seleção e agregação de serviços e recursos autônomos em larga escala, para solucionar problemas em organizações virtuais dinâmicas multi-institucionais. Esses ambientes são uma interessante solução para alcançar eficiência e alto desempenho na execução de aplicações, maximizando o uso dos recursos distribuídos[Foster 2002]. Esses recursos podem ser entendidos como processadores, discos, memórias, redes, largura de banda, clusters e outras facilidades, necessárias para a execução dos programas. A agregação possibilita uma maior disponibilidade dos recursos e serviços, melhorando a utilização do sistema como um todo[Foster, Kesselman e Tuecke 2001].

As principais características das camadas da arquitetura de uma grade são [Dantas 2005]:

- Ambiente: especificam determinadas operações locais que ocorrem em cada recurso como resultado do compartilhamento nas camadas superiores. Há mecanismos de negociação, para obter informações sobre a estrutura, o estado e as possibilidades dos recursos. E há me-

canismos de gerenciamento, para monitorar a qualidade de serviço;

- Conectividade: define os protocolos básicos para transações de rede específicas da grade;
- Recursos: a camada define protocolos e APIs (Application Program Interface) que forneçam segurança na negociação, iniciação, monitoramento, controle, geração de relatórios;
- Coletiva: atuam nas interações entre coleções de recursos.

É importante deixar claro as diferenças entre cluster e grades. No trabalho de [Colvero, Dantas e Cunha 2005] foi desenvolvida uma comparação entre estes tipos de ambientes. Nele, é mostrado que clusters refere-se a ambientes de um único domínio, com homogeneidade em sistema operacional. Já grades podem integrar múltiplos domínios heterogêneos. Além disso, atendem problemas de granularidade muito maior que clusters, e o custo para solução é compartilhado entre os domínios, em vez de se concentrar em uma única organização.

Segundo [Buyya 2008], se acontece o compartilhamento de recursos gerenciado por um único sistema global sincronizado e centralizado, então trata-se de um cluster. Neste ambiente, há gerenciamento local centralizado, em que todos os dispositivos trabalham cooperativamente em um objetivo comum. Por outro lado, grades integram e coordenam recursos e usuários de diferentes domínios que não possuem um controle central, ou seja, diferentes unidades administrativas de uma mesma empresa ou de companhias diferentes, e para execução de várias aplicações. Nestes ambientes, cada nó possui seu próprio gerente recursos e política de alocação.

Para [Foster 2002], uma grade deve integrar diferentes domínios, utilizando protocolos e interfaces padrões, o que permite a interoperabilidade. Além disso, devem ser de objetivos gerais, como autenticação, descoberta de recursos, políticas de acesso aos recursos. Caso contrário, o ambiente refere-se a aplicações para sistemas específicos, e não uma grade. Os serviços de uma grade são construídos sobre estes protocolos, oferecendo diferentes níveis de QoS.

Por outro lado, como visto anteriormente, existe a tendência para a utilização de ambientes multi-clusters. Esses ambientes permitem um gerenciamento entre clusters, entretanto, são limitados com relação a interoperabilidade. São utilizados protocolos e sistemas específicos dos middlewares.

Contrastando com multi-clusters, as grades são, em geral, altamente heterogêneas, abrangendo diferentes tipos de hardware, sistemas operacionais, dispositivos e serviços de diferentes provedores. As organizações

que compartilham recursos possuem diferenças em suas próprias políticas de uso em termos de hardware, software. Fora as diferenças físicas, há também distintas maneiras para descrição e alocação dos recursos, gerenciamento de processos, comunicação e segurança[Dantas 2005].

Este compartilhamento deve ser controlado, com uma definição clara dos provedores de recursos e consumidores, quais os recursos compartilhados, as políticas de permissões e condições para o compartilhamento. Desta forma, um conjunto de instituições definidas sob determinadas regras de compartilhamento é chamado de Organização Virtual (OV).

Para gerenciamento do compartilhamento desses recursos e dos serviços também existem na literatura vários projetos de middlewares para grades, como mostrado a seguir.

2.3.1 Middlewares

Existem vários middlewares para gerenciamento e controle de grades. Entre estes, é importante destacar o globus [Foster 2006], um pacote padrão para construção de grades que possui um conjunto de serviços básicos e bibliotecas de software para construção de ferramentas e aplicações voltadas a ambientes de grade, o Globus Toolkit.

Com objetivo de aprofundar o estudo sobre os tipos de middlewares para grades, e tê-lo como base para esta dissertação, desenvolvemos um trabalho comparativo entre alguns tipos de middlewares, o qual pode ser visto em [Ferreira et al. 2008].

Capítulo 3

Gerenciamento de Recursos em Grades

3.1 Semântica e Ontologias

Tratar da interoperabilidade no compartilhamento e coordenação do ambiente são tarefas complexas para usuários comuns, o sistema deve prover tecnologias de acesso em comum aos recursos dispersos entre as organizações da configuração, abstraindo as complexidades de gerenciamento destes recursos do usuários final. Um dos grandes desafio na área é oferecer transparência da heterogeneidade do sistema sem que isso comprometa o desempenho das aplicações [Buyyaa, Abramson e Venugopal 2005]. Neste contexto, middlewares tem sido desenvolvidos com objetivo de criar interfaces que abstraiam as complexidades das grades computacionais [Ferreira et al. 2008].

Uma das soluções para oferecer maiores abstrações ao usuário final são grades semânticas, as quais se apresentam como uma extensão das grades, nas quais informações, recursos e serviços possuem significados descritos em um modelo de dados semântico. Os dados são expressos por fatos, sentenças. Por meio desse conhecimento processável por máquinas, ocorre automação nas colaborações e processamentos em escalas globais, permitindo que computadores e pessoas trabalhem cooperativamente [Roure, Jennings e Shadbolt 2005].

A semântica facilita a descoberta de recursos, sua agregação automática e o entendimento de uso. Também é usada na descrição de tarefas; workflows, para facilitar a composição de recursos; na definição de quesito de QoS ou de segurança; para anotações de monitoramento do sistema; para facilitar a integração entre ambientes heterogêneos, com o mapeamento de terminologias de diferentes domínios; entre outros [Roure, Jennings e Shadbolt 2005].

Para descrever os recursos de grades são utilizadas ontologias, especificações formais e explícitas de um conceito compartilhado. Para alcançar interoperabilidade, ontologias são usadas no desenvolvimento de modelos que expressem o conhecimento de um determinado domínio, des-

crevendo conceitos, propriedades e atributos dos conceitos, relações entre eles, restrições[Roure, Jennings e Shadbolt 2005]. Desta forma, gera uma camada de comunicação única e comum a todos os usuários, com um vocabulário comum de entendimento compartilhado.

De acordo com [Su e Ilebrekke 2002], as linguagens para construção de ontologias são classificadas em :

- Tradicionais: possuem raiz na Inteligência Artificial ou na Engenharia de Conhecimento.

- Para Web: baseadas em padrões Web, para facilitar o intercâmbio de informações e seus significados na Internet, sendo a principal camada da Web Semântica. Um framework para construção de aplicações Web Semânticas é a API Jena.

Em [Wieder e Ziegler 2006], por meio da figura 3.1 são mostradas camadas de linguagens de marcação semântica para criação de ontologias, tendo como base XML e XML Schema. Em uma camada acima há o RDF[Klyne e Carroll 2004], que define expressões com sujeito, predicado, objeto, também chamadas de triplas, além de pares de valores de atributos. RDF Schema[Klein et al. 2003] surge para facilitar o entendimento de RDF, provendo uma definição formal de RDF, além de classes e propriedades. Neste contexto, todo recurso é nomeado com uma URI (Uniform Resource Identifier), através da qual se consegue acesso ao recurso.

Figura 3.1: Camadas de linguagens de marcação semântica, por [Wieder e Ziegler 2006]

No topo do RDF Schema, há várias linguagens. DAML+OIL [Connolly et al. 2001] é uma combinação entre DAML (DARPA Agent Markup

Language), linguagem de ontologia DAML-ONT, e OIL, que é a camada de ontologia de inferência para o RDF Schema. O OWL (Web Ontology Language)[McGuinness e Harmelen 2004] é baseado em DAML+OIL e é recomendação da W3C. Possui três sub-linguagens, OWL Lite, OWL DL e OWL Full.

3.2 Alocação de Recursos

A linguagem semântica é importante tanto para descrição dos recursos disponíveis do sistema, como para descrição de que tipo de recursos o usuário quer alocar. Quando uma tarefa é submetida a um gerenciador de tarefas, ela deve trazer consigo a descrição dos recursos necessários para execução (i.e., quantidade de processadores, memória, armazenamento necessário). Um escalonador decide a ordem de execução destas tarefas, adotando determinada política de escalonamento. Quando as tarefas são executadas, determinados recursos serão alocados para ela, de acordo com os requisitos da tarefa e também dependendo do estado do sistema com relação a carga de trabalho nos recursos.

Ou seja, a alocação de recursos é necessária para manter os recursos especificados pelo usuário reservados para a aplicação durante a execução da mesma. O tempo em que o recurso deve ficar alocado pode ser definido pelo requisitante ou pela política de alocação do sistema. Alguns problemas devem ser resolvidos para efetivar a alocação de recursos em grades. Entre estes problemas, estão: comparar as necessidades do usuário com o ambiente disponível; verificar a disponibilidade desses recursos para o momento requisitado; disponibilizar ou negociar outras opções de alocação, caso não seja possível atender a todas as requisições do usuário. A decisão de quais recursos serão alocados fica a cargo do escalonador e as políticas adotas por ele.

Para alocar os recursos, o escalonador pode adotar políticas de *space-sharing*, na qual os recursos disponíveis são particionados entre os processos que executam ao mesmo tempo. Este tipo de escalonamento cria partições disjuntas, independentes. Por exemplo, no SGE [Beginner's guide to sun tm Grid Engine 2009], quando diferentes *jobs* com requisitos de memória rodam em uma única máquina, a memória dessa máquina pode ser particionada entre os *jobs*. Para isso, pode-se configurar diferentes filas para cada um dos *jobs*, e associar uma porção da memória total da máquina para cada fila. Esta política de *space-sharing* pode ser combinada com *time-sharing*, com a qual o recurso atende cada *job* por um determinado espaço de tempo.

É possível subdividir as formas de alocação de recursos em:

- Alocação estática: os recursos são alocados de forma fixa, ou seja,

depois de alocado não pode mais ser alterado. Este tipo de alocação pode causar fragmentação, pois não permite a realocação, caso necessário para evitar pequenos fragmentos de recurso não utilizados. Desta forma não é possível aproveitar ao máximo a capacidade do ambiente.

- Alocação adaptativa: há flexibilidade dos recursos a serem alocados. O *job* pode iniciar com um espaço reduzido e, quando houver maior disponibilidade de recursos no sistema, expandir-se. Ou seja, são alocados mais ou menos recursos de acordo com a carga atual do sistema. Desta forma, a fragmentação é reduzida e há uma melhora na utilização dos recursos e a eficiência. Nota-se na literatura que o tempo de resposta melhora com essas propostas.

- Alocação semi-adaptativa: a quantidade de recursos a ser alocada para o *job* é flexível, mas a adaptação ocorre apenas em tempo de inicialização, durante a execução não é possível alterá-la. Desta forma, o *job* fica preso a um determinado tamanho (quantidade de recursos), não sendo eficiente para *jobs* muito longos. Apenas modelar o *job* pode levar a uma alocação ruim, se não for combinada com preempção.

- Alocação semi-estática: prediz o melhor momento de começar e melhor tamanho, ou seja, procura o tamanho ótimo dos *jobs* por meio de simulações. Há um comportamento cooperativo entre *jobs*, todos se configuram na hora da submissão, requerendo a quantidade de nós mais apropriada. Essas simulações apresentam melhoras significativas no tempo de resposta.

- Alocação dinâmica: são criados tantos processos quanto a quantidade de processadores alocados inicialmente para a aplicação. Durante a execução pode ocorrer adaptação, alterando o número de processos. É possível reconfigurar *jobs*, remover ou adicionar processos. A adaptação ocorre em determinados pontos do código. Pode ser quando chegam novas tarefas, ou quando terminam. Ou apenas quando o sistema não está muito carregado. Pode também ser feito em determinados períodos. Entretanto, envolve mais complexidades que os outros tipos de alocação.

Para que a alocação seja eficiente, deve suprir as necessidades de ambos os lados, usuários e provedores. Para suprir as necessidades dos provedores, é importante que os mecanismos adotados evitem a subutilização dos recursos. Ao mesmo tempo, para aumentar as garantias que a

aplicação do usuário seja executada com sucesso, trabalha-se com os piores casos, estimando-se grandes cargas para o sistema, além de serem criados *backups*para tolerância a falhas. Entretanto, estas medidas acarretam na subutilização dos recursos [Korpyljov et al. 2009].

Para melhorar a utilização dos recursos no sistema, é possível adotar políticas mais flexíveis de alocação, como as reservas antecipadas, as quais serão descritas na próxima seção.

3.3 Reserva Antecipada de Recursos

Existem aplicações que exigem maior garantia de execução em tempo determinado e de qualidade de serviço, como alguns sistemas multimídias, cirurgias remotas, conferências, *video-on-demand*. Muitas dessas aplicações possuem dados críticos para serem processados. Uma solução para este problema é usar reservas antecipadas de recursos. No trabalho de [Degermark et al. 1995] são mostrados alguns exemplos das necessidades do uso de reserva antecipada.

A reserva antecipada baseia-se em planejar as necessidades dos clientes para que seja feito o compartilhamento das capacidades dos recursos de forma a aumentar a utilização global desses recursos e também garantir a qualidade de serviço requisitada por cada cliente. Ou seja, na falta de um sistema adequado de reserva, pode ocorrer aumento de custo por se alocar uma quantidade de recursos desnecessária, ou então pode ocorrer o contrário, o serviço ser degradado por demanda excessiva de recurso [Foster et al. 1999].

Os principais parâmetros de uma reserva são o *start time*, que representa o momento em que o recurso requisitado deve ser alocado; e a duração necessária para esta alocação, para que o gerenciador saiba quando liberar o recurso. Na figura 3.2, de [Wolf e Steinmetz 1997], as reservas de recursos são classificadas de acordo com o tempo de reserva.

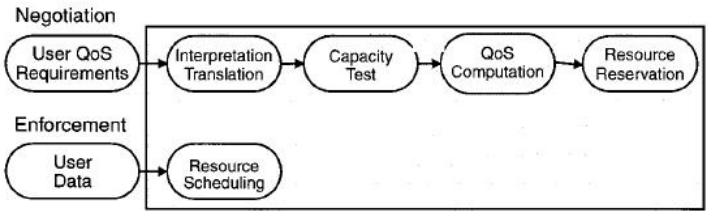

Figura 3.2: Fases do Gerenciamento de Recursos [Wolf e Steinmetz 1997]

Observa-se pela figura 3.2 que as reservas podem ser estáticas, nas quais os recursos são alocados durante todo o seu tempo ativo; e as reservas dinâmicas. Nas dinâmicas, existem reservas imediatas, entre as quais pode-se saber ou não o tempo de duração. E há as reservas antecipadas, em que aloca-se para um futuro próximo o recurso.

A figura 3.3 mostra os principais estados do gerenciamento de recursos em grades utilizando reserva avançada de recursos.

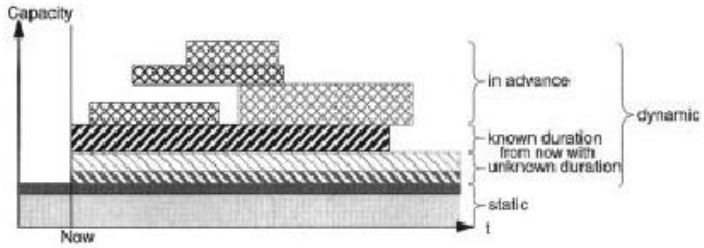

Figura 3.3: Tipos de Reserva [Wolf e Steinmetz 1997]

Primeiro ocorre uma fase de negociação entre o cliente (com os requisitos para sua aplicação) e o provedor (com determinada disponibilidade dos recursos). Nesta fase, o cliente entra com os requisitos de QoS que necessita, e é necessário ser feita a tradução da requisição para que seja aplicável aos recursos do ambiente. Nesta fase podem ser adotados métodos semânticos, como mostrado na seção anterior. Depois que consegue entender a requisição, o gerenciador verifica se o ambiente tem capacidade e se possui recursos disponíveis suficientes para responder. Além disso, também é verificado o desempenho que estes recursos podem oferecer para a específica requisição. Finalmente, caso estas questões estejam em ordem, é feita a reserva. Caso não seja possível responder a requisição, deve-se tentar negociar com o cliente alguma outra configuração como resposta.

A segunda fase é a de alocação e o escalonamento dos processos em si. Caso se trate de uma reserva imediata, esta fase ocorre em seguida à reserva. Já na reserva antecipada, pode haver um longo período até que os recursos sejam efetivamente alocados e os processos escalonados. O cliente só precisa enviar os dados para execução no momento em que esta for ocorrer efetivamente.

Todavia, ainda há divergências sobre as formas de implementação destes mecanismos há alguns desafios. A maior dificuldade é falta de su-

porte à reserva antecipada nos ambientes já em execução. Além disso, é complexo lidar com a grande heterogeneidade dos sistemas de grade, pois os recursos localizam-se em diferentes domínios, com diferentes políticas de controle e interfaces para comunicação [Foster et al. 1999]. Um terceiro problema é lidar com a complexidade das requisições, pois os clientes vão querer reservar vários recursos e não um só. Então o gerenciador deve coordenar a descoberta, seleção, alocação e reserva de vários recursos simultaneamente, e muitas vezes com dependências entre si. Este é um problema tratado no meta-escalonamento, como será visto na próxima seção.

3.4 Metacomputação

A metacomputação combina recursos geograficamente separados para atender a aplicações dos clientes. Essas aplicações podem ter vários componentes que interagem entre si, e seus processos devem ser escalonados de forma eficiente entre os recursos disponíveis no ambiente. Geralmente clusters de alto desempenho utilizam gerenciadores de recurso locais para coordenação interna dos processos e aplicações locais. Já em uma grade, abrangendo ambientes multi-clusters em diferentes domínios, deve haver um gerenciador global, que tenha controle sobre o sistema como um todo. O escalonamento desses processos entre os diferentes domínios é chamado de meta-escalonamento, o qual deve orquestrar os processos, recursos e comunicações de forma a reduzir o tempo de resposta para os clientes e otimizar a utilização dos recursos [Weissman 1998].

Desta forma, a principal tarefa do meta-escalonador é receber as requisições dos clientes e encaminhá-las aos sistemas mais adequados para atendê-las. Apesar de os gerenciadores de recursos locais terem sua independência, muitas vezes as decisões do meta-escalonador podem se sobrepor à eles. Para conseguir orquestrar todos os ambientes multi-domínios e heterogêneos que compõem uma grade, algoritmos eficientes para co-escalonamento, co-alocação e co-reserva de recursos são necessários.

3.4.1 Co-Alocação

Como visto anteriormente, a alocação de recursos foca no mapeamento das aplicações de um ou vários usuários sobre os recursos compartilhados do ambiente, com objetivos de reduzir o tempo de resposta geral, ou seja, das requisições como um todo, e melhor aproveitar os recursos disponíveis. Uma forma de reduzir o tempo de resposta geral é permitir que as aplicações executem seus processos globalmente, ou seja, que não se restrinjam a alocar recursos de um único cluster ou de um único domínio, permitindo, assim a co-alocação [Qin e Bauer 2009].

