PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TMRorR: UM NOVO ALGORITMO DE ESCALONAMENTO PARA O OURGRID QUE COMBINA O USO DE INFORMAÇÃO E REPLICAÇÃO

BENEVID FELIX DA SILVA

Dissertação apresentada com requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Cesar A. F. De Rose

Porto Alegre
2009

Dados Internacionais de Catalogação na Publicação (CIP)

S586t Silva, Benevid Felix da
TMRorR : um novo algoritmo de escalonamento para o OurGrid que combina o uso de informação e replicação / Benevid Felix da Silva. – Porto Alegre, 2009.
88 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Cesar A. F. De Rose

1. Informática. 2. Algoritmos. 3. Grade Computacional. I. Rose, Cesar Augusto Fonticiella De Rose. II. Título.

CDD 004.36

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada **"TMRorR: Um Novo Algoritmo de Escalonamento para o Ourgrid que Combina o Uso de Informações e Replicação"**, apresentada por Benevid Felix da Silva, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Processamento Paralelo e Distribuído, aprovada em 06/07/09 pela Comissão Examinadora:

Prof. Dr. César Augusto Fonticielha De Rose– Orientador — PPGCC/PUCRS

Prof. Dr. Luiz Gustavo Leão Fernandes - — PPGCC/PUCRS

Prof. Dr. Francisco Vilar Brasileiro – — UFCG

Homologada em 29/09/09, conforme Ata No 17/09 pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS

Campus Central
Av. Ipiranga, 6681 – P32 – sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320–3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

“Sonha e serás livre de espírito... luta e serás livre na vida.“

(Che Guevara).

AGRADECIMENTOS

Gostaria de agradecer principalmente a Deus, que com Fé consegui ultrapassar os muitos momentos de dificuldades, minha devoção a São Dimas, que esteve sempre em meus pensamentos. Agradeço também aos meus amados pais, Bené e Nenzinha, que sempre me apoiaram nos estudos e deram sempre o máximo de si para que eu aqui chegasse. Agradeço a minha amada esposa, Vanessa, por seu companheirismo, carinho e paciência em estar comigo enfrentando todas as dificuldades, e minhas filhas amadas, Myrella e Mariane, que fazem parte de minhas conquistas. Agradeço a todos os meus irmãos, em especial minha irmã Nilce, que foi meu maior exemplo e me deste os maiores incentivos em relação aos estudos. Agradeço ao meu Orientador, Cesar De Rose, pela paciência e compreensão diante minhas dificuldades. Agradeço aos meus amigos de Unemat e Mestrado, em especial Ivan, André, Tales, Maicon e Toni, que estavam presentes em toda esta caminhada. Agradeço ao Professor Dr. Francisco Brasileiro, pela oportunidade cedida de estar visitando o berço do OurGrid por alguns dias (LSD-UFCG), e aos seus desenvolvedores e agora amigos, Rodrigo Vilar e Diogo Vilar, pelo companheirismo e paciência.

Meu Muito Obrigado.