A co-alocação implica em dividir os processos de uma aplicação

em grupos, e alocar esses grupos de processos em diferentes clusters. Esse processo é importante principalmente na alocação de aplicações de alta granularidade. Considerando um ambiente de grade com múltiplos usuários compartilhando os recursos, uma aplicação que necessite de grande quantidade de processamento pode ter que esperar muito até conseguir recursos suficientes disponíveis para execução, tratando-se de um único cluster. Por outro lado, com a opção da co-alocação é possível que parte dos processos aproveitem recursos disponíveis de clusters diferentes, e os processos não precisem esperar tanto em fila.

Além disso, a co-alocação aumenta o aproveitamento do sistema como um todo, pois pequenas parcelas de recurso nos clusters que ficariam ociosos, podem ser utilizados por partes das aplicações. Ou seja, evita a fragmentação do ambiente.

Encontram-se na literatura alguns estudos sobre o desempenho geral e de comunicação entre processos com diferentes escalonadores, adotando a co-alocação (e.g. [Banen, Bucur e Epema 2003], [Li 2005]). Esses trabalhos mostram que adotar a co-alocação pode piorar muito o desempenho geral do sistema, caso não haja alguma política de controle do escalonamento dos processos entre clusters. Isso ocorre porque ao distribuir os processos de uma mesma aplicação em diferentes clusters, pode ocorrer destes processos dependerem de comunicação entre si, e, consequentemente acarretar na comunicação entre clusters. Esta comunicação pode ser dispendiosa, dependendo da capacidade dos links entre clusters e afetar significantemente o tempo de resposta [Qin e Bauer 2006].

3.4.2 Co-Reserva

O conceito de co-reserva refere-se a um conjunto de reservas antecipadas com relações temporais e espaciais entre si. Relação temporal, significa que o horário de uso de um recurso tem alguma relação com o horário de uso de algum outro recurso, como quando aloca-se processadores e a largura de banda necessária para comunicação durante o uso desses processadores. Contudo, co-reserva não significa que os recursos devem ser utilizados apenas simultaneamente. Por exemplo, como mostrado no trabalho de [Roblitz e Reinefeld 2005], em que reserva-se CPUs por 6 horas e um pipeline de visualização para 4 horas depois da reserva dessas CPUs. Já a relação espacial refere-se a localização desses recursos reservados, como reservar uma rede de interconexão entre essas CPUs e o pipeline, no mesmo horário em que o pipeline for reservado.

Geralmente são usadas tabelas *time-slot* para representar a porcentagem de recursos disponíveis e alocados no tempo, tanto recursos computacionais quanto de comunicação (e.g. [Foster et al. 1999], [Barz, Pilz

e Wichmann 2008]). A figura 3.4 mostra um exemplo típico de tabela *time-slot*, com as alocações correntes e as reservas futuras. Os blocos representam processos das aplicações em execução.

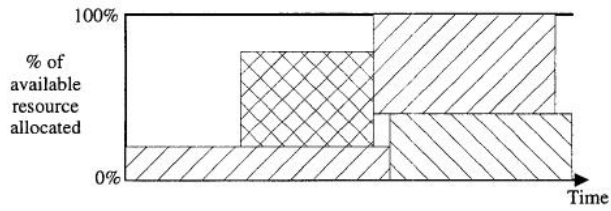

Figura 3.4: Exemplo de tabela *time-slot*, de [Foster et al. 1999]

3.4.3 Co-Escalonamento

Escalonadores decidem quando e onde um job vai executar, representando uma camada acima do sistema operacional, o qual escalona os jobs admitidos para execução. O escalonamento independente de processos, sem levar em consideração alguma política, pode comprometer o desempenho em sistema concorrentes distribuídos. Isso porque múltiplos processos comunicando entre processadores pode gerar muitos chaveamentos . Outro problema é que pode haver processadores ociosos, com processos na espera de comunicação, diminuindo o nível de utilização e possivelmente o tempo de resposta geral do ambiente.

Co-escalonamento é um conceito de sistemas distribuídos concorrentes, o qual significa escalonar processos/*threads* de uma mesma aplicação ao mesmo tempo e de forma coordenada. Aplicações de granularidade fina apresentam melhores resultados quando co-escalonadas. O co-escalonamento pode ser de dois tipos [Sodan 2005]:

Gang: as tarefas são agrupadas, de forma que todos os processos dependentes entre si executem simultaneamente, compartilhando recursos com *time sharing*. Assim não precisão esperar para receber ou responder a uma requisição. Não é muito flexível, as decisões de escalonamento são estritamente executadas.

Loosely: mais flexível na execução das tarefas, as decisões de escalonamento são seguidas aproximadamente, podendo ter alterações dinâmicas de acordo com as necessidades no momento da alocação.

3.4.4 Meta-Escalonadores

Um meta escalonador provê interfaces para acesso virtual aos recursos. Adota políticas globais para os provedores de recursos e consumidores. Desta forma, são definidas regras importantes em um ambiente de grade com co-escalonamento [Ding et al. 2009].

A seguir serão apresentados dois meta-escalonadores, para exemplificar suas funcionalidades. Foram escolhidos o CSF4 e o GridWay, por serem de código livre e devido a importância destes na literatura, pois ambos já fizeram parte do Globus toolkit.

3.4.4.1 GridWay

O Gridway [GridWay Metascheduler] é um meta-escalonador que permite compartilhamento de recursos em clusters, grades, supercomputadores, servidores. Suporta diferentes gerenciadores de recursos, tal como PBS, SGE, LSF, Condor, podendo organizar estes ambientes como um único sistema ou em diferentes domínios administrativos. A idéia é oferecer um ponto de acesso para todos os recursos da organização.

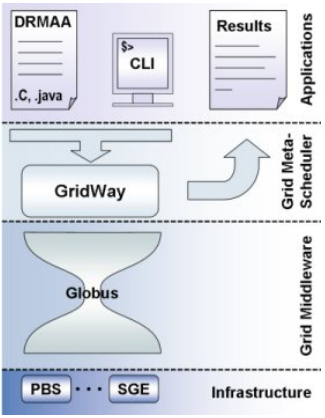

Figura 3.5: Configuração de ambiente com GridWay

A figura 3.5 representa a configuração do ambiente utilizando o GridWay, mostrando sua relação com o Globus. Ele é desenvolvido em uma camada acima da do globus, utilizando suas aplicações para acesso aos recursos. A API do DRMAA (*Distributed Resource Management Application API*) é usada como uma interface homogênea para os diferentes gerenciadores distribuídos locais. E o Meta-escalonador utiliza em con-

junto ferramentas e aplicações do globus nos ambientes locais de clusters.

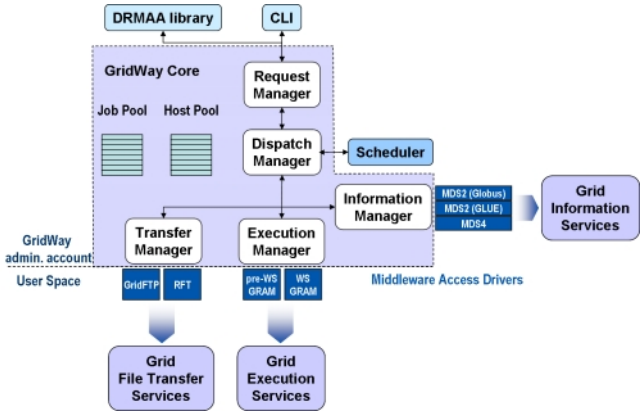

Figura 3.6: Principais componentes do Gridway

A arquitetura do GridWay pode ser vista na figura 3.6 e consiste dos seguintes componentes:

Interface do usuário: possui a API do DRMAA, e comandos básicos de submissão de processos, migração, monitoramento, entre outros.

Gridway core: responsável pelo gerenciamento da execução dos processos, oferecendo o escalonamento, tolerância a falhas, recuperação.

Scheduler toma as decisões de escalonamento e alocação dos recursos disponíveis na grade para os jobs submetidos.

Information Manager faz interface com serviços disponíveis para monitoramento e descoberta de recursos na grade.

Execution Manager faz interface com os serviços disponíveis para gerenciamento de jobs.

Transfer Manager faz interface com os serviços disponíveis para gerenciamento de dados.

3.4.4.2 CSF4

O CSF4 [Ding et al. 2009], um meta-escalonador WSRF (Web Services Resource Framework) que permite trabalhar com diferentes escalonadores locais. Desta forma, integra gerenciadores de clusters, entre os

quais LSF, SGE, PBS e Condor. Ele mantém independência dos escalonadores locais, mas tem uma visão global dos recursos, permitindo uma melhor utilização dos mesmos. Além disso, está incluído no Globus Toolkit GT4 e GT2.

Utilizando um escalonador global como o CSF4 é possível submeter jobs tanto para clusters individuais como para múltiplos clusters. Ao mesmo tempo controlar a execução dos jobs nos diversos ambientes heterogêneos. Ou seja, em uma configuração onde haja clusters com diferentes gerenciadores, como o SGE, LSF, e até máquinas individuais, o CSF pode enviar um job para um único gerenciador, acrescentando no comando para qual gerenciador está enviando. Ao mesmo tempo, também pode enviar integrar múltiplos clusters heterogêneos para processamento, utilizando o MPI, por exemplo 3.7.

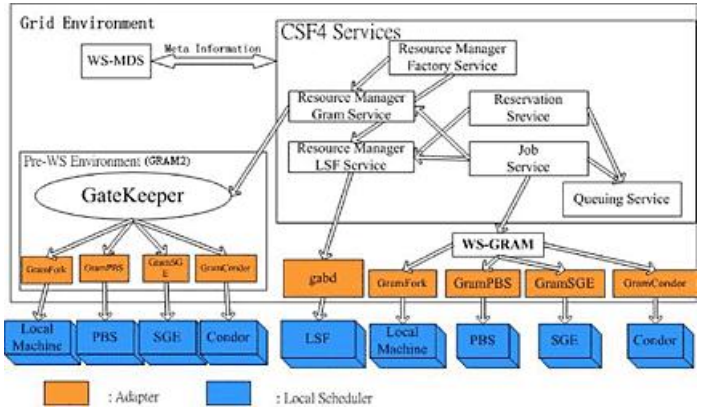

Figura 3.7: Arquitetura do CSF4

Uma das grandes vantagens do CSF4 é que oferece suporte a reservas de recursos globais, enquanto o GridWay ainda não. Entretanto o CSF4 não participa em conjunto das últimas atualizações do Globus, possuindo pouca documentação.

3.5 Trabalhos Correlatos

3.5.1 Selecionador de recursos estático utilizando ontologias [Silva e Dantas 2007]

Como foi visto, a alocação de recursos é um processo importante no gerenciamento de uma grade computacional. O mecanismo escolhido para tal processo é um fator essencial na eficiência do compartilhamento

de recursos entre grades, influenciando no desempenho do sistema, nível de utilização dos recursos, usabilidade para o usuário final.

Neste contexto, as ontologias se mostraram como uma importante ferramenta na descrição dos recursos de grades. Por meio das ontologias, é possível a integração de diferentes grades computacionais, abstraindo suas complexidades, para que o usuário final enxergue os ambientes como um único sistema.

Em [Silva e Dantas 2007] é feito um trabalho que trata processo de mapeamento de ontologias para seleção de recursos em ambientes de grades computacionais. É apresentada uma ferramenta desenvolvida para facilitar o processo de *matching* entre recursos de grades. Para tal, foi utilizado um método de integração semântica baseado na descrição de múltiplas ontologias. Foram consideradas interações humanas com objetivo de construir um conhecimento mais lógico na criação de requisições, visto que sistemas inteiramente automáticos não são capazes de reconhecer todas as possíveis relações entre as diferentes ontologias. Para integrar as diferentes ontologias, foi desenvolvida uma ontologia compartilhada em comum (OR), que pode ser vista na figura 3.10.

Para que o administrador do sistema tenha seus recursos cadastrados no servidor, é necessário que estabeleça manualmente relações entre a descrição dos recursos de sua organização virtual e a OR, como visto na figura 3.8. Essa relação deve ser informada ao *matchmaker*, que fará o trabalho da associação durante consultas.

Também foi criada uma ontologia de requisição (QO), que auxilia o integrador de ontologias para que o usuário final possa fazer consultas de recursos no ambiente, como mostrado na figura 3.9. Por meio destas consultas, o usuário consegue restringir o tipo de sistema operacional a ser utilizado, versão do mesmo, arquitetura e velocidade do processador, quantidade de memória, entre outros parâmetros.

[Silva e Dantas 2007] trata da heterogeneidade e integração de OVs, criando uma solução para tal. Após uma consulta para determinada tarefa, o usuário recebe um retorno, com os IPs dos possíveis recursos para executar a tarefa requisitada. Outro aspecto importante do trabalho é que há uma interface de fácil interação para o usuário final. Entretanto, ao realizar a consulta no sistema, [Silva e Dantas 2007] considera apenas as ontologias previamente inscritas pelos provedores de forma estática, sem levar em consideração se os recursos que vieram como resposta à pesquisa estão ocupados no momento ou não. Ou seja, é feita apenas uma seleção dos recursos.

Em contrapartida, [Qin e Bauer 2009] trata de sistemas dinâmicos na alocação de recursos, levando em consideração apenas a quantidade

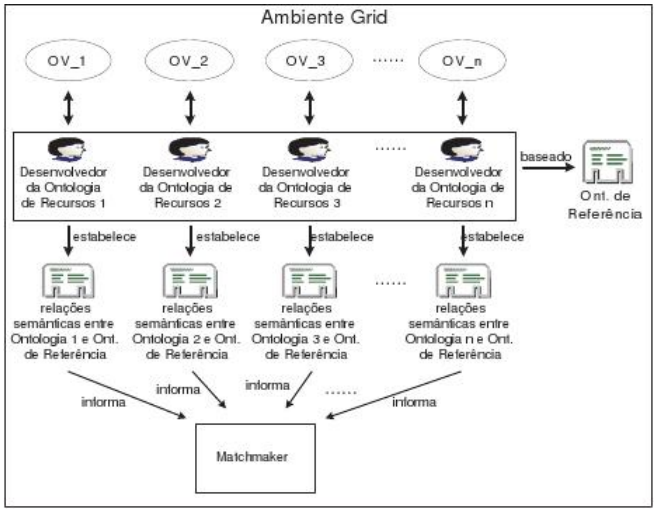

Figura 3.8: Funcionamento do matchmaker, de [Silva e Dantas 2007]

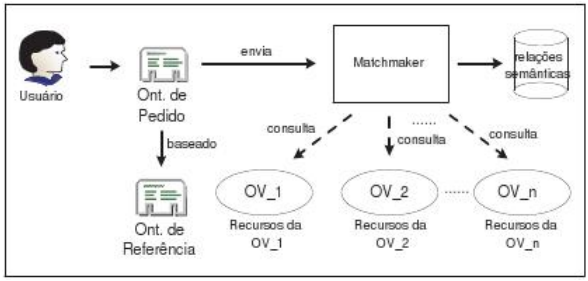

Figura 3.9: Consulta a Recursos, de [Silva e Dantas 2007]

de processadores e o nível de utilização da banda, como será detalhado a seguir.

3.5.2 Co-alocador de recursos baseado na lógica difusa [Qin e Bauer 2009]

Com o objetivo de satisfazer a demanda crescente de de recursos computacionais compartilhados em organizações e melhorar o nível de utilização destes recursos, no trabalho de [Qin e Bauer 2009] foi desenvolvida

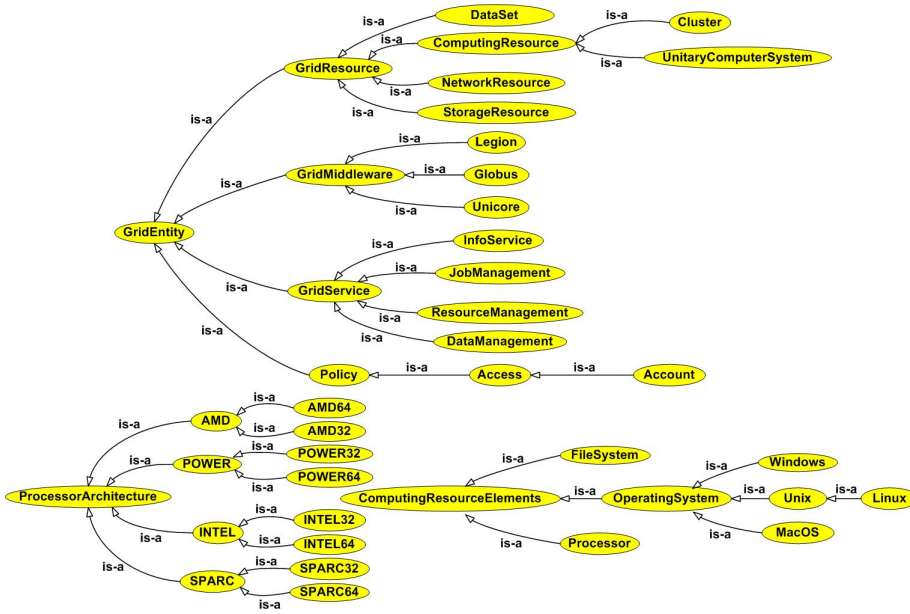

Figura 3.10: Ontologia de Referência, de [Silva e Dantas 2007]

uma estratégia para co-alocação de recursos. O trabalho trata do problema de alocação de recursos em configurações de multi-clusters computacionais quando há jobs grandes para serem acomodados no sistema, sendo necessária a distribuição de seus processos entre clusters. Esta estratégia se baseia na geração de duas constantes, as quais são usadas para controlar o nível de saturação dos links entre clusters (λ), e para controlar a divisão dos processos durante a co-alocação (δ).

O trabalho aproveita da lógica difusa para gerar estes dois parâmetros, como detalhado no Apêndice A. A lógica difusa é interessante de ser utilizada pois é baseada na incerteza e imprecisão, permite distinguir classificações e oferece um determinado nível de confiança. É uma lógica multi-valorada, permitindo que se avaliem diferentes aspectos do parâmetro avaliado.

Para gerar o controle do nível de saturação entre clusters, λ, são realizados cálculos baseados em quatro parâmetros: o tamanho da requisição em quantidade de processadores, o tipo de comunicação entre clusters, a largura de banda máxima do link e o nível de utilização corrente do link.

As figuras 3.11, 3.12 e 3.13 mostram como é feito alguns dos cálculos utilizando lógica difusa.

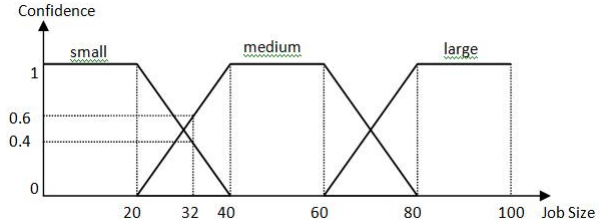

Figura 3.11: Função relacionada ao tamanho do job, de [Qin e Bauer 2009]

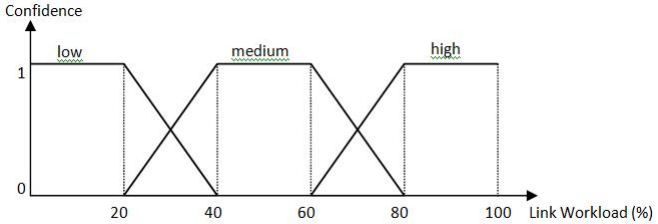

Figura 3.12: Função relacionada à carga do link, de [Qin e Bauer 2009]

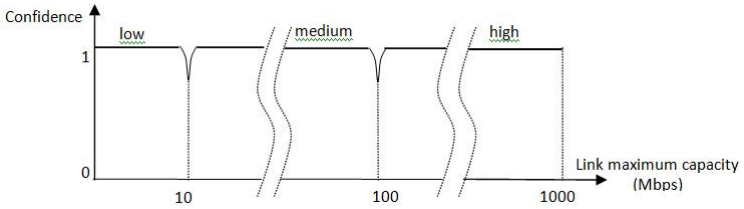

Figura 3.13: Função relacionada à largura de banda máxima, de [Qin e Bauer 2009]

Algumas funções são definidas para oferecer um grau de associação entre os valores de entrada e de saída. Por meio de uma tabela de

regras, esses valores são mapeados. Posteriormente, esses valores adquiridos nas funções são utilizados para gerar os valores finais de λ e δ. Para isso é utilizado o método de centro de gravidade CoG (*center of gravity*) [Bai, Zhuang e Wang 2006], [Wang 1997]. Desta forma, primeiramente calcula-se o valor de λ, para calcular o controle da saturação dos links entre clusters. Os resutados dos clusters selecionados nesta primeira fase são utilizados na segunda fase para o cálculo do valor de δ, para controlar como os processos serão distribuídos. Este controle é baseado apenas no tamanho da requisição em quantidade de processadores e o tipo de comunicação entre clusters (*peer-to-peer* ou *master-slave*). Desta forma, este parâmetro é ajustado diferentemente para jobs grandes e jobs pequenos.