TMRorR: UM NOVO ALGORITMO DE ESCALONAMENTO PARA O OURGRID QUE COMBINA O USO DE INFORMAÇÃO E REPLICAÇÃO

RESUMO

A distribuição de tarefas de forma eficiente em grades computacionais possui grandes desafios que devem ser considerados por um algoritmo de escalonamento. Esse desafios estão relacionados com as características das grades, como a escalabilidade, heterogeneidade, dinamicidade, entre outros. Dentre as diversas propostas de algoritmos de escalonamento existentes, grande parte utilizam da informação obtida do ambiente ou da replicação de tarefas como forma de atingir um desempenho satisfatório na execução das aplicações dentro da grade. Os algoritmos de escalonamento que dependem somente das informações para realizar o escalonamento estão restritos a ambientes em que as mesmas estejam disponíveis e sejam confiáveis. Já os algoritmos que utilizam somente de replicação não dependem de nenhuma informação. A existência de ambientes em que a presença de informação não é totalmente confiável ou não atende a todos os recursos da grade, motiva o surgimento de algoritmos que utilizam uma técnica híbrida. Neste caso a informação, quando disponível, é utilizada para realizar o escalonamento, porém, quando não, utiliza-se da replicação de tarefas. Este trabalho realiza um estudo acerca do escalonamento de tarefas realizado pelo WQR do OurGrid e apresenta um novo algoritmo para escalonamento de aplicações *Bag-of-Tasks* aqui denominado TMRorR (*Task for More Reliable or Replicate*). Este algoritmo utiliza uma técnica híbrida e, considera sempre que um recurso que possui informação é sempre mais confiável do um que não possui e, também, faz uma distinção entre recursos pertencentes ao domímio local e recursos pertencentes a outros domínios, tendo regras de escalonamento com algumas diferenças para ambos os casos. O algoritmo foi simulado utilizando o GridSim e implementado no OurGrid para realizar os experimentos. Comparando os resultados obtidos, eles mostraram um desepenho melhor do TMRorR em relação WQR na maioria dos casos, como também, uma redução no desperdício de ciclos de CPU com a realização de um controle maior na criação de réplicas.

Palavras Chave: Grade Computacional, Algoritmos de Escalonamento, *Bag-of-Task*, OurGrid.

TMRorR: A new scheduling Algotithm for OurGrid that combines the use of the information and replication

ABSTRACT

An efficient distribution of tasks in computational grids has major challenges that have to be overcomed by a scheduling algorithm. These challenges are related to the characteristics of the environment, such, scalability, heterogeneity, dynamicity, among others. Most of the existing scheduling algorithms use information obtained from the environment or the replication of tasks in order to achieve a satisfactory performance in the execution of applications. Algorithms that rely on resource information to accomplish their task are restricted to environments in which such information is available and reliable all across the grid. On the other hand, algorithms that use replication could be used in any environment but they introduce a significant overhead to the scheduling. However, although reliable information is usually not available across the Grid because of the dynamicity of the resources, it's possible to have updated resource information inside a local site. This motivates the development of algorithms that use a hybrid technique. In this case the information is used to perform the scheduling when available - for example in the local site - and only when it's not available the scheduler uses replication, thus reducing its overhead. This work proposes such an hybrid algorithm called TMRorR - Task More Reliable or Replicate for the Ourgrid middleware. The algorithm was simulated with GridSim and implemented in OurGrid to perform the experiments. Obtained results showed a better performance of TMRorR in comparison to the Ourgrid scheduler in several scenarios and a reduction in the overhead of creating replica.

Keywords: Grid Computing, Scheduling Algorithms, Bag-of-Task, OurGrid.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

BoT	*Bag-of-Task*
DFPTLF	*Dynamic Fast Processor Task Large First*
DNS	*Domain Name Server*
ICMP	*Internet Control Message Protocol*
OV	Organização Virtual
SWAN	*Sandboxing Without A Name*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
TMRorR	*Task More Reliable or Replicate*
UDP	*User Datagram Protocol*
WQ	*Work Queue*
WQR	*Work Queue with Replication*
WQR-FT	*Work Queue with Replication Fault Tolerant*
XMPP	*Extensible Messaging and Presence Protocol*

SUMÁRIO

1 INTRODUÇÃO

Uma grade computacional pode ser compreendida como uma estrutura global e heterogênea de computadores dispersos geograficamente que provê acesso seguro, consistente, transparente e coordenado ao compartilhamento de recursos entre uma coleção dinâmica de indivíduos e instituições [DAN05][CIR07].

Este sistema agrega serviços e recursos computacionais distribuídos geograficamente com objetivo de melhorar e até mesmo viabilizar uma aplicação intangível para os recursos computacionais que estão disponíveis em um site de uma instituição, como *clusters*, supercomputadores, instrumentos científicos, repositórios de dados, dispositivos de visualização dentre outros.

Um site, de acordo com [CIR07], é considerado um dos componentes básicos do ambiente, além de possuir uma administração local, podendo disponibilizar muitos ou até mesmo nenhum recurso em um dado momento. Seus recursos comumente são heterogêneos, baseados em diferentes tipos de hardwares (PCs, supercomputadores), plataformas (sistemas operacionais, arquitetura de CPUs) e softwares (bibliotecas e linguagens de programação). Um conjunto de sites geograficamente dispersos, que compartilham seus recursos, forma uma Organização Virtual (OV) [FOS02a].