Usando este método, os autores conseguiram reduzir o tempo de resposta geral do sistema e aumenta a utilização dos recursos do ambiente. Entretanto, não são consideradas questões sobre reserva de recursos para jobs críticos. Além disso, não leva em conta a heterogeneidade do sistema para descrição dos recursos, como no trabalho anterior mostrado sobre ontologias.

3.5.3 Novo algoritmo difuso para escalonamento global para ambientes multi-clusters em grades [Vahdat-Nejad, Monsefi e Naghibzadeh 2007]

No trabalho de [Vahdat-Nejad, Monsefi e Naghibzadeh 2007], é proposto um algoritmo utilizando lógica difusa para escalonamento global em ambientes de grades com configurações multi-clusters. É utilizada uma arquitetura em camadas, na qual um escalonador global (na camada da grade), administra jobs e clusters no ambiente, determinando em qual cluster determinado job deve executar. Essa decisão é baseada nas necessidades de processamento do job e na quantidade de comunicação entre processos do job. Após o escalonamento global, o escalonador local do cluster decide em que máquinas o job deve executar, aplicando a política de FIFO (*first-in first-out*).

O trabalho considera que jobs com quantidade de comunicação significativa entre processos podem ter um tempo de resposta mais lento, dependendo da carga de comunicação na rede interna do cluster. Desta forma, a questão tratada neste trabalho é sobre qual cluster é mais adequado para execução de cada job. Para calcular a forma mais adequada de escalonamento do job, foi criado um método com lógica difusa, utilizando a carga de comunicação atual do cluster e os requisitos de comunicação do job. Assim, um job com taxa alta de comunicação entre processos deve ser escalonado para clusters com rede pouco carregada, criando um sistema de prioridades entre os clusters disponíveis no ambiente.

São aplicados pesos (W1 e W2) aos clusters, de forma a criar prioridades, que determinam o local de execução do job. O primeiro peso, W1, refere-se à quantidade de máquinas disponíveis no cluster com baixo processamento, com relação à quantidade de processos do job. O segundo peso, W2, refere-se aos requisitos de comunicação do job, com relação a carga da rede no cluster. Foi gerado então um coeficiente para determinar a fórmula de prioridade do cluster com relação ao job: *0.7 W1 + 0.3 W2*. Calculada a prioridade de cada cluster para cada job, estes podem ser escalonados.

A lógica difusa é aplicada no cálculo do peso W2. A tabela 3.1 mostra as regras usadas no mapeamento entre os parâmetros de entrada e saída utilizados nas funções.

Tabela 3.1: Regras para mapeamento de parâmetros em [Vahdat-Nejad, Monsefi e Naghibzadeh 2007]

Ratio/AvailableBW	Low	Medium	High
Low	Moderate	Large	Large
Medium	Small	Moderate	Large
High	Very Small	Small	Large

O trabalho faz simulações comparando seus resultados com resultados utilizando o algoritmo de *best-fit*, e consegue resultados significativos, diminuindo o tempo de resposta dos jobs. Isto acontece porque o algoritmo de *best-fit* ignora os requisitos de comunicação entre processos dos jobs, e o tráfego na rede interna dos clusters.

Entretanto, este trabalho oferece ao usuário uma quantidade pequena de parâmetros de requisitos do job para a decisão de escalonamento, avaliando apenas as necessidades de comunicação e processamento. Além disso, o critério de prioridades adotado com relação ao processamento não permite co-alocação, diminuindo as chances de jobs com maiores necessidades de processamento serem escalonados rapidamente, como acontece com jobs menores.

3.5.4 Um mecanismo de reserva antecipada de banda em grades de video usando lógica difusa [Liu, Dai e Chuang 2006]

No trabalho de [Liu, Dai e Chuang 2006] é proposto um algoritmo de controle inteligente para reserva antecipada de recursos em grades, utilizando lógica difusa e redes neurais. A idéia é dar suporte para serviços

de videos em rede, utilizando reserva de recursos antecipada para oferecer um QoS garantido no futuro.

O trabalho foca nos parâmetros de *delay* (atraso) e *jitter*, os quais são importantes na entrega de vídeos e reconstrução de qualidade. Entretanto, são parâmetros muito imprecisos para os usuários quando requeridos antecipadamente. Por isso é difícil criar modelos matemáticos baseados nessas variáveis e que ofereçam resultados satisfatórios com relação aos requisitos de recursos da rede em uma reserva antecipada.

Desta forma, é proposto em [Liu, Dai e Chuang 2006] um modelo inteligente de previsão de prioridade dos requisitos na reserva antecipada de recursos baseado em redes neurais difusas, utilizando as variáveis de *delay* e *jitter* para decisão de prioridade na execução. Redes neurais e sistemas difusos são associados em um sistema integrado, combinando a capacidade de aprendizado das redes neurais com a capacidade de solução da lógica difusa.

No método utilizado, há dois sinais de entrada, correspondendo ao *delay* e *jitter*. O sinal de saída representa os conjuntos difusos de prioridade sobre as reservas avançadas. Para cada entrada são definidos 3 conjuntos, representando os valores grande, médio e pequeno. Foi desenvolvida uma arquitetura de rede neural difusa com 10 camadas, tendo como resultado a prioridade das requisições de recursos na rede.

Durante este processo, são criados 2 conjuntos, um de requisições de alta prioridade, e outro com requisições de baixa prioridade. Estes são utilizados no algoritmo de controle para a execução da reserva avançada. Este algoritmo é usado para encontrar um caminho e banda disponível para transmissão do vídeo. Caso não haja banda disponível de acordo com o pedido nas requisições de alta prioridade, a banda é desalocada para que requisições de baixa prioridade possam utilizá-la.

Foi utilizado um modelo de simulação para testar o algoritmo proposto, com a topologia eqos [Burchard e Heiss 2003]. Os resultados mostraram bom balanceamento de carga e desempenho de rede, além de resolver conflitos de QoS relacionados a prioridade das requisições.

Entretanto, é muito impreciso utilizar parâmetros de *delay* e *jitter* antecipadamente. Além disso, os testes levaram em consideração apenas o parâmetro de banda. Apesar de comentar a possibilidade de utilizar o mesmo com outros recursos, não há testes mostrando a eficiência em outros casos. Por fim, não é comentado no artigo sobre a possibilidade dc co-alocação.

3.5.5 *ARSimpleSpaceShared* do GridSim 5.0 [Sulistio e Buyya 2004]

O GridSim é um pacote para simulação de ambientes de grade baseada em Java. Possui ferramentas para composição de aplicações, informação e descoberta de serviços, interfaces para aplicações e permite a composição de modelos com recursos heterogêneos em diferentes organizações. O pacote permite a implementação de escalonadores específicos para o compartilhamento dos recursos, mas também oferece dois escalonadores padrões para uso: *TimeShared*, usando política de *Round Robin*; ou *SpaceShared*, com *First Come First Serve (FCFS)*.

Na implementação da reserva antecipada de recursos do GridSim, as entidades de recursos possuem características complementares e é adotado um escalonador com política própria de reserva, o *ARSimpleSpaceShared*, que usa a abordagem *FCFS*. Como este simulador oferece suporte a reserva antecipada de recursos, se tornou uma ferramenta interessante para simulação do funcionamento do GRADI. E, além disso, sua política de reserva também faz parte dos trabalhos relacionados para comparação.

A figura 3.14 mostra o diagrama de transição de estados da reserva antecipada no GridSim. Para aceitar uma reserva, o escalonador confere possíveis conflitos com as reservas já realizadas ou com o tamanho do *slot* livre. As principais estruturas usadas são uma lista com as reservas realizadas e uma lista de índices.

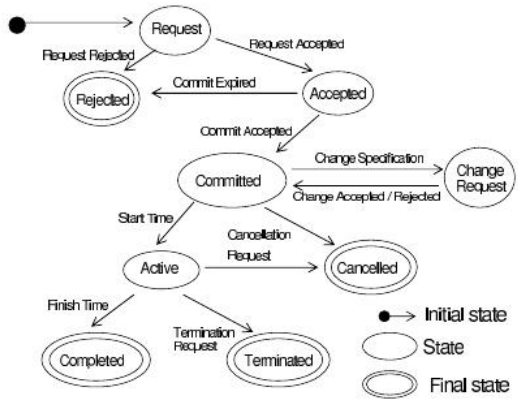

Figura 3.14: Diagrama de estados na reserva antecipada do GridSim, por [Sulistio e Buyya 2004]

3.5.6 Comparação entre os trabalhos

A tabela 3.2 mostra uma análise comparativa entre os trabalhos relacionados citados em cada subseção.

O trabalho citado na subseção [Silva e Dantas 2007] mostra um selecionador estático de recursos em grades. Este trabalho tem a vantagem de permitir a seleção de uma grande variedade de recursos, tal como tipo de processador, tipo de sistema operacional, quantidade mínima de processadores, disco rígido, memória, entre outros. Entretanto, esta seleção ocorre de forma estática, não havendo maiores garantias de seleção, diferentemente dos outros trabalhos, que permitem alocação dinâmica dos recursos.

Uma segunda vantagem do trabalho em [Silva e Dantas 2007] é que permite a integração de várias organizações virtuais, fazendo o matching de diferentes descrições, o que não é possível nos outros trabalhos correlatos.

Por outro lado, os trabalhos descritos em [Vahdat-Nejad, Monsefi e Naghibzadeh 2007], [Liu, Dai e Chuang 2006] e [Sulistio e Buyya 2004] realizam reserva de recursos. O primeiro faz a reserva baseado na quantidade de comunicação na rede interna do cluster, enquanto o segundo leva em consideração a prioridade das requisições, como critério de QoS. Já o terceiro possui algoritmo simples FCFS para reserva, entretanto, oferece também mecanismos de alocação, descoberta e integração de recursos heterogêneos. No trabalho mostrado em [Vahdat-Nejad, Monsefi e Naghibzadeh 2007] é falado sobre a necessidade de informações atualizadas sobre o estado da rede nos clusters, mas não é descrito como garante essa atualização de informação.

O trabalho explicado em [Qin e Bauer 2009] não apresenta reserva de recursos, mas possui a vantagem sobre os outros da co-alocação de recursos entre clusters. Desta forma, este trabalho favorece jobs maiores, com necessidade de muito processamento, e também evita espera longa nas filas de clusters para alguns jobs, visto que pode compartilhar os recursos entre clusters.

Tabela 3.2: Comparação entre os trabalhos relacionados

	Silva e Dantas 2007	Qin e Bauer 2009	
Integra OVs	X	-	
Atualização	estático	dinâmico	
Co-Alocação	-	X	
Reserva Antecipada	-	-	
Tipos de Recursos	vários	banda e nºde processadores	

	Vahdat-Nejadte al. 2007	Liu, Dai e Chuang 2006	Sulistio e Buyya 2004
Integra OVs	-	-	X
Atualização	dinâmico	dinâmico	dinâmico
Co-Alocação	-	-	-
Reserva Antecipada	X	X	X
Tipo de Recursos	banda e nºde processadores	banda	vários

Capítulo 4

Proposta e Implementação da Arquitetura

Um ambiente de multi-clusters coordenados e compartilhados que permitam solucionar problemas dinâmicos, envolvendo diferentes organizações, pode ser considerado como uma grade computacional. Uma aplicação usualmente é composta por grupos de processos, tanto dependentes quanto independentes. Esses processos podem ter diferentes funções e serem paralelamente distribuídos em um sistema de grade. O compartilhamento de recursos possibilita uma otimização no nível de utilização dos mesmos, melhorando os tempos de resposta para as aplicações.

Este trabalho apresenta o GRADI, uma proposta de arquitetura para co-reserva de recursos antecipada para ambientes multi-clusters configurados como uma grade. Visa melhorar o nível de utilização dos recursos e o tempo de resposta total do sistema. O gerenciador de recursos proposto possui sistema de descrição e seleção de recursos, e utiliza ontologias para a associação de recursos entre sistemas heterogêneos. Além disso, apresenta uma abordagem dinâmica para reservas imediatas e antecipadas, e um sistema de co-alocação baseado em lógica difusa. O mecanismo desenvolvido oferece suporte a tarefas que necessitem de um nível de processamento muito alto, o qual não pode ser suportado por um único cluster. Com esse objetivo é utilizada uma interface de seleção e co-reserva global, a qual permite a solicitação para clusters locais de um nodo remoto.

Para desenvolvimento do protótipo, foram realizadas algumas etapas de pesquisa. Primeiramente foi realizado um estudo comparativo entre diferentes gerenciadores de recursos para grades, para formar base de conhecimento sobre as pesquisas e desafios na área. Este trabalho inicial pode ser visto em [Ferreira et al. 2008].

Em uma segunda etapa, foi desenvolvida uma extensão para que o trabalho de [Silva e Dantas 2007], o qual aborda seleção estática de recursos em grades, oferecesse suporte também a alocação dinâmica de recursos. Desta forma, há maiores garantias de disponibilidade dos recursos em tempo de execução para o cliente. Também foi acrescentado um método de alocação para co-alocação de recursos, baseado na lógica difusa do tra-

balho de [Qin e Bauer 2007]. Por fim, foi desenvolvido um ambiente de execução em simulador, como ambiente de teste para o protótipo. Essa etapa gerou os artigos [Janson et al. 2009] e [Ferreira et al. 2009].

Na terceira etapa, foi desenvolvido um método de reserva antecipada e co-reserva de recursos, baseado na reserva de links e processadores, gerando a publicação [Janson, Dantas e Bauer 2010]. Com este módulo, aumentam as garantias de disponibilidade dos recursos no sistema no momento da execução. Além disso, é possível alcançar melhores resultados de utilização dos recursos disponíveis no sistema.

O protótipo utiliza uma interface gráfica de fácil manipulação para usuários comuns, para seleção e reserva dos recursos. Nas próximas seções serão explicados o desenvolvimento da arquitetura e implementação para o protótipo durante as etapas 2 e 3. Os resultados obtidos serão mostrados no próximo capítulo.

4.1 Arquitetura

No GRADI é assumido que quando uma aplicação envia uma tarefa para ser executada em uma grade, essa tarefa é composta de *n* processos. O foco da arquitetura desenvolvida é propor uma abordagem dinâmica de reserva de recursos para configurações multi-clusters. O mecanismo desenvolvido oferece suporte a tarefas que necessitem de um nível de processamento muito alto, o qual não pode ser suportado por um único cluster. Com esse objetivo é utilizada uma interface de seleção e co-reserva global, a qual permite a solicitação direta, de um nodo remoto, e negociação de reservas com os gerenciadores locais dos clusters compartilhados na grade.

A figura 4.1 mostra os principais componentes do ambiente proposto. No lado do cliente há uma interface interativa, a qual recebe informações sobre restrições estabelecidas por um usuário. Essas restrições são relacionadas aos tipos e quantidades de recursos necessárias para a execução de uma aplicação. Quando enviada, a requisição passa pelo servidor, o qual possui informações semânticas estáticas a respeito dos recursos de cada cluster do ambiente. O *Selector* é responsável por fazer a comparação da requisição com as informações estáticas armazenadas. Todos os cenários possíveis para responder a requisição são encontrados nas informações estáticas.

Posteriormente, o *AR_Manager*, o qual representa a principal contribuição deste trabalho, compara os resultados com as reservas já realizadas, procurando filtrar resultados que causem conflitos. Desta forma é definido o ambiente que será reservado para a requisição. Entretanto, dependendo da quantidade de processadores requisitados, pode ser que não

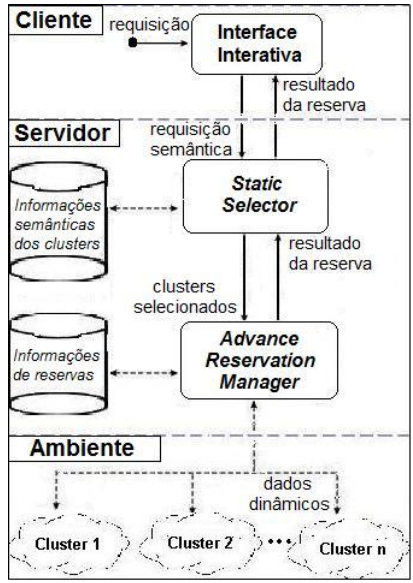

Figura 4.1: Principais componentes.

haja um cluster que consiga atender a essa requisição. Neste caso, são realizadas reservas distribuídas entre clusters, controladas por um algoritmo baseado em lógica difusa, o qual será detalhado posteriormente.

Para que se evitem conflitos de interesse entre requisições globais e locais, o *AR_Manager* confere dinamicamente com o ambiente a disponibilidade dos recursos selecionados no momento da reserva. A seguir, a arquitetura serão detalhados os principais componentes do Servidor.

4.1.1 Arquitetura do Servidor

Na parte do servidor está a principal contribuição deste trabalho. É o servidor que decide o escalonamento das reservas, segundo as requisições dos usuários. A Figura 4.2 mostra a arquitetura do servidor proposta, explicita as principais funções e a interação com o ambiente.

Matchmaker Semântico Aplica uma abordagem para integração de recursos em sistemas com múltiplas ontologias, baseado em [Silva e Dantas 2007]. Este módulo tem acesso a informações estáticas armazenadas sobre os recursos de cada OV que estão compartilhados no ambiente e também às diferentes formas de descrição destes nas

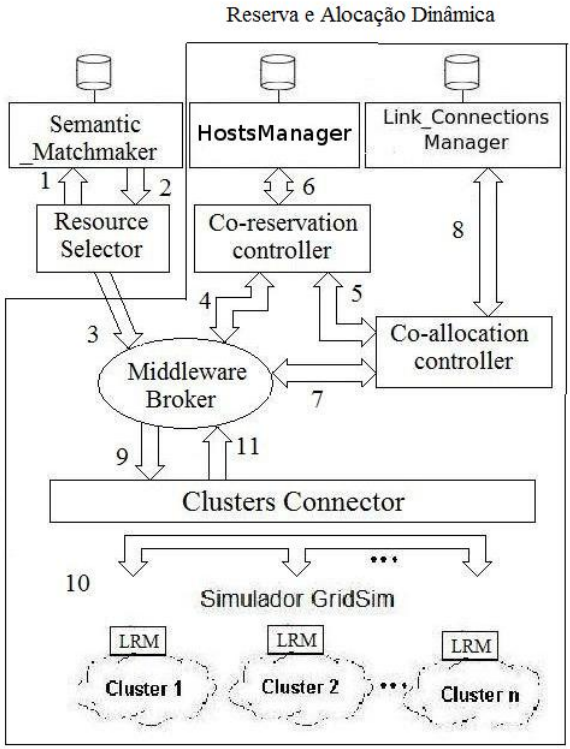

Figura 4.2: Arquitetura proposta no GRADI.

organizações. Com este módulo é possível obter informações sobre estes recursos de forma transparente com relação à heterogeneidade das descrições.

Reservations Manager Neste módulo está o algoritmo de reserva antecipada. Tem acesso às estruturas que guardam todas as reservas globais realizadas em cada cluster do ambiente, sendo responsável por diminuir a fragmentação do uso dos recursos no tempo, além de informar colisões, ambiguidades e sobrecarga dos sistemas.

Link_Connections Manager Este módulo tem acesso a informações estáticas sobre a carga dos links entre clusters reservados e livres. Possui informações sobre as conexões e o nível de utilização de banda

entre os clusters. É responsável por evitar a sobrecarga destes links, e oferecer alternativas de conexão.

Resource Selector Recebe as requisições de reserva do cliente remoto e utiliza as informações do *Matchmaker Semântico* para fazer uma primeira seleção de recursos. Os recursos selecionados dependem de parâmetros da requisição, tal como, tipo do sistema operacional, a arquitetura do processador, sua freqüência, quantidade de memória.

Co-Reservation Controller Possui o algoritmo de co-reserva dos recursos. Administra e negocia de forma dinâmica com os gerenciadores locais como os recursos serão reservados simultaneamente, evitando ao máximo a sub-utilização do sistema global como um todo. Verifica também se há necessidade de co-alocação dos recursos.

Co-Allocation Controller Possui o algoritmo de co-alocação dos recursos. Neste, é necessário fazer controle da banda utilizada pelos processos entre clusters. Além disso, faz a parte de verificação e adaptação dinamica dos recursos selecionados pelo módulo *Selector* para serem alocados de acordo com a disponibilidade nos sistemas reais locais.

Clusters Connector Estrutura que mantém transparente a comunicação do middleware do servidor com os gerenciadores locais de cada cluster.

LRM Gerenciador de recursos local (*Local Resources Manager*) desenvolvido para que cada cluster receba as requisições de reserva globais, realizando a alocação e reserva antecipada local no ambiente simulado. Permite também requisições locais, e faz a verificação dos recursos disponíveis, estas requisições de acordo com a disponibilidade do sistema local.

Middleware Broker estrutura central que atua como um broker genérico, exercendo o meta-escalonamento no topo do sistema de grade. Gerencia e acompanha as tarefas globais, fazendo a integração entre os módulos de seleção, co-alocação e co-reserva. Ou seja, possui o algoritmo de decisão para cada etapa da execução das requisições.

A decisão de execução geral das requisições no sistema segue respectivamente as seguinte etapas, de acordo o indicado na figura 4.2.

1. O *Resource Selector* recebe a requisição de reserva do cliente, e envia ao *Matchmaker Semântico*.

2. O *Matchmaker Semântico* faz a integração da descrição dos parâmetros pedidos com as descrições existente no sistema e verifica os recursos compartilhados descritos existentes, retornando ao *Resource Selector*.

3. A seleção dos recursos passa pela primeira filtragem, em que O *Resource Selector* verifica estaticamente quais clusters podem atender aos parâmetros pedidos. Estas respostas são enviadas ao *Middleware Broker*, o qual verifica a necessidade de co-reserva (**4**) ou co-alocação (**7**) de recursos.

4. Precisando de co-reserva, o *Co-Reservation Controller* consulta o *Co-Reservation Controller*.

5. Precisando de co-alocação, o *Co-Reservation Controller* consulta o *Co-Allocation Controller*.

6. É feita a decisão de reserva, baseado no estado do sistema de reservas armazenado, e o resultado retorna ao *Middleware Broker*.

7. Tratando-se de uma reserva imediata que necessita de co-alocação, o *Middleware Broker* consulta o *Co-Allocation Controller*.

8. O *Co-Allocation Controller* consegue verificar o estado global de reserva dos links entre clusters para decidir como os processos serão distribuídos.

9. O *Middleware Broker* utiliza o *Clusters Connector* para verificar a disponibilidade dos recursos de forma dinâmica no ambiente.

10. Comunicação remota do middleware com cada cluster local selecionado para reserva.

11. O *Clusters Connector* retorna ao *Middleware Broker* a resposta do cluster local. Caso a reserva tenha sida realizada com sucesso, os recursos de processamento e banda utilizados são armazenados pelos *Reservations Manager* e *Connections Manager*

4.1.2 Co-Alocação

O job submetido a um gerenciador de recursos em grades deve trazer consigo a descrição dos recursos necessários para execução, tal como quantidade de processadores, memória, armazenamento necessário, etc. A ordem de execução desses jobs depende de uma série de fatores adotados pelo escalonador, entre estes a configuração do ambiente, prioridade dos

usuários, disponibilidade dos recursos, e é baseado em algoritmos de escalonamento. Para que possam ser executados, determinados recursos serão alocados para o job, de acordo com os requisitos pedidos pelo cliente e também dependendo do estado do sistema com relação a carga de trabalho nos recursos. Nesta primeira parte do desenvolvimento do trabalho, foram focadas a descrição e alocação dinâmica de recursos em grades computacionais.

O protótipo foi desenvolvido para configurações de grades com multi-clusters. Desta forma, há diferentes organizações que descrevem seus próprios recursos. Estes devem ser associados por similaridades de descrição no middleware, para haver um compartilhamento global transparente ao usuário final. Foi feita descrição semântica dos recursos utilizando ontologias, e o método de associação é uma extensão do trabalho de [Silva e Dantas 2007].

O trabalho de [Silva e Dantas 2007] mostra uma ferramenta para integração semântica de múltiplas ontologias, levando em consideração a interação humana, como visto detalhadamente no capítulo 3. É proposto uma expansão das requisições, oferecendo maiores informações sobre os ambiente de grade. Assim, é possível construir as configurações de requisição para execução das aplicações considerando as configurações multicluster disponíveis.

Entretanto, todas as operações realizadas em [Silva e Dantas 2007] consideram unicamente informações estáticas, ou seja, não são consideradas mudanças que possam ocorrer das informações. É importante ressaltar que na prática o uso dos recursos não acontece de forma estática. A carga dos recursos e a disponibilidade dos mesmos no sistema se alteram com frequência.

Um segundo problema relacionado à abordagem de [Silva e Dantas 2007] é que o tamanho limite do job em quantidade de processamento é limitado ao tamanho do maior cluster do sistema (o que contém maior número de processadores). Ou seja, só seleciona um cluster por job e, assim, alguns jobs maiores não conseguem executar ou precisam esperar muito tempo na fila até que se libere a quantidade necessária de recursos em algum cluster. Por outro lado, há processadores suficientes ociosos no ambiente se for levar em consideração a soma dos recursos dos múltiplos clusters na grade. Além disso, não é considerada a carga na comunicação entre clusters, importante para maximizar a comunicação entre processos.

Uma forma simples de aprimorar a execução de aplicações paralelas e distribuídas no trabalho de [Silva e Dantas 2007] é arrecadar dinamicamente informações de desempenho do sistema, sobre os recursos utilizados. O Trabalho de [Qin e Bauer 2009] apresenta exatamente um

método de alocação dinâmica que pode completar [Silva e Dantas 2007]. Utilizando os algoritmos da lógica difusa proposto em [Qin e Bauer 2009], a qual é detalhada no capítulo 3, é possível monitorar e arrecadar informações sobre processadores dos clusters e links de comunicação entre clusters. Como resultado dessa junção, é esperado que se tenha melhores cenários de alocação, visto que serão utilizadas informações mais precisas, as quais facilitam a formação de melhores configurações para a execução das aplicações.

Assim sendo, neste trabalho, na parte de alocação dinâmica dos recursos, foi realizada uma extensão do algoritmos de [Silva e Dantas 2007]. As requisições são, então submetidas a um novo módulo de alocação, o qual possui um algoritmo para decisão de alocação, baseado em informações dinâmicas do sistema, e na lógica difusa de [Qin e Bauer 2009]. Desta forma, é utilizada toda a vasta informação estática semântica do ambiente oferecida pelo middleware, além de se ter informações atualizadas relacionadas a carga dos sistemas para alocação.

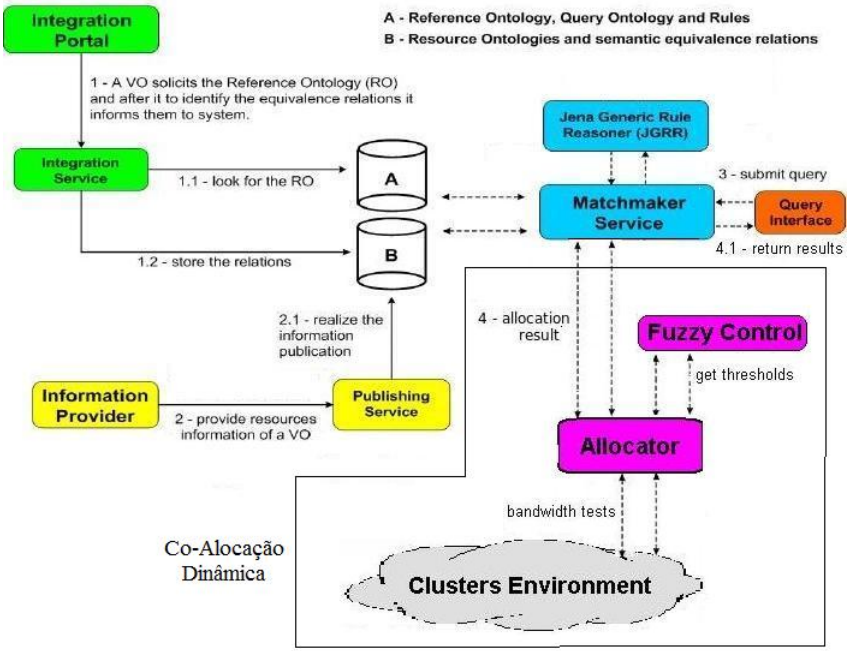

Figura 4.3: Arquitetura extendida para co-alocação dinâmica.

A figura 4.3 mostra a arquitetura de seleção de recursos extendida para suportar co-alocação dinâmica de recursos, com os componentes adicionados na nova arquitetura proposta e as principais modificações na comunicação entre estes módulos. No lado do cliente, há uma interface interativa, a qual recebe as restrições de configuração exigidas pelo usuário na requisição. Essas informações são traduzidas em dados semânticos e enviadas ao servidor, o qual possui informações estáticas sobre os recursos de cada cluster do ambiente. O servidor associa as informações da requisição com as informações sobre os recursos dos clusters. Todas os possíveis resultados para a requisição são pesquisados nos dados armazenados, exceto as informações sobre quantidade de processamento, o qual é deixado para o módulo de alocação verificar dinamicamente.

O módulo de alocação analisa e compara os resultados do *Matchmaker*, baseado nas características requisitadas para o job e na situação atual de cada sistema pedido. O ambiente é acessado diretamente por este módulo para que se obtenha dados dinâmicos sobre a disponibilidade dos recursos nos clusters e de caraga dos links entre estes clusters. Dependendo do resultado desta consulta, o *Allocator* determina como vai ser realizada a alocação para o job. Além disso, verifica se os processos precisam ser distribuídos por mais de um cluster, utilizando o módulo de *Fuzzy Control* para decidir uma forma adequada para esta distribuição. O resultado calculado é retornado ao usuário, como uma possível forma de alocação mais adequada de acordo com as configurações pedidas para o job.

4.1.3 Co-Reserva

A simples alocação dos recursos não oferece garantias de que a aplicação executará exatamente no momento requisitado. Em ambientes com concorrência de recursos entre usuários, pode ocorrer dos recursos estarem ocupados, ou outros jobs possuírem prioridades maiores. Neste caso, pode acontecer do usuário ter que esperar para execução de sua aplicação ou então executá-la com uma qualidade de serviço (QoS - *Quality of Service*) baixa. Nos caso de aplicações críticas ou multimídias, por exemplo, é necessário oferecer maiores garantia de execução e nível de serviço. Entretanto, ainda não há um acordo padrão que sirva de modelo para middlewares em ambientes complexos como as grades, devido à dinamicidade destes sistemas.

Além disso, o acesso simultâneo a múltiplos recursos computacionais (co-alocação) deve considerar algumas restrições tal como dependência temporal entre estes recursos. Neste contexto, estratégias para reservas antecipadas de recursos auxiliam a garantir esta relação temporal no momento da execução da aplicação. Além disso tem sido estudadas como um

meio de oferecer QoS com relação à disponibilidade dos recursos, para evitar imprevistos ou baixa qualidade do serviço. Desta forma, para que o usuário tenha acesso aos recursos é importante que seja feito um acordo de nível de serviço (SLA – *Service Level Agreement*) com o provedor. Este acordo define o tempo, quantidade de processamento e outros parâmetros de QoS, a um determinado custo, definindo a prioridade do usuário no sistema [Netto, Bubendorfer e Buyya 2007].

A reserva antecipada está diretamente relacionada com esses objetivos de QoS, pois auxilia a garantir a disponibilidade dos recursos, estimar tempo de resposta e alcançar deadlines. O usuário pode especificar um determinado nível de serviço para sua aplicação, o qual é mapeado para requisição de recursos. Relembrando que estas requisições podem ter dependências temporais entre si. Assim, esta etapa do trabalho foca na reserva antecipada de recursos como um meio de oferecer disponibilidade dos recursos no momento da execução, co-alocação e, ao mesmo tempo, se adequar à imprevisibilidade das grades.

Um dos grandes desafios para reserva de recursos em grades é administrar os recursos dos sistemas autônomos locais (no caso, gerenciadores dos clusters). Em [Gehring e Preiss 1999] são estudados cenários em que os escalonadores locais não cooperam com o escalonador global. Entretanto, lidar com essas negociações entre protocolos não é foco deste trabalho. O uso de reservas antecipadas pode ser uma forma elegante de evitar estas limitações, visto que permite delegar decisões de escalonamento para gerenciadores locais, verificando localmente a possibilidade de reserva.

Um segundo desafio está na imprevisibilidade dos ambientes de grades. Quando uma reserva antecipada é aceita no sistema, é esperado que o usuário tenha acesso aos recursos que reservou em um determinado momento específico. Entretanto, podem ocorrer mudanças nas filas até entre o momento da aceitação da reserva e o momento da execução em si do job. Isto pode ocorrer por vários motivos, entre estes, usuários que cancelam ou modificam suas requisições, falha dos recursos requisitados, erros na estimativa do tempo de uso dos recursos. Para minimizar este problema, o middleware desenvolvido monitora dinamicamente a possibilidade e disponibilidade local dos recursos no momento da reserva e no também no momento da execução, podendo ser pedido o reescalonamento do job, caso necessário.

Visto as vantagens proporcionadas pela reserva de recursos, foram adicionados à arquitetura mostrada na figura 4.3, componentes para co-reserva de recursos. Para melhor entendimento da estrutura proposta, na figura 4.4 é mostrada a arquitetura de co-reserva em camadas.

A arquitetura de reserva antecipada de recursos aplicada neste tra-

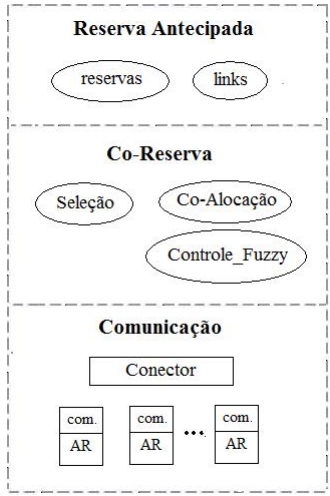

Figura 4.4: Arquitetura de co-reserva em camadas.

balho possibilita reservas concorrentes de diferentes tipos de recursos em uma configuração de grade com múltiplos clusters. Também permite o planejamento de execução do job, no qual componentes de processamento e banda podem ter relação de dependência temporal e espacial. Foi adotado um mecanismo de co-reserva de recursos que oferece algumas garantias de execução, aumentando a eficácia.

A camada de *Co-Reserva* utiliza o mecanismo de *matching semântico* explicado anteriormente para uma seleção inicial dos recursos. Posteriormente, estes recursos selecionados passam por uma série de filtragens. É verificado dinamicamente no sistema a disponibilidade dos recursos selecionados; se necessário, é realizado o algoritmo de co-alocação; a forma de distribuíção dos processos é baseada em lógica difusa para controle dos níveis de saturação dos links entre clusters e para controle do desempenho na comunicação entre processos; A decisão de reserva é feita automaticamente e de acordo com os recursos selecionados que atendam as requisições do usuário e ao mesmo tempo maximizem o nível de utilização dos recursos do sistema.

Na camada de *Reserva Antecipada* é feita a decisão de cada reserva unitária, tanto de itens de processamento quanto de banda. É nesta camada que estão todas as informações armazenadas sobre o ambiente, e também é nela que são executadas em si as reservas globais e também a dependência

entre estas reservas, ou seja, processadores e banda reservados para um mesmo job e com dependências físicas/temporais. Cabe a este módulo decidir se uma reserva pode ser realizada ou não, e oferecer possibilidades de reserva que maximizem o nível de utilização dos recursos. Isso de acordo com os períodos ociosos do sistema e possíveis fragmentações.

Por fim, a camada de *Comunicação* representa o diálogo do middleware com os gerenciadores de recursos locais. Em cada cluster local há um módulo de reserva de recursos com diferentes algoritmos de reserva, de acordo com a política adotada por cada administrador dos clusters. Estes módulos recebem mensagens padrões de requisição do middleware e executam a reserva localmente, ou tentam adaptar a requisição às possibilidades locais.

4.2 Implementação do GRADI

Nesta seção serão mostrados detalhes da implementação dos principais componentes e funções da reserva antecipada de recursos do GRADI. Primeiramente será passada uma visão geral da implementação da arquitetura e do *Broker*, o qual gerencia os módulos e requisições. Em seguida serão explicados o desenvolvimento dos algoritmos de reserva e co-reserva de recursos. Por fim, será mostrada a interface do servidor com o ambiente, no caso o simulador GridSim.

A figura 4.2 da seção de arquitetura apresenta os principais componentes formadores do GRADI. Entre estes estão o *Selector*, o *Broker*, o *ReservationController* e o *HostsManager*. Para ilustrar a implementação geral destas estruturas e sua correlação, é mostrado na figura 4.5 o diagrama de atividades de uma reserva de recursos simples, ou seja, sem co-alocação. Este diagrama mostra apenas as atividades gerais em algumas das estruturas que participam de uma reserva. Ao longo de cada subseção serão explicadas mais detalhadamente as atividades mostradas no diagrama, e serão apresentadas mais algumas estruturas.

4.2.1 Selector

O módulo *Selector* permite que um usuário decida que tipo de recursos do cluster são requisitos para a execução de uma tarefa específica. São oferecidas várias características para servirem de parâmetros (como tipo de sistema operacional e quantidade mínima de memória), os quais são utilizados como filtros no processo de seleção semântica. É importante observar que a quantidade de processadores não é avaliada neste momento, permitindo a seleção de ambientes com uma quantidade menor de processadores do que o requisitado pelo usuário. Desta forma há maior flexibilidade na escolha de ambiente de execução caso haja necessidade de co-alocação do job posteriormente.