Através de uma OV é possível que um ou mais usuários possam submeter suas aplicações paralelas para serem executadas mais rapidamente, devido a um número muito maior de recursos disponíveis. Contudo, é preciso que tenham políticas que coordenem o acesso aos recursos que são compartilhados.

Estas políticas podem ser divididas em escalonadores de recursos e escalonadores de aplicação. Os escalonadores de aplicação solicitam aos escalonadores de recursos os computadores que lhe são necessários para executar a aplicação, e este ultimo, por sua vez, procura atender ao pedido retornando-lhe a quantidade solicitada.

Um usuário da grade ao utilizar um escalonador de aplicação, espera basicamente que, sua aplicação paralela, seja executada nos recursos disponíveis na grade. Contudo, podem existir outras preocupações, como por exemplo, o tempo de execução da aplicação.

Os escalonadores de aplicação muitas vezes precisam atender aos requisitos do usuário, e também, estarem preparados para lidar com os desafios que são característicos das grades, como recursos escalares, dinâmicos e totalmente

heterogêneos. Estes desafios contribuem com o surgimento de escalonadores baseados em diferentes políticas, as quais consideram as características do ambiente.

Dentre os inúmeros escalonadores existentes, este trabalho está relacionado com aqueles que implementam políticas baseadas em: replicação de tarefas, uso de informação do ambiente e híbridas (informação e replicação).

O uso de escalonadores que implementam políticas que utilizam a informação no processo de escalonamento contribui com a realização eficiente do mapeamentos entre tarefas e recursos, porém, em muitos casos, a obtenção destas informações dentro da grade pode ser uma tarefa complexa, devido à escalabilidade e dinamicidade do sistema.

O que pode ocorrer, em muitos casos, é a obtenção parcial de informações dentro da grade. O que pode dificultar a utilização de algoritmos como o *XSufferage* e DFPTLF, apresentados em 4.2.2 e 4.2.3, respectivamente. Estes algoritmos dependem da informação, não sendo possível realizar o escalonamento de tarefas em ambientes nos quais ela seja incompleta.

O uso de escalonadores que utilizam a replicação de tarefas tem demonstrado bons resultados, como exemplo o algoritmo WQR do OurGrid [CIR07]. Ao utilizar a replicação de tarefas, durante a execução de uma aplicação, uma tarefa que já esteja em execução em determinado recurso pode ser replicada em outro que se torne disponível. Com isso, espera-se que, a réplica em execução possa alocar um recurso que seja mais rápido que a tarefa original, sendo executada então em um tempo menor e contribuindo para reduzir o tempo total de execução da aplicação (Makespan).

Os escalonadores que implementam um política híbrida utilizam a replicação de tarefas e também, a informação quando está disponível [NOB08]. As limitações na obtenção da informação permitem que estes escalonadores funcionem basicamente da seguinte forma: se a informação está disponível, utiliza-a para realizar mapeamentos mais eficientes e caso não esteja, replica as tarefas para as máquinas disponíveis.

Neste trabalho, realizou-se então, um estudo de um novo algoritmo de escalonamento para o *middleware* OurGrid, que entre outras coisas, implementou uma política híbrida para realizar o escalonamento de tarefas. O novo algoritmo aqui denominado TMRorR, teve como base o algoritmo de escalonamento que atualmente é utilizado pelo OurGrid, o WQR, juntamente com os demais algoritmos apresentados neste trabalho.

Desta forma, os objetivos do trabalho, descritos na próxima seção, partem dos estudos realizados a cerca do escalonamento do OurGrid [OUR09] e dos algoritmos apresentados na seção 4.2.

1.1 Objetivos do Trabalho

Realizar um estudo acerca da política de escalonamento do OurGrid, o WQR, e, propor uma nova política com os princípios básicos:

- Utilizar as informações sobre os recursos, quando disponível.
- Utilizar a replicação quando a informação não estiver disponível;
- Controlar a criação de réplicas para minimizar o uso desnecessário de processamento.

1.2 Organização do Restante do Documento

Os demais capítulos estão organizados da seguinte forma, o Capitulo 2 apresenta as definições acerca das grades e, no Capítulo 3, é apresentado uma taxonomia muito conhecida para classificação de algoritmos de escalonamento em sistemas distribuídos sob uma visão específica de algoritmos para grades, os principais desafios que atualmente são encontrados nestes ambientes e alguns algoritmos que estão relacionados com este trabalho, no Capitulo 4, é descrito o middleware OurGrid, no Capítulo 5, é apresentado o algoritmo de escalonamento proposto neste estudo, o TMRorR. No Capitulo 6, apresenta-se os resultados das simulações e dos experimentos que foram realizados, e por fim, no capitulo 6, são apresentadas as conclusão deste estudo e trabalhos futuros.

2 AMBIENTES DE GRADE

Uma grade possui uma analogia à rede de distribuição de energia elétrica convencional. Onde um determinado usuário faz uso do recurso (energia elétrica) sem ter a necessidade da localização do mesmo (fonte distribuidora). Em uma grade os recursos ao invés de energia elétrica podem envolver computadores, instrumentos científicos, entre outros, e os usuários possuem acesso transparente aos mesmos.

Muitas vezes o termo grade é utilizado de forma equívoca, segundo Dantas [DAN05], o simples fato de possuir uma pequena rede de computadores interligados par-a-par (i.e. *peer to peer*) ou disponibilizar serviços para usuários via rede internet não conceitua-se como uma grade computacional. Este sistema é algo muito mais complexo que envolve uma série de características inerentes ao ambiente constituído de forma geograficamente dispersa.

Não se pode confundir uma grade com um ou mais *clusters* de computadores, pois estes possuem uma única administração central e os computadores trabalham como uma única entidade física pertencente a uma instituição ou domínio.

Um ambiente de grade pode agregar recursos geograficamente distribuídos, incluindo *clusters*, supercomputadores, instrumentos científico, repositórios de dados, dispositivos de visualização dentre outros.

Um site é um componente básico do ambiente, possui uma administração local e pode, ainda, disponibilizar muitos ou até mesmo nenhum recurso em um dado momento. Seus recursos comumente são heterogêneos, baseados em diferentes tipos de hardwares (e.g computadores pessoais, supercomputadores, etc.), plataformas (e.g sistemas operacionais, arquitetura de CPUs, etc.) e softwares (e.g bibliotecas e linguagens de programação). Um conjunto de sites geograficamente dispersos que compartilham seus recursos formam uma Organização Virtual (OV) [FOS02a].

Em uma OV não se pode obter uma quantidade constante de recursos durante um grande período de tempo. Isto deve-se ao modo como eles são disponibilizados em cada site. Em muitos ambientes, os recursos estão disponibilizados de forma compartilhada entre as aplicações locais tornando, muitas vezes, indisponíveis para serem utilizadas por aplicações remotas.

Mesmo a grade sendo um ambiente complexo, durante a execução de determinadas aplicações, espera-se uma redução no tempo de execução das mesmas, afim de justificar sua utilização.

A seguir, demonstra-se alguns dos conceitos que determinam as características desses ambientes.

2.1 Sistemas de Grade

Existe um esforço em classificar diferentes tipos de grade de acordo com suas principais funcionalidades, mas ainda não existe uma padronização. Buyya et al. [BUY02] considera as categorias demonstradas a seguir.

2.1.1.1 Grade Computacional

Algumas aplicações não são passíveis de serem executadas em apenas um computador ou supercomputador existente atualmente, uma vez que elas dependem de sistemas que agregam grandes quantidades de recursos computacionais para sua execução. Para este tipo de recursos existem duas subcategorias, baseadas na forma como são utilizados os recursos provenientes da grade, aqui interpretados como Super-Computação Distribuída e Computação Intensiva [CIR03]. As aplicações que executam sob a primeira subcategoria são executadas paralelamente em múltiplos computadores devido à necessidade de reduzir o tempo de execução das tarefas e, possuem todos os recursos disponíveis dedicados para sua execução. Alguns exemplos destas aplicações são as simulações nucleares e predição de fenômenos naturais [CIR03].