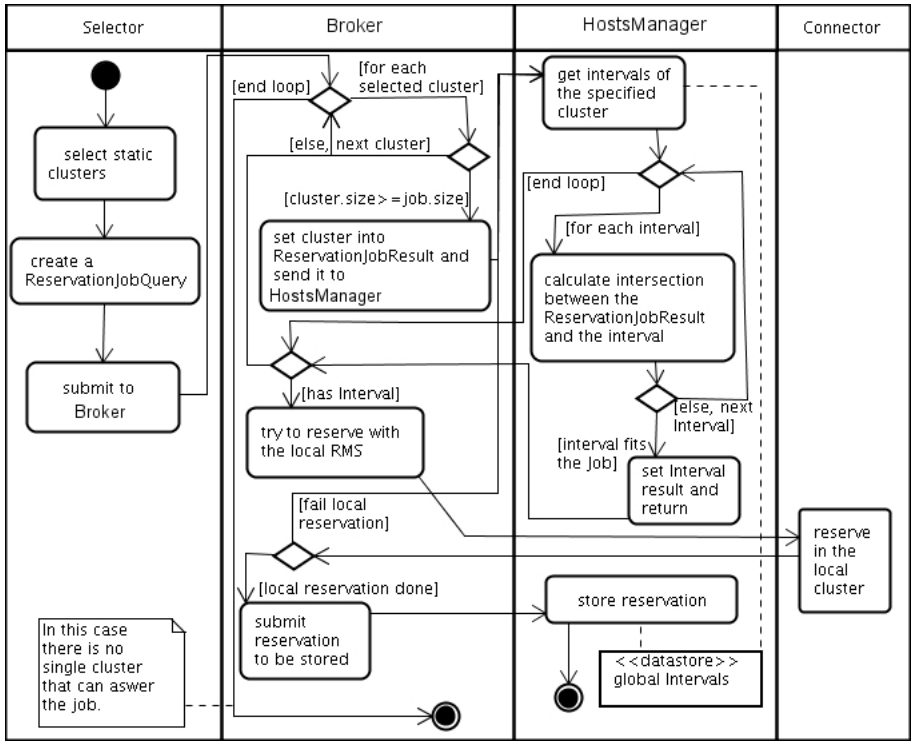

Figura 4.5: Diagrama de atividades na reserva de recursos.

No término do processo de seleção, apenas os clusters que correspondem à todos requisitos (exceto sobre quantidade de processadores) permanecem como possível resposta. Posteriormente, estes clusters passam por uma filtragem final realizada pelo *Broker*, o qual decide o possível cenário ideal de alocação.

Os parâmetros que compõem uma requisição no GRADI podem ser subdivididos em dois conjuntos:

Requisitos de ambiente: são requisitos que descrevem as características de configuração do ambiente necessários para execução do job. Estes requisitos são avaliados pelo *Selector*, o qual compara os dados semânticos armazenados sobre as organizações que compartilham recursos, como descrito na arquitetura.

Requisitos de reserva: são os requisitos necessários para realização de uma reserva única ou co-reserva de recursos, associando a correlação entre reservas no tempo. Esta parte é avaliada dinamicamente e considera a quantidade de processadores necessária, a qual é referida como *o tamanho do job*; a disponibilidade dos clusters selecionados; e uma estimativa da largura de banda necessária entre os clusters. Além disso, também possui três tipos de parâmetros de tempo: o *start time*, ou seja, o momento em que o job está pronto para ser executado; um *deadline*, que sinaliza um prazo final para o fim da execução; e a duração da reserva do recurso para o job.

Após o *Selector* gerar os resultados estáticos, esse passa o resultado para o *Broker*, juntamente com a requisição inicial. Por fim, na segunda fase, o *Broker* gera a reserva final realizada no sistema, como resultado para a requisição. Este resultado é composto pelo IP do cluster a reservado; a quantidade de processadores; e o intervalo de tempo reservado. Este intervalo é composto do momento de início da reserva e a sua duração. A seguir será detalhada a implementação do *Broker*.

4.2.2 Broker

O *Broker* tem função de gerenciamento das requisições no servidor, além de tomar decisões no controle das reservas e execuções dos processos. Cada requisição que chega do *Selector* é tratada por um processo separado, o qual é analisado e encaminhado de acordo com as decisões do algoritmo do *Broker*, detalhado na figura 4.1. Este algoritmo mostra a estratégia de reserva na função *dealJob()* do *Broker*, utilizada em cada requisição.

Antes de introduzir a estratégia é necessário explicar alguns termos e notações utilizadas na descrição do algoritmo:

- Q denota a requisição do usuário (*Query*), com os requisitos de tamanho do job $jobSize$, banda bw e as variáveis de tempo (startTime, duração, deadline).
- Os clusters selecionados primeiramente pelo *Selector*, estão definidos como $C_1, C_2, \ldots, C_n$.
- Em $N = (C_1, C_2, \ldots, C_n)$, é mostrado o conjunto de clusters selecionados como resposta pelo *Selector*, e faz parte da entrada do algoritmo. Além disso, N está organizado em ordem crescente com relação ao P_i, considerando $1 \leq i \leq n$.
- P_i representa a quantidade de processadores disponíveis em um cluster C_i.

- Cada objeto de reserva de recurso criado de acordo com as requisições são definidos por R_1, R_2, ... , R_n.

- $M = (C_1, C_2, \ldots, C_n)$ representa o conjunto de reservas para a requisição Q, e é a saída do algoritmo.

- rc é uma instância do $ReservationController$.

- c é uma instância do $Connector$.

A estratégia do *Broker* pode ser dividida em duas etapas, na primeira tenta-se reservar um único cluster para a requisição. Não conseguindo, é tentada a distribuição dos processos entre clusters. Na primeira etapa, ao receber uma requisição, o algoritmo verifica a disponibilidade dos recursos para a requisição de reserva de acordo com a atual situação do ambiente, por meio das informações de reservas realizadas armazenadas no servidor. Esta verificação ocorre chamando a função *checkReservation*, do módulo de reserva de recursos *ReservationController*. Neste caso, são considerados apenas os clusters onde $P_i \geq jobSize$, os quais são separados em L.

A possibilidade de reserva mais adequada para a requisição é retornada pela função, e então tenta-se reservar o recurso remoto, com o gerenciador de recursos local do cluster, $rq \rightarrow reserve(R_1)$. Após confirmada a reserva local, são atualizados no servidor global as estruturas com informações do sistema, como reservas e intervalos livres, que serão mais detalhados na próxima subseção.

Caso haja algum conflito na reserva local, e esta não possa ser realizada, o algoritmo faz uma chamada recursiva apenas com os clusters que ainda não foram pesquisados. Se percorrendo todos os clusters selecionados não for possível realizar uma reserva com um único cluster, é tentada a distribuição dos processos entre os clusters selecionados pelo *Selector*, $rq \rightarrow split(Q, C)$. Caso tenha êxito, são feitas atualizações também na estruturas do servidor global, considerando o uso de banda nas conexões entre clusters.

4.2.3 Reserva antecipada

Quando o *Broker* faz a primeira verificação sobre a possibilidade de reserva, é chamada a função *checkReservation* do módulo ReservationController. Esta primeira parte do algoritmo acessa as informações locais sobre reservas anteriormente realizadas para verificar a possibilidade de efetivação da reserva requisitada. Para isso, são verificados a os intervalos livres de tempo dos processadores dos clusters. A figura 4.6 mostra o

Algorithm 4.1 Decisão de reserva do $Broker$: $dealJob()$

Input: Uma requisição de reserva Q, com o tamanho do job $jobSize$ e as variáveis de tempo ($startTime, dur, deadline$); o conjunto de clusters selecionados $N = (C_1, C_2, ..., C_n)$ em ordem crescente de P_i.

Output: $M = (R_1, R_2, ..., R_n)$, correspondendo a cada reserva realizada no sistema

1: **boolean** $sucess \leftarrow$ **False**
2: $rSize \leftarrow$ tamanho de R
// L é usado para pesquisa de reserva simples
3: **if** $L = null$ **then**
4: **for all** C_i tal que $P_i \geq jobSize$ **do**
5: add C_i to L
6: **end for**
7: add L to Q
8: **end if**
9: $R \leftarrow checkReservation(Q, L)$
// Verifica se há necessidade de co-alocação
10: **if** $rSize \geq 0$ **then**
11: $success \leftarrow reserve(R_1)$ // Tenta reserva local
// Armazenamento global
12: **if** $success =$ **True then**
13: $storeReservations(M)$
14: $calcIntervals(M)$
15: **else**
16: $dealJob(Q, L)$ // recursividade
17: **end if**
// Reserva com co-alocação
18: **else**
19: $success \leftarrow split(Q, C)$
// Armazenamento global
20: **if** $success =$ **True then**
21: $storeReservations(M)$
22: $storeConnections(M)$
23: $calcIntervals(M)$
24: **end if**
25: **end if**
26: **Return** M

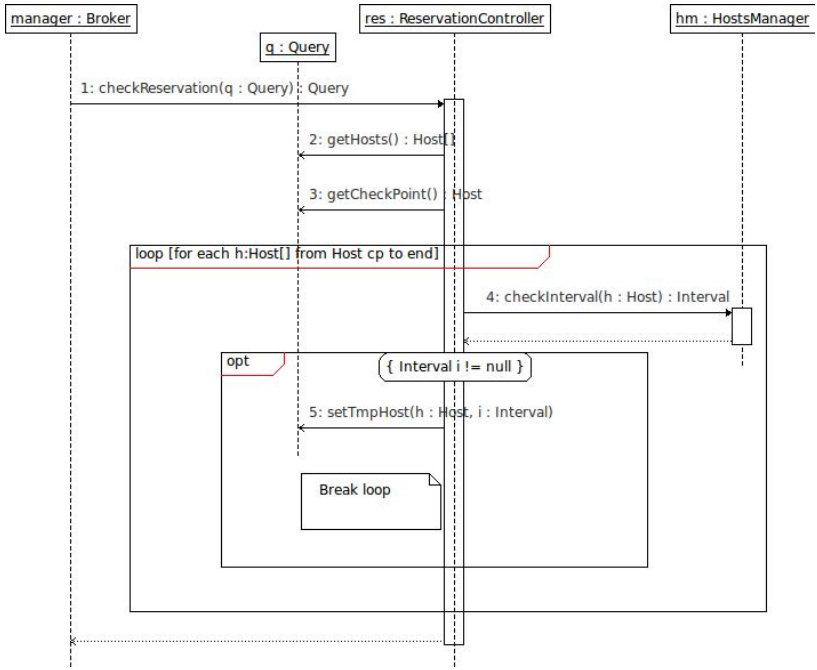

Figura 4.6: Verificação de disponibilidade para reserva

diagrama de sequência dessa verificação estática, destacando a interação entre os principais componentes neste processo.

O *ReservationController* recebe a requisição com os possíveis clusters que podem atendê-la sozinhos. Estes, estão organizados em ordem decrescente com relação a quantidade de processadores. Desta forma, quando a lista de clusters for percorrida, o cluster com menor capacidade de processamento que possa atender o job será selecionado, deixando os clusters maiores livres para que futuros jobs com maior requisição evitem comunicação entre clusters e *overhead* de rede.

A disponibilidade de cada *host* é testada, até que se encontre um que esteja disponível para reserva dentro do intervalo de tempo indicado na requisição. Quando este é encontrado, é armazenado na *Query*, que é o objeto retornado. O *Broker*, então, tenta fazer a reserva remota no cluster local. Caso haja conflitos e esta reserva não possa ser realizada, o *Broker* chama novamente esta função. Assim, o *host* que não pode ser reservado é utilizado como *checkpoint* para a próxima busca.

Para verificar a disponibilidade do cluster é necessário ter acesso a algumas informações sobre o estado do sistema, como as reservas realizadas, intervalos de tempo dos processadores ocupados. Assim, o *ReservationController* chama o *HostsManager*, o qual gerencia uma série de estruturas de acesso a informações. Este módulo tem acesso a informações sobre as reservas globais realizadas, aos intervalos livres de cada cluster e também a carga na banda dos links entre clusters nos intervalos de tempo. Este último será mais detalhado posteriormente.

Um intervalo é adequado a um job quando $max(sI, sJ)$-$min(dI, dJ)$ $\geq durJ$, onde *sI* corresponde ao início do intervalo, *sI* é o *start time* da tarefa, *dI* corresponde ao final do intervalo, *dJ* é o *deadline* da tarefa, e *durJ* corresponde à duração da tarefa. Este algoritmo percorre os intervalos de ociosidade dos processadores do cluster. O intervalo com menor duração que corresponda aos requisitos da tarefa (inclusive de tempo) é selecionado para alocação. Os intervalos maiores são deixados para possíveis jobs futuros que necessitem de maior tempo de execução, oferecendo maior chance de serem alocadas em um único cluster.

O intervalo encontrado é associado ao resultado e o *Broker* passa para a interface de conexão com os clusters, a qual envia remotamente estes resultados para o cluster local escolhido. Se o cluster puder realizar a reserva por inteira, ou seja, com todos os recursos pedidos e dentro de algum intervalo compatível, a reserva local é realizada e o intervalo reservado é retornado ao servidor. O intervalo decidido estaticamente no servidor nem sempre é o mesmo que vai ser efetivamente reservado. Entretanto, é importante verificar estaticamente a disponibilidade de intervalos ociosos antes tentar uma reserva local, pois ganha-se tempo evitando comunicação com clusters que não poderão atender à requisição.

Por fim, o resultado da reserva é armazenado por meio do *HostsManager*, e os intervalos ociosos da estrutura são atualizados. A cada alteração nas reservas são feitas atualizações no gerenciamento geral dos recursos para garantir informações dos intervalos disponíveis de cada cluster. Os períodos ociosos são armazenados em ordem de duração, para que as pesquisas na estrutura retornem o menor intervalo apropriado à requisição. Este comportamento reduz a fragmentação e beneficia tarefas que requerem muito processamento, deixando livres os intervalos maiores.

Se ao final da pesquisa não for encontrado um cluster que consiga responder sozinho a requisição, então o *Broker* tenta distribuir os processos da tarefa por mais de um cluster, como é detalhado a seguir.

4.2.4 Co-Reserva

Na de co-reserva de recursos do GRADI, os processos de uma tarefa são distribuídos entre alguns dos clusters escolhidos pelo *Selector*. O controle dessa distribuição é realizado por um método empregando lógica difusa, baseado em [Qin e Bauer 2007]. Este algoritmo gera dois valores percentuais:

δ - depende da quantidade de processadores e do tipo de comunicação entre os clusters da OV. Este valor é usado no controle da distribuição dos processos, evitando excesso de comunicação entre clusters, que poderia aumentar significantemente o custo da execução da tarefa;

λ - depende dos mesmos parâmetros de δ, além da banda utilizada na comunicação entre os processos e da largura de banda máxima entre cada cluster selecionado para alocação. Este valor é usado no controle da saturação das conexões entre clusters. Quando uma conexão está saturada, não pode ser utilizada.

A figura 4.7 mostra como ocorre a decisão sobre a distribuíção dos processos entre clusters. Cada cluster selecionado inicialmente pelo *Selector* é testado pelo componente *CoAllocationController*, para verificação de diponibilidade mínima de processamento. Os clusters que passam nesse teste vão sendo somados até que se alcance a quantidade de processadores necessária. Para controlar a distribuição dos processos, pelo menos um dos clusters deve obedecer à expressão $c1Size \geq \delta * jobSize$, onde *c1* é o cluster, e *c1Size* a quantidade de processadores deste. Se o cluster atende à primeira condição de controle, então é feita uma busca por um intervalo ocioso (*c1i*) para a tarefa no cluster *c1*.

O *Connections_DB_Manager* faz uso de estruturas que mapeiam as conexões entre os clusters. Uma conexão é composta pelos IPs de ambos os clusters, a largura de banda máxima, e a carga atual. Esta carga é atualizada cada vez que uma reserva é feita, ou quando o recurso é liberado. Assim, se uma reserva requer comunicação entre clusters, ela deve estimar quanto da banda será utilizado.

Em uma próxima fase, são verificados todos os clusters que possuem conexão direta ao *c1*. Esta é uma função que utilize o *Connections_DB_Manager*, o qual possui acesso a informação prévia sobre todas as conexões entre clusters do sistema. Esses clusters devem passar pelo controle de saturação da conexão, o qual define que as conexões entre *c1* e *c2* devem seguir a regra $workload/maxBW \geq \lambda$. Neste caso, *workload* é a carga de banda utilizada na comunicação entre processos e *maxBW* é a largura de banda máxima entre *c1* e *c2*. Como resultado, se a conexão estiver saturada, *c2* não é adicionado na lista de clusters a receber processos.

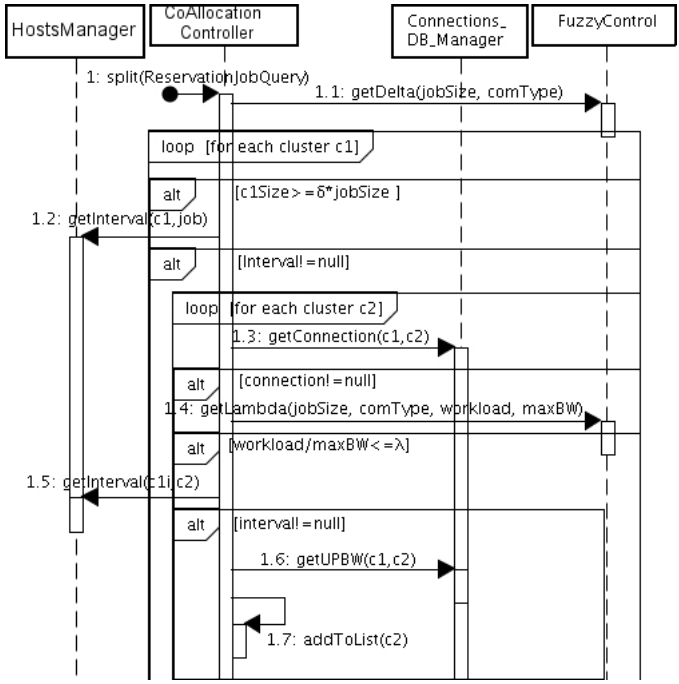

Figura 4.7: Processo de co-alocação

Posteriormente, são calculados os intervalos. Se há um intervalo em *c2* que combina com *c1i*, então o cluster *c2* é adicionado a uma lista na ordem de UPBW (*Unit Process Bandwidth*). O UPBW é um valor que significa $linkW/c1Size$, onde *linkW* é a carga corrente da banda sendo utilizada entre *c1* e *c2*. Essa informação é oferecida pelo *Connections_-DB_Manager*. Quando há processadores suficientes na lista de resultados para satisfazer a requisição, é criada uma reserva para cada cluster então se tenta alocar todos os clusters localmente, por meio do *Connector*. Se uma das reservas falharem, o *CoAllocationController* usa um próximo cluster para ser o *c1*.

4.2.5 Interface com o simulador

O ambiente usado para testes do GRADI foi em simulação, utilizando o GridSim. Para comunicar o servidor com o simulador do GridSim, é usado um conector de troca de mensagens. Foi desenvolvido um ambiente de testes no GridSim, o qual é explicado mais detalhadamente na

seção sobre os resultados. Com relação à comunicação e o escalonamento dos jobs quando chegam no simulador, foi desenvolvida uma extensão do escalonador local de clusters disponível no GridSim.

O servidor entra em contato com o ambiente em três casos: para verificar disponibilidade dos recursos, atualizar informações e solicitar reservas ou alocações. A figura 4.8 mostra o fluxograma para a chegada de requisições de reservas implementado no simulador.

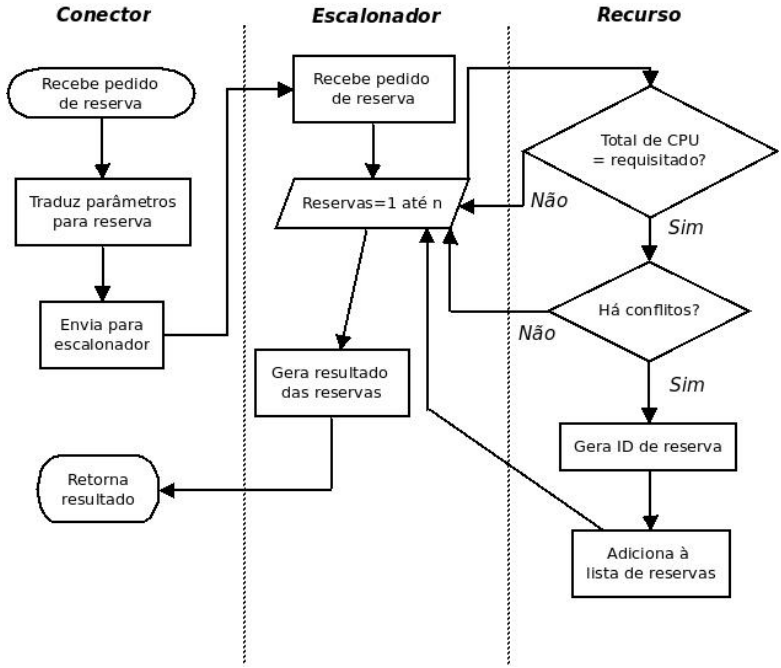

Figura 4.8: Fluxograma da chegada de reservas

Ao lado esquerdo, há um módulo conector para recebimento das mensagens do servidor e tradução dos parâmetros, visto que o simulador exige alguns tipos diferentes do utilizado pelo servidor. Então, é enviado ao escalonador local um job já formatado para a reserva do GridSim.