As aplicações sob a segunda categoria trabalham com o fluxo de grande quantidade de dados durante sua execução. O ambiente computacional desta subcategoria precisa escalonar um número muito grande de tarefas fracamente acopladas, com intuito de melhorar ou aumentar a utilização dos computadores com processadores ociosos. Como exemplo de aplicações pertencentes a esta abordagem tem-se a renderização de imagens e o reconhecimento de padrões.

2.1.1.2 Grade de Dados

Uma grade de dados visa possibilitar que uma enorme quantidade de dados pertencentes a uma aplicação sejam manipulados de forma transparente, como se estivesse em um ambiente local. Exemplos desta categoria são os bancos de dados distribuídos e as bibliotecas digitais.

As grades computacionais também são utilizadas para prover serviços de dados, porém, existe uma grande diferença entre elas. De acordo com [BUY02], uma grade de

dados deve possuir a infra-estrutura necessária para possibilitar a utilização de aplicações que gerenciam o armazenamento e acesso aos dados.

2.1.1.3 Grade de Serviços

Uma aplicação que pertencente a esta abordagem, pode-se dizer que trabalha a partir de um paradigma de prestação de serviços computacionais. Os computadores que pertencem a esta categoria, muitas vezes, estão disponibilizando voluntariamente seus recursos computacionais ou buscam os que estão disponíveis em um site de outra instituição. A categoria é dividida em três sub-categorias de acordo com suas funcionalidades: sob-demanda, colaborativo e multimídia.

Em um ambiente sob-demanda, uma determinada aplicação pode dinamicamente alocar recursos de acordo com sua necessidade. Estes recursos são disponibilizados de forma colaborativa e compreendem além de processamento, os dados, os softwares e os instrumentos científicos.

Um ambiente colaborativo reúne pessoas e instituições que, de forma espontânea, compartilham seus recursos computacionais ociosos para programas de pesquisas científicas ou trabalham em um mesmo projeto compartilhando informações.

Um ambiente multimídia, por sua vez, é utilizado para prover infra-estrutura com QoS para execução de diversas aplicações multimídia concorrentes. Significa, então, que uma aplicação pode ter inúmeros acessos simultâneos mantendo um padrão de qualidade que é provido pela arquitetura de um sistema.

Mesmo que haja outras classificações para os ambientes de grade, optou-se pelas acima abordadas em função da filiação teórica assumida neste trabalho, como não existe um padrão bem definido optou-se em trabalhar com esses três tipos, uma vez que, os autores citados são umas das referências nessa área.

2.2 Arquitetura e Topologia de Grades

Em busca de uma padronização acerca das definições citadas anteriormente e de uma delimitação de uma arquitetura que permita a interoperabilidade entre os domínios que fazem parte de uma OV, filia-se a proposição de Ian Foster [FOS01].

Esta proposta considera os diversos fatores que são fundamentais em uma grade como autenticação, autorização, mecanismos de troca de mensagens, compartilhamento de recursos, escalonamento e balanceamento de tarefas.

Figura 1 – Arquitetura em Camadas proposta por Ian Foster.

A definição de Ian Foster possui como abordagem a divisão da composição de um grade em camadas, mostrada na Figura 1, a qual faz uma analogia à arquitetura TCP/IP utilizada largamente nas redes locais e Internet. Apresenta-se, a seguir, uma explanação mesmo que sucinta de cada uma delas.

2.2.1.1 Camada Fábrica

Esta camada fornece as funcionalidades necessárias para o compartilhamento de recursos na grade. Através dela é que se implementam as operações locais e específicas para cada tipo de recurso e as políticas de acesso aos mesmos. Outros mecanismos necessários são os que buscam informações e gerenciam a utilização dos recursos visando monitorar a qualidade de serviço.