As reservas recebidas podem ser reservas imediatas, neste caso ocorrendo a direta aloção ou co-alocação dos recursos. Ou então reserva antecipadas, sendo importante as variáveis de tempo. Tratando-se de uma co-reserva, o escalonador percorre a lista de reservas requisitadas pelo job tentando efetuá-las com o recurso.

Cada recurso no GridSim possui escalonadores locais e filas próprias de jobs, como mostrado em [Sulistio e Buyya 2004]. Para aceitar uma requisição, o escalonador do recurso verifica sua disponibilidade e também se há conflitos nos intervalos de tempo com reservas já realizadas. O escalonador retorna como resultado ao servidor as reservas que conseguiu efetuar. Caso em uma co-alocação não for possível reservar localmente todos os clusters requisitados, as reservas são canceladas, e é retornado o resultado ao servidor sobre o cluster que não está disponível.

Capítulo 5

Resultados Experimentais

Neste capítulo serão mostrados o ambiente e os resultados obtidos de alguns casos de estudo utilizando o protótipo desenvolvido. Os experimentos desta pesquisa foram realizados utilizando o simulador GridSim. Foi considerado que todos os clusters oferecem suporte à reserva avançada de recursos, ou seja, os escalonadores dos clusters locais são capazes de realizar reserva para tarefas locais e globais.

5.1 Aspectos da Simulação

Este trabalho baseia-se em uma configuração com um servidor central, clientes remotos e clusters remotos, formando um ambiente de grade com multi-cluster, como é visto na Figura 4.1. O servidor central armazena informações dos clusters e possui um módulo de co-reserva global. Este módulo decide em que cluster os processos de uma tarefa devem ser executados. Em outras palavras, os escalonadores locais dos clusters oferecem suporte à reserva antecipada tanto para tarefas locais como para globais.

O GridSim foi usado para simular esta configuração de multi-clusters. Esta ferramenta oferece um framework de eventos discretos para simulação de diversas classes de entidades de todo um sistema de grade, tal como recursos heterogêneos, usuários, tarefas, escalonadores. Como mostrado na subseção [Sulistio e Buyya 2004], são oferecidos alguns escalonadores para uma ou mais configurações de domínios administrativos, como clusters e grades. Esta característica é interessante para o projeto, pois permite a simulação de configurações multi-cluster em grades.

Outra característica importante do GridSim é que permite reserva antecipada de recursos. Esse pacote de software emprega um algoritmo FCFS (*first-come-first-served*) para escalonamento global. Entretanto, neste trabalho foi utilizado o módulo de co-reserva, no qual a decisão de escalonamento se baseia não somente na disponibilidade dos clusters locais, mas também nas características das tarefas e dos recursos. Já nos clusters locais foi utilizado o suporte a reserva antecipada oferecido pelo simulador, o qual verifica conflitos locais.

A seguir serão mostrados alguns experimentos. Nestes, cada processo é representado por uma entidade *gridlet* do GridSim, e uma tarefa é composta por um ou mais processos.

5.2 Ambiente Experimental

O ambiente criado no GridSim para simulação é composto de duas OVs, cujas características são descritas na Tabela 5.1. VO-C, suponhamos uma instituição canadense, a qual é formada por três clusters conectados entre si e como, por exemplo, VO-B, uma instituição brasileira, possuindo dois clusters. Entretanto, os clusters da VO-C não possuem ligação direta com os clusters da VO-B.

Tabela 5.1: Configuração Multi-cluster

VO-C	processadores	VO-B	processadores
Cluster_01	7	Cluster_1	4
Cluster_02	4	Cluster_2	4
Cluster_03	3	-	-

Após criar a configuração multi-cluster, foram identificados alguns conceitos básicos para serem utilizados nas duas organizações. As similaridades entre essas configurações podem ser observadas na Tabela 5.2. Na primeira coluna é mostrada a ontologia de referência utilizada na descrição das requisições, e também usada para realizar o matching semântico entre as diferentes ontologias de outras organizações. As outras duas colunas mostram as ontologias de descrição utilizadas pelas organizações virtuais que compartilham seus recursos no ambiente proposto para os testes. Essas ontologias foram construídas utilizando Protégé 3.3.1.

5.3 Estudo de Caso

5.3.1 Requisição

Como visto na figura 3.10, vários parâmetros do sistema podem ser utilizados pelo usuário em uma requisição. Isto para que o usuário possa passar ao middleware as configurações necessárias para executar sua aplicação e, assim, o middleware possa decidir quais dos recursos disponíveis melhor se adequam às necessidades do usuário.

Nos experimentos realizados neste trabalho, foram considerados alguns desses parâmetros nas requisições dos usuários. Como o foco do trabalho está na quantidade de processadores e na largura de banda utilizada,

Tabela 5.2: Ontologias utilizadas nos experimentos

Request's Concepts	VO_B's Concepts	VO_C's Concepts
min_number_of_cpus	[numero_cpus]	[number_of_cpus]
min_virtual_memory_available	[swap_livre_MB]	[free_virtual_memory]
min_clock_speed	[velocidade_cpu]	[max_clock_speed]
owner	[nome_conta]	[login]
number_of_cpus	[numero_cpus]	[number_of_cpus]
min_physical_memory_available	[memoria_livre_MB]	[free_physical_memory]
min_free_disk_space	[espaco_disco_livre_GB]	[available_space]
os_type	[SO, SistemaOperacional]	[os_type]
max_load_cpu_15min		[percentage_load_15min]
max_load_cpu_5min		[percentage_load_5min]
max_load_cpu_1min		[percentage_load_1min]

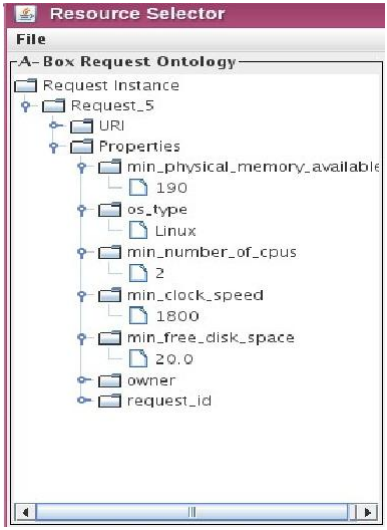

Figura 5.1: Requisição padrão dos testes

grande parte dos outros parâmetros serão modificados durante as requisições, com objetivo de que não influenciem nos resultados obtidos.

Assim, os pedidos dos usuários consideram os seguintes requisitos mostrados na figura 5.1. Ou seja, sempre que o middleware buscar resultados para as requisições, vai procurar em ambientes que correspondam a esses requisitos da figura 5.1, sendo que o parâmetros *min_number_of_cpus* é o único que se modifica para cada requisição. Em seguida, tabela 5.3 mostra os requisitos para reserva de recursos que alteram respectivamente para cada usuário nos experimentos realizados.

Tabela 5.3: Parâmetros requisitados para reserva de recursos

Parâmetros	user_0	user_1
CPUs	2	6
start time (min)	10	20
duração (min)	60	60
deadline (min)	80	90

A seguir serão mostrados alguns experimentos comparando resultados com resposta dinâmica e com reserva de recursos. Nestes, cada processo é representado por uma entidade *gridlet* do GridSim, e uma tarefa é composta por um ou mais processos.

5.3.2 Resultados

Com objetivos de avaliar os aprimoramentos obtidos na extensão do trabalho, os experimentos realizados mostram resultados obtidos considerando as configurações do ambiente original do trabalho de [Silva e Dantas 2007], e posteriormente comparando estes resultados com os resultados obtidos com as características adicionadas com a alocação dinâmica. Por fim, estes resultados também são comparados com os resultados do protótipo final, com configurações para co-reserva de recursos, como mostrado na arquitetura da figura 4.1.

Foram considerados dois usuários, user_0 e user_1, os quais fazem requisições de reservas, cujas características são mostradas na figura 5.1. Considerando primeiramente uma requisição simples, como a do user_-0, na qual o usuário pede apenas por 2 processadores, foram obtidos os seguintes resultados das figuras 5.2, 5.3 e 5.4 para as configurações analisadas a seguir.

A figura 5.2 mostra o resultado da pesquisa do user_0 obtido no sistema com a configuração original de [Silva e Dantas 2007], em que são apenas selecionados de forma estática os recursos previamente armazenados para compartilhamento entre as organizações OV_A e OV_B. No lado esquerdo da interface são mostrados os requisitos do pedido, e no lado direito estão as respostas, que representam os sistemas encontrados pela ferramenta e que correspondem ao pedido.

Neste segundo teste, da figura 5.3, é mostrado o mesmo pedido, mas na configuração desenvolvida que permite alocação dinâmica dos recursos. Assim como no primeiro caso, a ferramenta busca na base semân-

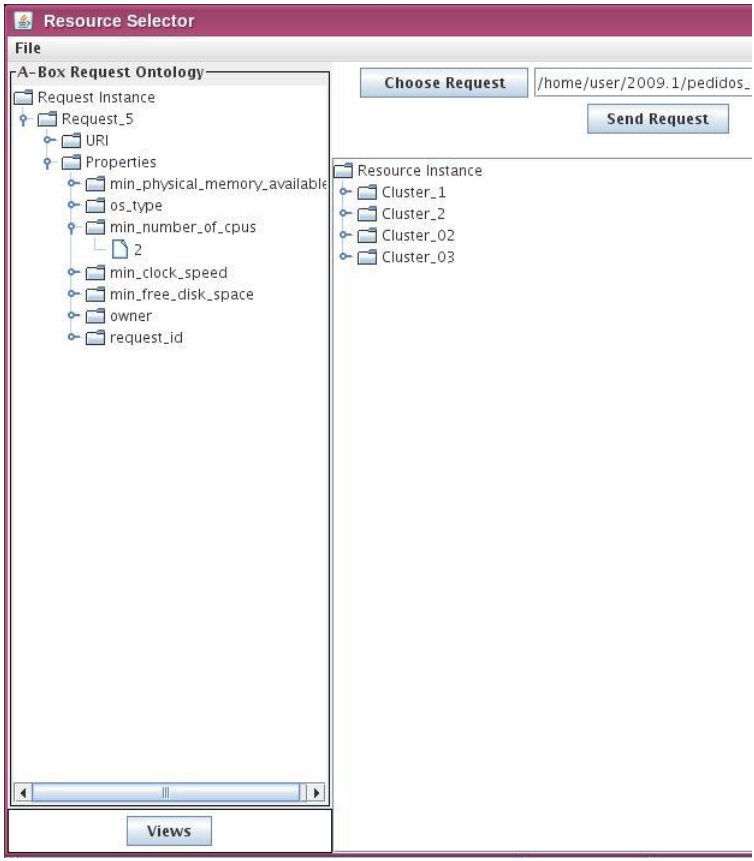

Figura 5.2: Seleção estática de recursos – user_0

tica os resultados. Entretanto, também confere dinamicamente a disponibilidade dos recursos locais (na parte inferior da interface), mostrando a quantidade de processadores disponíveis de cada cluster no momento da alocação. Além disso, também mostra a porcentagem utilizada da banda entre o servidor e cada cluster. Dessa forma o usuário tem informações atualizadas sobre o estado do sistema para tomar uma melhor decisão de alocação.

Já na figura 5.4 é acrescentada a configuração que permite reserva de recurso, para o mesmo teste, agora com tempo de reserva e deadline. É importante destacar que neste caso, o middleware além de verificar di-

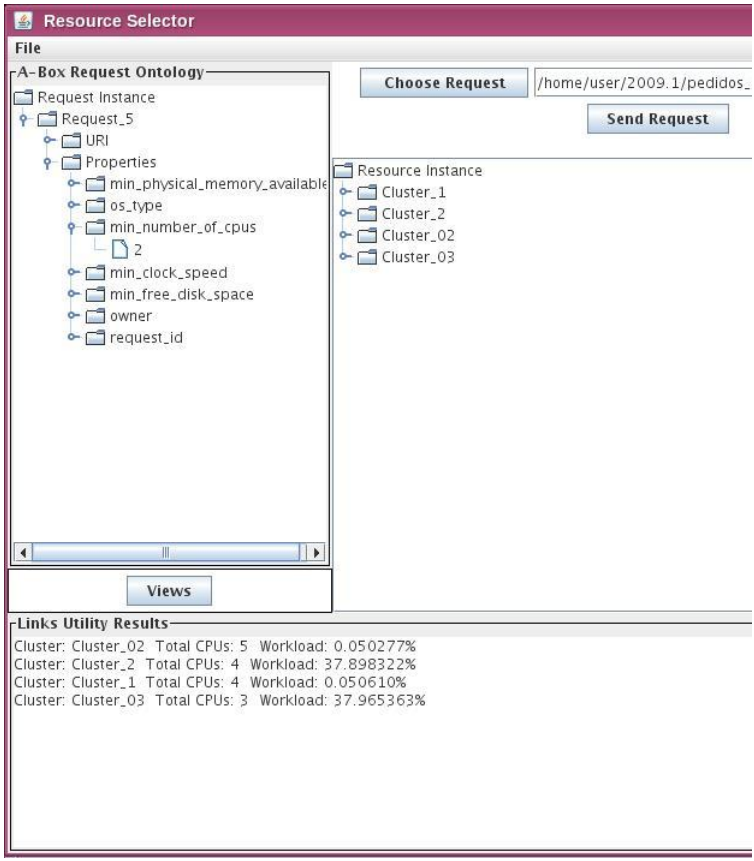

Figura 5.3: Alocação dinâmica de recursos - user_0

namicamente as possibilidades de reserva (mostrado na parte direita da figura), também toma a decisão sobre qual das opções selecionadas seria a mais adequada para a requisição do usuário (parte inferior da figura). Em seguida, realiza a reserva no ambiente e retorna para o usuário o resultado sobre a quantidade de recursos reservadas e em qual cluster.

Desta forma, dentre os clusters que podem responder ao user_0, o Cluster_03 é o que possui menor quantidade de processadores livres. Como não houve conflitos, o cluster foi reservado localmente e globalmente, como mostrado na figura.

Por outro lado, o user_1 pediu por 6 processadores. O Cluster_-

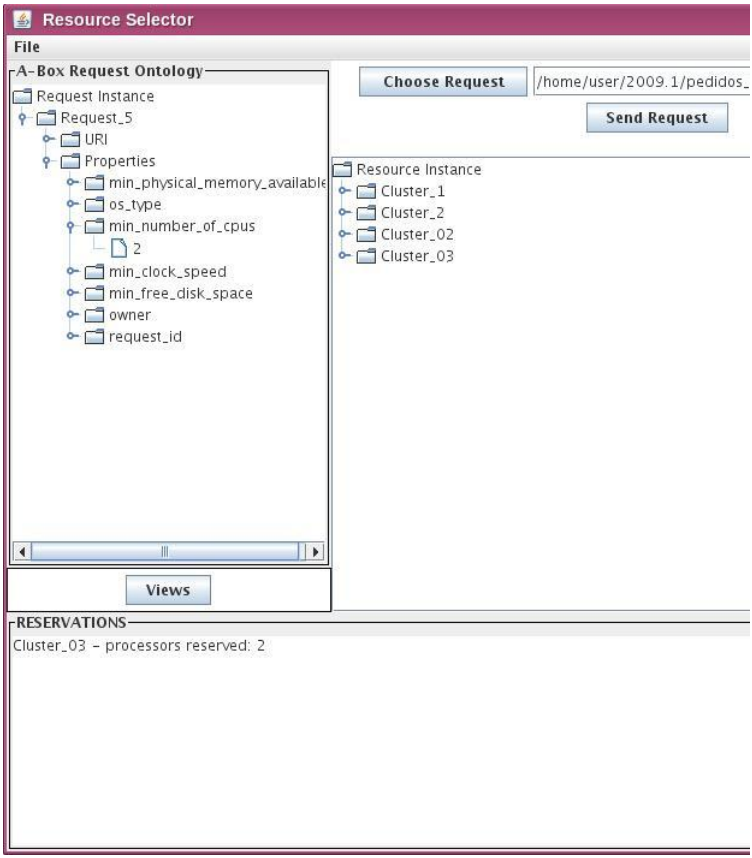

Figura 5.4: Reserva de recursos - user_0

01 seria o único cluster capaz de responder sozinho a essa requisição de processamento. Entretanto, não satisfaz a quantidade mínima de disco exigida. Por essa razão, não serve como resultado no ambiente estático e não aparece nos resultados estáticos do lado direito da tela.

Nenhum dos clusters do ambiente compartilhado possui sozinho capacidade de responder a esta requisição. Assim, utilizando as configurações do trabalho de [Silva e Dantas 2007], a ferramenta não encontra solução para o pedido, como mostrado na figura 5.5. Isso ocorre porque a ferramenta não considera a possibilidade de junção de processadores ociosos entre clusters.

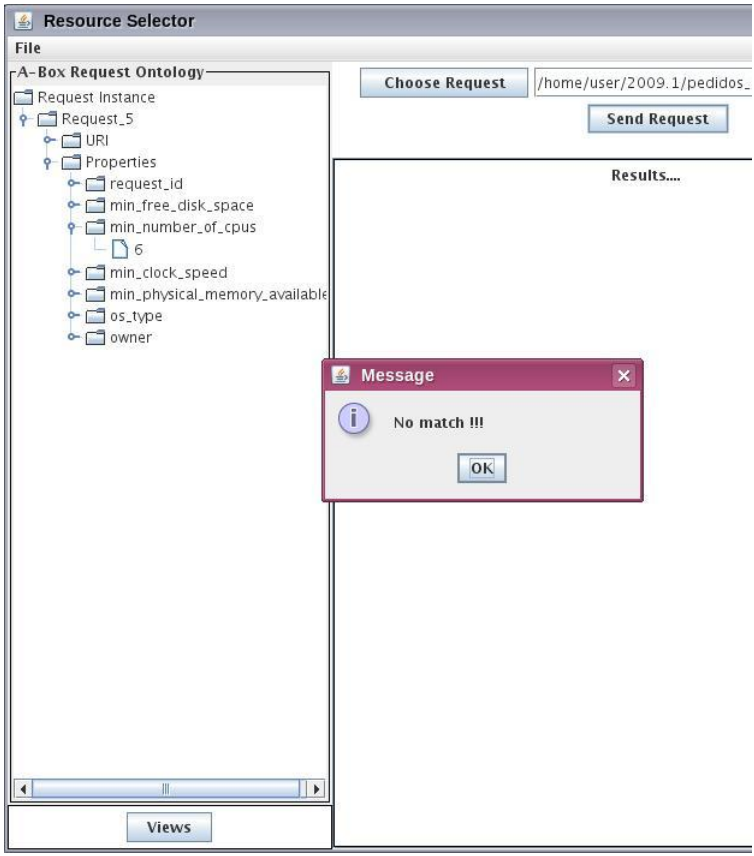

Figura 5.5: Seleção estática de recursos – user_1

A mesma consulta do user_1 realizada no ambiente com alocação dinâmica já apresenta uma solução para o problema, como mostrado na figura 5.6. Na parte direita da figura são mostrados os clusters que possuem a configuração requerida pelo usuário, mas sem considerar a quantidade mínima de processamento. Isto porque a nova proposta considera todos os processadores existentes em todos os clusters que possuem um link de comunicação entre si. Neste caso, o middleware calcula a combinação de clusters que possa responder a requisição e com uma porcentagem razoável de banda disponível para comunicação entre estes, sugerindo a distribuição dos processos.

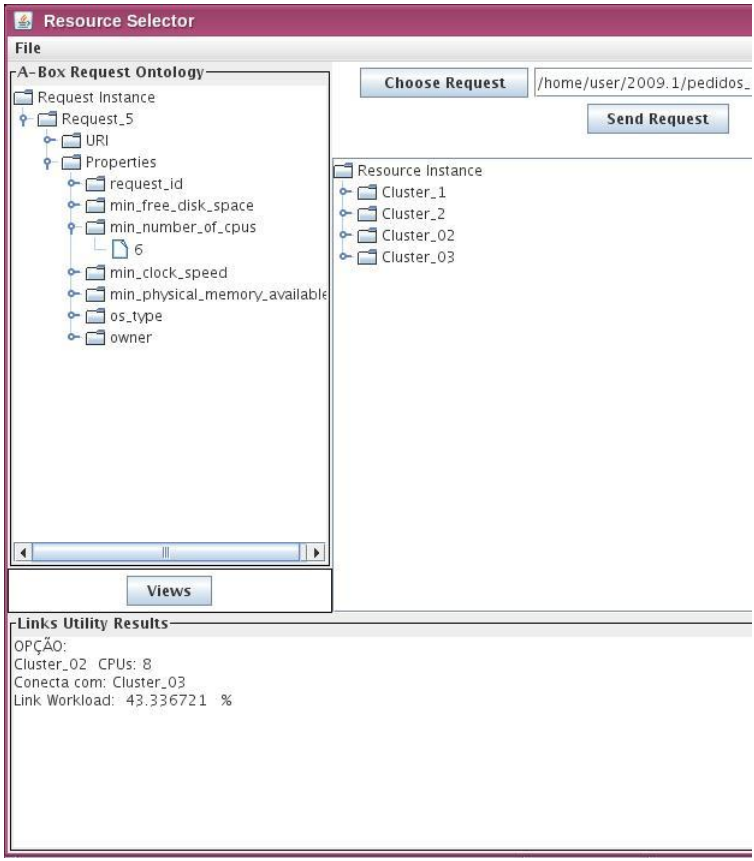

Figura 5.6: Alocação dinâmica de recursos - user_1

A solução mais adequada de alocação encontrada pelo middleware para este pedido envolve o Cluster_02 e o Cluster_03, que combinados possuem um total de 8 processadores. É mostrada também a porcentagem de banda utilizada no link de comunicação entre estes clusters no momento da alocação, na parte inferior da interface.

Por fim, o mesmo pedido do user_1 para 6 processadores é realizado no sistema com reserva de recursos, mas considerando parâmetros de tempo, sendo uma reserva para o futuro, como mostrado na tabela 5.3. É feito o mesmo cálculo para distribuição dos processos, mas considerando informações de reservas anterirores, e então é feito dinamicamente

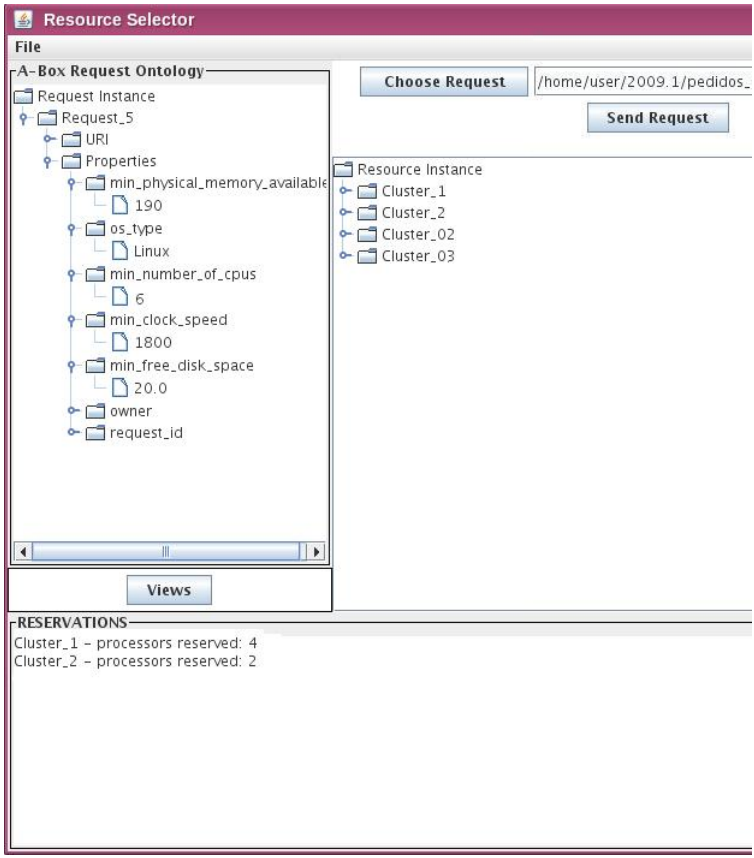

Figura 5.7: Reserva de recursos - user_1

a reserva local dos clusters calculados como mais adequados. O resultado da distribuição pode ser visualizado na figura 5.7, a qual mostra para o usuário que 4 processadores do Cluster_1 e 2 processadores do Cluster_2 foram reservados.

É importante notar que neste caso o Cluster_03 possui 2 processadores reservados pelo user_0, e por isso não pode servir como resposta para a requisição do user_1, que pede intervalo de tempo similar. Neste caso, a única combinação de clusters possível é a mostrada no resultado da figura 5.7.

A figura 5.8 mostra a quantidade de processadores reservadas por

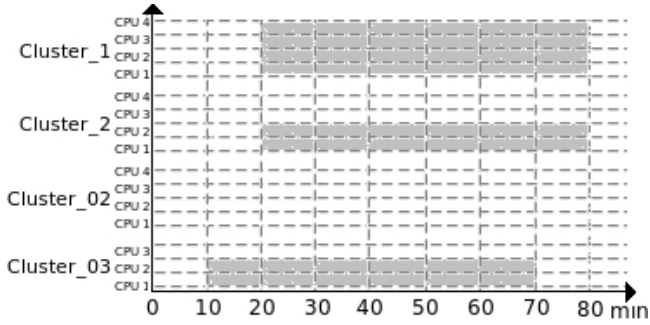

Figura 5.8: Mapa de alocação de recursos.

estas requisições desses usuários em relação ao tempo. Analisando os resultados, percebe-se que uma distribuição adequada dos processos entre clusters pode evitar a fragmentação na utilização dos recursos no tempo.

A distribuição de processos entre clusters possibilita atender jobs com grande necessidade de processamento, sem afetar o desempenho do sistema, caso seja feita uma análise das possibilidades de comunicação dos processos entre clusters.

O uso de co-reservas antecipadas em conjunto com co-alocação se mostra uma ferramenta importante para o desempenha de ferramentas para grades. Permite a execução de jobs maiores, evitando demora excessiva no sistema e oferecendo uma maior utilização dos recursos disponíveis no ambiente. Além disso, oferece maiores garantias de disponibilidade de recursos, melhorando a qualidade do serviço.

5.4 Experimentos em maior escala

5.4.1 Descrição

Nesta segunda etapa de experimentos, são realizados testes variando a quantidade de usuários com objetivo de avaliar o nível de utilização dos recursos, eficácia e desempenho do GRADI, o qual é comparado com o *ARSimpleSpaceShared* do GridSim.

Como ambiente de testes, foi utilizado o mesmo descrito na seção 5.2. O ambiente suporta um determinado número de requisições, e a medida que estas aumentam é possível avaliar o comportamento sobre a utilização dos recursos em cada trabalho quando há grande concorrência entre os usuários.

Cada usuário envia uma requisição com diferentes parâmetros de

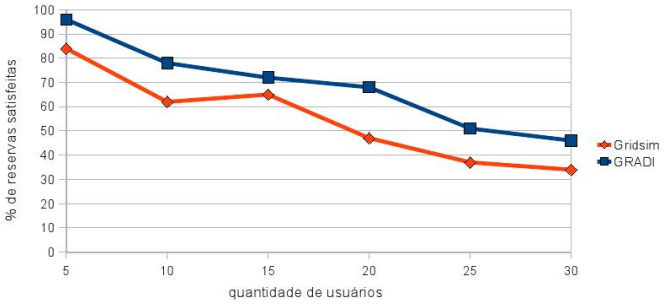

Figura 5.9: Reservas realizadas por quantidade de usuários concorrentes

restrição. Foi considerado que essas requisições enviadas são compostas de processos com dependência de comunicação entre si e com restrições rígidas sobre o tempo de execução. A quantidade de processos para cada requisição é gerada de forma aleatória, dentro do intervalo [3,6]. Os tempos são medidos em minutos e também gerados aleatoriamente: o início de cada job varia de 0 (reserva imediata) a 100 minutos; enquanto a duração de 10 a 60 minutos.

Foram realizados 6 subconjuntos de testes, cada qual referindo-se a uma determinada quantidade de usuários (5, 10, 15, 20, 25, 30). Em cada subconjunto foram realizados repetidos testes, de forma que os resultados dos gráficos referem-se a porcentagem média obtida nos experimentos.

Os testes foram gerados automaticamente, sem a considerar a interface gráfica do GRADI, e nem a opção de seleção manual dos recursos.

5.4.2 Resultados

No experimento, uma determinada quantidade de usuários enviam requisições. O gráfico da figura 5.9 mostra a porcentagem de êxito em reservas, com relação a quantidade de usuários no sistema. Percebe-se que o GRADI apresenta sempre maior eficácia nas requisições. Isso ocorre pelo algoritmo que permite a co-reserva de processadores entre clusters. Desta forma, percebe-se que o algoritmo do GRADI apresenta uma melhor eficácia na reserva.

Mesmo em testes com poucos usuários o GRADI consegue alcançar resultados melhores de reserva. A tendência é que quanto mais usuários no sistema, maior a diferença entre os resultados dos dois trabalhos. Entretanto, alguns resultados mostraram-se próximos, como no caso de 15 usuários. Isso acontece porque o *ARSimpleSpaceShared* faz várias tentati-

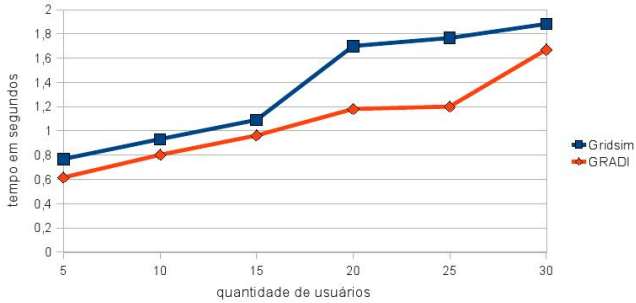

Figura 5.10: Tempo final de resposta do sistema para todas as requisições

vas em diferentes clusters que não apenas o sugerido na requisição, o que aumenta as chances de conseguir a reserva. Já o GRADI retorna ao usuário quando não consegue reserva no determinado cluster solicitado.

No resultado para 30 usuários o sistema está saturado, com apenas 49% de reservas atedidas, enquanto no *ARSimpleSpaceShared* este número cai para 33%.

Por outro lado, as funcionalidades a mais do GRADI influenciam no seu tempo de resposta, como mostrado no gráfico da figura 5.10. Neste é mostrado o tempo de resposta médio de execução de todas as reservas para cada subconjunto de usuários.

Até 15 usuários o sistema apresenta um crescimento de tempo de resposta linear, ligeiramente acima do *ARSimpleSpaceShared*. Entretanto, com o aumento significativos de usuários, o sistema passa a ter um overhead considerável, por saturação no sistema. Isso se deve a uma grande quantidade de co-alocações e comunicação entre clusters, visto que é verificado de forma dinâmica os recursos do cluster de acordo com a requisição, e o link de comunicação entre os mesmos.

Já o *ARSimpleSpaceShared* apresenta um melhor tempo de resposta até 25 usuários. Com 30 usuários há também uma subida considerável, visto que todos os clusters são consultados para cada usuário e são realizadas várias tentativas de alocação sem êxito, visto que o sistema está saturado.

5.5 Considerações Finais

A tabela 5.4 mostra a comparação entre as principais características dos trabalhos relacionados com as características do middleware apresentado nesta dissertação.

Como visto, o trabalho proposto acrescenta alocação dinâmica, co-alocação e reserva de recursos ao trabalho de [Silva e Dantas 2007]. São também realizados testes em ambientes utilizando simulações, o que não tinha sido feito antes em [Silva e Dantas 2007].

Para controle de co-alocação, foi utilizado o algoritmo de lógica difusa, proposto no trabalho mostrado em [Qin e Bauer 2009]. Entretanto, este trabalho não aborda a questão da descrição de recursos em diferentes Ovs. Além disso, se restringe a seleção de processadores e banda, enquanto o middleware proposto trabalha com a seleção de diferentes tipos de recursos, o que permite ao usuário fazer uma maior restrição ao tipo de configuração em que deseja executar seus jobs.

Outro ponto importante, é que o middleware desenvolvido permite reserva antecipada de recursos. Este item é abordado nos trabalhos mostrados em [Vahdat-Nejad, Monsefi e Naghibzadeh 2007] e [Liu, Dai e Chuang 2006]. O primeiro utiliza uma arquitetura em camadas, com um escalonador global, o qual faz as decisões de escalonamento relativas à grade, enquanto os escalonadores locais tomam suas próprias decisões, cooperando com o global. O middleware desenvolvido no trabalho desta dissertação possui uma arquitetura parecida, entretanto, o escalonador global possui armazenado as reservas realizadas e informações sobre o ambiente de cada cluters, permitindo uma decisão estática inicial sobre a reserva.

Por outro lado, [Vahdat-Nejad, Monsefi e Naghibzadeh 2007] verifica dinamicamente a situação da rede interna dos clusters para decidir a possibilidade de execução. O middleware proposto não avalia se a rede interna do cluster está saturada, mas avalia as conexões entre clusters, para realizar co-alocação.

Por fim, no trabalho correlato em [Liu, Dai e Chuang 2006], são feitas reservas de recurso para transferência de vídeo, permitindo prioridades entre as requisições. O trabalho do middleware proposto não trata ainda dessa questão, entretanto, permite uma configuração muito mais detalhada com relação aos requisitos do ambiente para execução pedidos pelo usuário. Além disso, permite a co-alocação, o que não é abordado em [Vahdat-Nejad, Monsefi e Naghibzadeh 2007] e nem em [Liu, Dai e Chuang 2006].

É importante ressaltar as diferenças no foco de cada trabalho de reserva de recursos. [Vahdat-Nejad, Monsefi e Naghibzadeh 2007] trata do tráfego na rede interna dos clusters, e cria um sistema que associa cada cluster a requisição de acordo com o nível de comunicação e saturação interno de cada um. Já no trabalho de [Liu, Dai e Chuang 2006], a reserva de recursos depende da prioridade criada para cada requisição. Esta prioridade é baseada nos parâmetros de *delay* e *jitter* dados pelo usuário. Posteriormente é decidido onde rodar as requisições prioritárias no ambi-

ente em que a banda não esteja saturada para transferência de vídeos.

O middleware proposto neste trabalho favorece, na reserva antecipada, a execução de jobs com necessidade de muito processamento, ao permitir a co-alocação. As reservas são baseadas em vários quesitos passados pelo usuário, que não apenas processamento. Fragmentos ociosos entre reservas são aproveitados para evitar uma maior fragmentação durante o tempo, o que pode prorrogar a execução de jobs grandes. Além disso, é feito um controle de saturação dos links entre os clusters. Assim como este middleware, [Vahdat-Nejad, Monsefi e Naghibzadeh 2007] e [Liu, Dai e Chuang 2006] utilizam lógica difusa na decisão de reserva.

Tabela 5.4: Quadro Comparativo

	Silva e Dantas 2007	Qin e Bauer 2009	Sulistio e Buyya 2004
integra OVs	X	-	X
Atualização	estático	dinâmico	dinâmico
Co-Alocação	-	X	-
Reserva Antecipada	-	-	X
Tipos de Recursos	vários	processamento e banda	vários

	Vahdat-Nejad te al. 2007	Liu, Dai e Chuang 2006	GRADI
integra OVs	-	-	X
Atualização	dinâmico	dinâmico	dinâmico
Co-Alocação	-	-	X
Reserva Antecipada	X	X	X
Tipos de Recursos	processamento e banda	banda	vários

Capítulo 6

Conclusões e Trabalhos Futuros

6.1 Conclusões

Neste trabalho foi proposta uma arquitetura para reserva antecipada de recursos em grades organizadas em configuração multi-cluster, com objetivo de maximizar a utilização dos recursos. Para alcançar estes objetivos, foi desenvolvido o GRADI, um protótipo de middleware para reserva antecipada de recursos. Os testes simulados com o GRADI mostraram resultados satisfatórios sobre a distribuição dos processos no ambiente de grade.

Para desenvolver o GRADI, foram estudados vários middlewares na literatura, focando nas formas de gerenciamento, reserva e alocação dos recursos, dentre os quais, o CSF e o Gridway. Com esta base foi possível detalhar conceitos fundamentais sobre meta-escalonamento, co-reserva e co-alocação de recursos. Foram destacados dois trabalhos, [Silva e Dantas 2007] e [Qin e Bauer 2009], respectivamente sobre descrição e co-alocação de recursos, os quais motivaram a arquitetura para a implementação do GRADI. Por fim, foram realizados experimentos por meio de simulações comparando as funcionalidades do GRADI com outros gerenciadores da literatura.

Como primeira parte da implementação, foi feita uma extensão do trabalho de [Silva e Dantas 2007], o qual trata do gerenciamento integração de diferentes descrições de recursos para compartilhamento entre organizações utilizando ontologias. O objetivo inicial foi acrescentar suporte a co-alocação dinâmica entre destes recursos. A implementação do algoritmo de co-alocação teve como base os estudos de [Qin e Bauer 2009], empregando lógica difusa para controle de banda entre clusters. Por fim, foi desenvolvido o terceiro módulo que trata da reserva de recursos, utilizando lógica difusa para co-reserva entre processadores e banda, mantendo ontologias para integração entre as diferentes descrições.

As simulações foram feitas no GridSim. Este simulador foi escolhido, principalmente, por oferecer suporte a reserva de recursos no am-

biente simulado. Foi desenvolvido um ambiente experimental neste simulador, visando criar um ambiente com concorrência entre usuários pelos recursos. Os experimentos iniciais comparam o trabalho inicial estático, com uma segunda implementação adicionando co-alocação dinâmica e com o trabalho final, o qual possibilita reserva de recursos. Os resultados mostraram-se satisfatórios, percebendo-se que aumentou a possibilidade de configurações entre recursos para atender as requisições, e também que o usuário passa ter maiores garantias de execução em tempo e recursos para suas aplicações.

Foram realizados testes comparativos também com o escalonador para reserva de recursos do GridSim [Sulistio e Buyya 2004], os quais mostraram o aumento no nível de utilização dos recursos com o protótipo do GRADI, o qual apresenta melhores resultados com o aumento de usuários. Entretanto, isto acarreta em um pior desempenho geral do sistema, comparado ao de [Sulistio e Buyya 2004].

6.2 Contribuições

Com o desenvolvimento e implementação deste trabalho de mestrado, são listados a seguir as principais contribuições adquiridas:

- O desenvolvimento de uma nova abordagem para reserva de recursos, o qual tem como diferencial o enfoque na reserva de configurações adequadas para melhor execução de cada job; e também na maximização da utilização dos recursos disponíveis no sistema, e permitindo que algumas requisições maiores, que ficariam mais tempo em filas, tenham maiores possibilidades para reserva. O método une algoritmo de lógica difusa para avaliação das condições futuras de banda entre clusters e ontologias para lidar com a heterogeneidade na descrição dos sistemas.

- Uma segunda contribuição foi o aprimoramento do protótipo de [Silva e Dantas 2007], acrescentando as funcionalidades de reserva antecipada e co-alocação de recursos, adicionando à interface gráfica interativa com o usuário.

- Outra contribuição importante foram os resultados obtidos com a avaliação comparativa com trabalhos semelhantes, mostrando um melhor nível de utilização dos recursos e maiores possibilidades de configuração para o usuário.

- Por fim, o trabalho também oferece uma revisão bibliográfica sobre gerenciamento de recursos focando meta-escalonamento. Neste são

explicados principais conceitos na literatura sobre descrição de recursos, co-escalonamento, co-alocação e co-reserva, detalhando um pouco sobre exemplos de protótipos desenvolvidos na área.

6.3 Trabalhos Futuros

O GRADI ainda pode ter melhorias em vários setores, o que possibilita o desenvolvimento de outros trabalhos em algumas áreas. Alguns dos possíveis trabalhos futuros são sugeridos a seguir:

- Adicionar prioridades aos usuários, oferecendo diferentes níveis de qualidade de serviço no uso dos recursos.
- Acrescentar um módulo de tolerância a falhas para execução das aplicações no ambiente. Por enquanto o módulo apenas verifica dinamicamente a possibilidade de execução antes do envio.
- Estudar melhorias para maximizar o desempenho do sistema como um todo;
- Realizar testes em ambiente real, trabalhando a questão de desenvolver interfaces para implementação com diferentes gerenciadores de recursos locais.

Referências Bibliográficas

[Agarwal et al. 2006]AGARWAL, V. et al. Deco: Data replication and execution co-scheduling for utility grids. *Lecture Notes in Computer Science, Service-Oriented Computing (ICSOC'06)*, p. 52–65, 2006.

[Andrade et al. 2007]ANDRADE, J. et al. Applications of grid computing in genetics and proteomics. n. 4699, p. 791–798, 2007.

[Bai, Zhuang e Wang 2006]BAI, Y.; ZHUANG, H.; WANG, D. Advanced fuzzy logictechnologies in industrial applications. *Springer-Verlag London Limited*, 2006.

[Banen, Bucur e Epema 2003]BANEN, S.; BUCUR, A.; EPEMA, D. A measurement-based simulation study of processor co-allocation in multicluster systems. *JSSPP - Springer-Verlag, Lect. LNCS2862*, p. 1–20, 2003.

[Barz, Pilz e Wichmann 2008]BARZ, C.; PILZ, M.; WICHMANN, A. Temporal routing metrics for networks with advance reservations. *CCGRID*, p. 710–715, 2008.

[Beginner's guide to sun tm Grid Engine 2009]BEGINNER'S guide to sun tm Grid Engine. White paper, july 2009.

[Boden et al. 1995]BODEN, N. et al. Myrinet: A gigabit-per-second local area network. v. 15, p. 29–36, 1995.

[Burchard 2005]BURCHARD, L.-O. Analysis of data structures for admission control of advance reservation requests. *IEEE Transactions on Knowledge and Data Engineering*, v. 17, n. 3, p. 413–424, 2005.

[Burchard e Heiss 2003]BURCHARD, L.-O.; HEISS, H.-U. Performance issues of bandwidth reservations for grid computing. *Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'03)*, 2003.

[Buyya 2003]BUYYA, R. *High Performance Cluster Computing: Architectures and Systems*. [S.l.]: Axcel Books, 2003.

[Buyya 2008]BUYYA, R. Market-oriented grid computing and the gridbus middleware. *IEEE 16th International Conference on ADCOM*, Jan. 2008.

[Buyyaa, Abramson e Venugopal 2005]BUYYAA, R.; ABRAMSON, D.; VENUGOPAL, S. The grid economy. *Proceedings of the IEEE*, v. 93, n. 3, p. 698–714, 2005.

[Colvero, Dantas e Cunha 2005]COLVERO, T. A.; DANTAS, M.; CUNHA, D. P. da. Ambientes de clusters e grids computacionais: Características, facilidades e desafios. UNESC - Criciúma, 2005.

[Condor 2010]CONDOR. 2010. Disponível em: <http://www.cs.wisc.edu/condor/>.

[Connolly et al. 2001]CONNOLLY, D. et al. Daml+oil. *Reference Description. W3C Note*, March 2001.

[Coulouris, Dollimore e Kindberg 2005]COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Distributed Systems: Concepts and Design*. fourth. [S.l.]: Pearson Education, 2005.

[Czajkowski, Foster e Kesselman 1999]CZAJKOWSKI, K.; FOSTER, I.; KESSELMAN, C. Resource co-allocation in computational grids. p. 219–228, 1999.

[Dalheimer, Pfreundt e Merz 2005]DALHEIMER, M.; PFREUNDT, F.-J.; MERZ, P. Calana: a general-purpose agent -based grid scheduler. *14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*, p. 279–280, 2005.

[Dantas 2005]DANTAS, M. *Computação Distribuída de Alto Desempenho: Redes, Clusters e Grids Computacionais*. [S.l.]: Axcel Books, 2005.

[Degermark et al. 1995]DEGERMARK, M. et al. Advance reservation for predicted service. *Fifth Intero e national Workshop on Network and Operating System Support for Digital Audio and Video*, April 1995.

[Ding et al. 2009]DING, Z. et al. Implement the grid workflow scheduling for data intensive applications with csf4. *IEEE Fourth International Conference on eScience*, p. 563–569, January 2009.

[DRMAA]DRMAA. DRMAA Working Group. Disponível em: <http://www.drmaa.org/>.

[Eickermann et al. 2007]EICKERMANN, T. et al. Co-allocation of mpi jobs with the viola grid metascheduling framework. *CoreGRID, Technical Report Number TR-0081 (JSSPP'07), LNCS 4942*, p. 152–168, 2007.

[Ejarque et al. 2008]EJARQUE, J. et al. Using semantics for resource allocation in computing service providers. p. 583–587, 2008.

[Ferreira et al. 2008]FERREIRA, D. J. et al. Grid computing middleware survey. *International Information and Telecommunication Technologies Symposium*, Dec. 2008.

[Ferreira et al. 2009]FERREIRA, D. J. et al. Toward resource management in multi-cluster grid configurations through an ontology-fuzzy approach. *International Conference on Grid Computing and Applications (GCA)*, 2009.

[Foster 2002]FOSTER, I. What is the grid? a three point checklist. p. 22–25, 2002.

[Foster 2006]FOSTER, I. Globus toolkit version 4: Software for service-oriented systems. *International Conference on Network and Parallel Computing*, Springer-Verlag LNCS, p. 2–13, 2006.

[Foster et al. 1999]FOSTER, I. et al. A distributed resource management architecture that supports advance reservations and co-allocation. p. 27–36, 1999.

[Foster, Kesselman e Tuecke 2001]FOSTER, I.; KESSELMAN, C.; TUECKE, S. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Software Architecture*, v. 3, n. 15, 2001.

[Gaj et al. 2002]GAJ, K. et al. Performance evaluation of selected job management systems. Florida, April 2002.

[Gehring e Preiss 1999]GEHRING, J.; PREISS, T. Scheduling a metacomputer with uncooperative sub-schedulers. *Proc. of the 5th Int'l Workshop on Job Scheduling Strategies for Parallel Processing, LNCS(1659)*, p. 179–201, 1999.

[GridWay Metascheduler]GRIDWAY Metascheduler. Disponível em: <http://www.gridway.org>.

[Hussain e Ahmed 2006]HUSSAIN, S.; AHMED, G. A comparative study and analysis of pvm and mpi for parallel and distributed systems. p. 183–187, Feb. 2006.

[Janson, Dantas e Bauer 2010]JANSON, D.; DANTAS, M. A. R.; BAUER, M. A. An ontological-fuzzy approach to advance reservation in multi-cluster grids (aceito). *Journal of Physics: Conference Series (JPCS)*, 2010.

[Janson et al. 2009]JANSON, D. et al. Dynamic resource matching for multi-clusters based on an ontology-fuzzy approach. 2009.

[Klein et al. 2003]KLEIN, M. et al. *Ontologies and Schema Languages on the Web (D. Fensel, J.A. Hendler, H. Lieberman and W. Wahlster, eds.)*. [S.l.: s.n.], 2003.

[Klyne e Carroll 2004]KLYNE, G.; CARROLL, J. Resource description framework (rdf) - concepts and abstract syntax. *W3C Recommendation*, Feb. 2004.

[Korpyljov et al. 2009]KORPYLJOV, D. et al. Open grid services architecture overview. *10th International Conference on CADSM*, April 2009.

[Li 2005]LI, K. Job scheduling for grid computing on metacomputers. *IPDPS*, 2005.

[Liu, Dai e Chuang 2006]LIU, X.; DAI, Q.; CHUANG, L. A fuzzy advance reservation mechanism of network bandwidth in video grid. *Third International Conference in Fuzzy Systems and Knowledge Discovery (FSKD'06)*, v. 4223, p. 706–715, 2006.

[LSF 2010]LSF. LSF Platform, 2010. Disponível em: <http://www.platform.com/grids/platform-lsf>.

[LSF guide 2002]LSF guide. Platform LSF Programmer's Guide, versão 7 update 2, 2002.

[LSF Product Suite]LSF Product Suite. Platform Computing Inc. Disponível em: <http://www.platform.com/Products>.

[Manager/Moab Workload Manager]MANAGER/MOAB Workload Manager. Disponível em: <http://www.clusterresources.com/pages/products.php>.

[McGuinness e Harmelen 2004]MCGUINNESS, D.; HARMELEN, F. v. Owl web otology language overview. *W3C Recommendation*, Feb. 2004.

[Mirin e Worley 2008]MIRIN, A.; WORLEY, P. Extending scalability of the community atmosphere model. 2008.

[MPI Project]MPI Project. The Message Passing Interface (MPI) standard. Disponível em: <http://www.mcs.anl.gov/research/projects/mpi/>.

[Netto, Bubendorfer e Buyya 2007]NETTO, M. A. S.; BUBENDORFER, K.; BUYYA, R. Sla-based advance reservations with flexible and adaptive time qos parameters. *In Proceedings of the 5th International Conference on ServiceOriented Computing (ICSOC)*, v. 4749, p. 119–131, 2007.

[PBS]PBS. Disponível em: <http://www.pbsgridworks.com>.

[Petrini et al. 2003]PETRINI, F. et al. Performance evaluation of the quadrics interconnection network. v. 6, p. 125–142, April 2003.

[Pinto, Mendonça e Dantas 2008]PINTO, L. C.; MENDONçA, R. P.; DANTAS, M. Impact of interconnection networks and application granularity to compound cluster environments. p. 468–473, July 2008.

[Pugliese, Talia e Yahyapour 2008]PUGLIESE, A.; TALIA, D.; YAHYAPOUR, R. Modeling and supporting grid scheduling. *Journal of Grid Computing*, v. 6, p. 195–213, 2008.

[PVM project]PVM project. PVM: Parallel Virtual Machine. Disponível em: <http://www.csm.ornl.gov/pvm/>.

[Qin e Bauer 2006]QIN, J.; BAUER, M. A simulator for job co-allocation in multiple hpc clusters. *PDCS*, p. 308–314, 2006.

[Qin e Bauer 2007]QIN, J.; BAUER, M. A. An improved job co-allocation strategy in multiple hpc clusters. p. 18, 2007.

[Qin e Bauer 2009]QIN, J.; BAUER, M. A. Job co-allocation strategies for multiple high performance computing clusters. p. 323–340, 2009.

[Roblitz e Reinefeld 2005]ROBLITZ, T.; REINEFELD, A. Co-reservation with the concept of virtual resources. *CCGRID*, p. 710–715, 2005.

[Roure, Jennings e Shadbolt 2005]ROURE, D. D.; JENNINGS, N.; SHADBOLT, N. The semantic grid: Past, present, and future. *Proceedings of the IEEE*, v. 93, n. 3, p. 669–681, 2005.

[Seifert 1998]SEIFERT, R. *Gigabit Ethernet: Technology and Applications for High Speed LANs*. [S.l.]: Addison-Wesley, 1998.

[SGE 2010]SGE. Open Source Grid Engine, 2010. Disponível em: <http://gridengine.sunsource.net/>.

[Shanley 2002]SHANLEY, T. *Infiniband Network Architecture*. [S.l.]: Addison-Wesley, 2002.

[Silva e Dantas 2007]SILVA, A. P. C.; DANTAS, M. A. R. A selector of grid resources based on the semantic integration of multiple ontologies. Gramado, Brazil, p. 143–150, 2007.

[Sodan 2005]SODAN, A. C. Loosely coordinated coscheduling in the context of other approaches for dynamic job scheduling: a survey. *Journal Concurrency and Computation: Practice and Experience*, v. 17, p. 1725–1781, 2005.

[Su e Ilebrekke 2002]SU, X.; ILEBREKKE, L. A comparative study of ontology languages and tools. *14th International Conference In Advanced Information Systems Engineering*, p. 761–765, 2002.

[Sulistio e Buyya 2004]SULISTIO, A.; BUYYA, R. A grid simulation infrastructure supporting advance reservation. *In Proc. of the 16th International Conference on Parallel and Distributed Computing and Systems (PDCS'04)*, November 2004.

[Takefusa et al. 2008]TAKEFUSA, A. et al. Gridars: An advance reservation-based grid co-allocation framework for distributed computing and network resources. LNCS 4942, p. 152–168, 2008.

[Torque 2010]TORQUE. Torque/PBS, 2010. Disponível em: <http://www.clusterresources.com/products/torque-resource-manager.php>.

[Vahdat-Nejad, Monsefi e Naghibzadeh 2007]VAHDAT-NEJAD, H.; MONSEFI, R.; NAGHIBZADEH, M. A new fuzzy algorithm for global job scheduling in multiclusters and grids. *International Conference on Computational Intelligence for Measurement Systems and Applications (CISMA'07)*, p. 54–58, 2007.

[Vogels et al. 2000]VOGELS, W. et al. Tree-saturation control in the ac3 velocity cluster. August 2000.

[Wang 1997]WANG, L. X. A course in fuzzy systems and control. *Prentice Hall PTR*, 1997.

[Wang et al. 2007]WANG, M. et al. A pipeline virtual service pre-scheduling pattern and its application in astronomy data processing. v. 83, n. 1, p. 123–132, 2007.

[Weissman 1998]WEISSMAN, J. B. Metascheduling: A scheduling model for metacomputing systems. *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing*, p. 348, 1998.

[Wieder e Ziegler 2006]WIEDER, P.; ZIEGLER, W. Bringing knowledge to middleware - grid scheduling ontology. *Future Generation Grids, Springer*, p. 47–59, 2006.

[Wolf e Steinmetz 1997]WOLF, L. C.; STEINMETZ, R. Concepts for resource reservation in advance. *Journal of Multimedia and Applications*, p. 255–278, 1997.

[Wu et al. 2005]WU, L. et al. An adaptive advance reservation mechanism for grid computing. *Sixth International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT'05)*, p. 400–403, 2005.

[Yeo et al. 2006]YEO, C. S. et al. Cluster computing: High-performance, high-availability, and high-throughput processing on a network of computers. 2006.

Apêndice A

Lógica Difusa

A.1 Detalhes da Lógica Difusa Utilizada

A lógica de fusa é uma forma multi-valorada, com objetivo de alcançar maior aproximação dos resultados. Suas variáveis possuem valores dentro do intervalo entre 0 e 1, podendo ser parcialmente verdadeiro ou parcialmente falso. Além disso, usando linguística, o grau das variáveis é definido por meio de funções específicas, podendo atribuir pesos diferentes aos valores de entrada. Desta forma, a lógica difusa pesquisa o tratamento da incerteza, gerando saída lógica a partir de entradas imprecisas [Bai, Zhuang e Wang 2006].

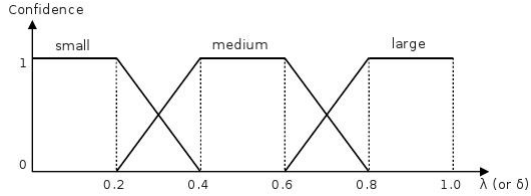

Figura A.1: Função de saída dos valores de controle λ e δ, em [Qin e Bauer 2009]

Este trabalho de dissertação utiliza o método de [Qin e Bauer 2009] para gerar valores de controle de saturação de banda e distribuição de processos entre clusters, respectivamente λ e δ. Neste trabalho, são geradas 4 funções específicas, sobre as seguintes variáveis de entrada:

1.Temanho do job (quantidade de processadores requisitado);

2.Carga do link entre clusters;

3.Largura máxima de banda entre clusters;

4.Tipo de comunicação entre processos do job;

Tabela A.1: Regras para mapeamento de valores, de [Qin e Bauer 2009]

Entrada		Saída	
Variável	**Valor Linguístico**	λ	δ
Carga do Link	Baixa	Grande	
	Média	Médio	
	Alta	Pequeno	
Largura de Banda Máxima	Baixa	Pequeno	
	Média	Médio	
	Alta	Grande	
Tamanho do Job	Pequeno	Grande	Pequeno
	Médio	Médio	Médio
	Grande	Pequeno	Grande
Tipo de Comunicação entre Processos	master-salve	Grande	Pequeno
	all-all	Pequeno	Grande

Cada uma dessas funções tem como resultado um nível de confidência, e valores linguísticos (por exemplo:*pequeno, médio, grande*) para estes resultados. Então, são utilizadas as regras da Tabela A.1 para cálculo dos valores linguísticos de controle λ e δ.

As variáveis de entrada possuem pesos diferentes no cálculo dos valores de controle, de acordo com a importância do parâmetro no resultado final. Esses pesos são aplicados aos resultados, e por fim, é utilizado o método de centro de gravidade CoG (*center of gravity*[Bai, Zhuang e Wang 2006], [Wang 1997]), com a função da Figura A.1 para gerar o resultado final dos valores de λ e δ.

Apêndice B

Publicações

B.1 Trabalhos Publicados

Título: Grid Computing Middleware Survey

Evento: 7th International Information and Telecommunication Technologies Symposium (I2TS)

Local: Foz do Iguaçu, Paraná, Brasil

Data: 03/12/08 - 05/12/08

Autores: D. J. Ferreira, Cristiano C. da Rocha, Matheus A. Vieira, Rodrigo G. Silva, Miguel L. C. Cartagena, Leandro S. Grapiuna e M.A.R. Dantas

Extrato: B4

Título: Dynamic Resource Matching for Multi-Clusters Based on an Ontology-Fuzzy Approach

Evento: 23th International Symposium on High Performance Computing Systems and Applications (HPCS)

Local: Kingston, Ontario, Canadá

Data: 14/06/09 - 16/06/09

Autores: Denise Janson, Alexandre P.C. Silva, M. A. R. Dantas, Jinhui Qin e Michael A. Bauer

Extrato: B3

Título: Toward Resource Management in Multi-Cluster Grid Configurations through an Ontology-Fuzzy Approach

Evento: International Conference on Grid Computing and Applications (GCA)

Local: Las Vegas, Nevada, EUA

Data: 13/07/09 - 16/07/09

Autores: Denise Janson, Alexandre P. C. Silva, Mario A. R. Dantas, Jinhui Qin e Michael A. Bauer

Extrato: B3

Título: An Ontological-Fuzzy Approach to Advance Reservation in Multi-Cluster Grids (aceito)

Evento: Journal of Physics: Conference Series (JPCS)

Local: Toronto, Ontario, Canadá

Data: 05/06/10 - 09/06/10

Autores: D. J. Ferreira, Mario A. R. Dantas e Michael A. Bauer

Extrato: B4