A camada implementa funções específicas para cada tipo de recurso possível disponibilizado na grade:

- Recursos computacionais - mecanismos para iniciar e monitorar a execução de aplicações. Possibilita o gerenciamento de recursos alocados para determinada aplicação e também provê informações sobre o hardware, software e carga de processamento.
- Recursos de armazenamento - mecanismos de gerenciamento que permitam o controle sobre os recursos alocados para a transferência de dados (espaço, taxa de transferência do disco e da rede, CPU) e informações sobre as características de hardware, software, armazenamento, e taxa de transferência.
- Recursos de rede - permite o gerenciamento sobre os recursos alocados para transferências de rede e fornece informações sobre as características da rede e sua carga.

2.2.1.2 Camada Conectividade

Aqui são definidos os protocolos que envolvem a comunicação e a autenticação nas transações de rede. Enquanto os protocolos de comunicação são responsáveis pelo acesso aos recursos disponibilizados na camada mais inferior do modelo (Fábrica), os protocolos de autenticação são responsáveis por prover mecanismos de segurança na disponibilização e utilização dos recursos pelos usuários.

A camada é responsável pelos requisitos de comunicação, incluindo o transporte, roteamento e o servidor de nomes. Geralmente, estes protocolos são mapeados para a pilha de protocolos TCP/IP, especificamente, pela rede (IP e ICMP), pelo transporte (TCP e UDP) e pela aplicação (DNS).

2.2.1.3 Camada Recurso

Esta camada utiliza as funcionalidades da camada Conectividade para negociação segura, monitoramento, controle e contabilização de operações de compartilhamento em recursos individuais. Ela também utiliza a camada Fabrica para acessar e controlar os dispositivos locais. As implementações dos protocolos nesta camada são divididas em duas classes:

- Protocolos de informação - que buscam obter informações a respeito da estrutura e do estado dos recursos, como a configuração, carga atual e políticas de uso.
- Protocolos de gerenciamento - os quais são utilizados para negociar acesso a um dado recurso compartilhado, especificando parâmetros de qualidade de serviço e permissões de acesso aos dados na execução de uma tarefa.

2.2.1.4 Camada Coleção

Esta camada fornece protocolos e serviços que não estão associados a um recurso específico, mas a uma coleção destes. Possui protocolos e serviços que atuam nas interações entre sites. Os componentes desta camada baseiam-se nos níveis de recursos e aplicação, provendo serviços como:

- Serviços de diretório - facilita o funcionamento das OVs, pois estas passam a descobrir a existência e as prioridades dos recursos compartilhados. Estes recursos podem ser localizados por nome e/ou outros atributos.
- Serviço de alocação - permite que recursos possam ser alocados para determinado propósito e o agendamento de tarefas em recursos apropriados.

• Serviço de monitoramento - os recursos podem ser monitorados em relação a falhas, sobrecarga, acessos não autorizados, etc.

• Serviço de replicação de dados - suporte ao gerenciamento dos recursos de armazenamento para maximizar a performance de acesso a dados. Por exemplo, pode-se monitorar o tempo de resposta, confiabilidade e custos.

• Sistema de programação específico - possibilita a utilização de modelos de programação conhecidos no ambiente de grade (e.g MPI - *Message Passing Interface*), usando serviços como busca e alocação de recursos e segurança. Um fator importante nesta camada é que os protocolos e interfaces são baseados em padrões abertos para facilitar a integração do ambiente.

2.2.1.5 Camada Aplicação

Os programas desenvolvidos pelos usuários e executados dentro de uma OV estão nesta camada. As aplicações são construídas através da utilização de serviços providos pelas camadas inferiores, uma a uma. Em todas as camadas existem protocolos definidos que fornecem serviços, como gerenciamento de recursos, localização, acesso a dados, dentre outros.

As facilidades providas pela separação em camadas do modelo proposto são equivalentes as do modelo TCP/IP, tanto no desenvolvimento de aplicações quanto para uma melhor visualização das funcionalidades.

2.3 Conclusões do Capítulo

A busca pela padronização de um modelo que atenda a todos os preceitos de uma arquitetura complexa como as grades tem aproximadamente dez anos de discussão[DAN05]. Portanto, dentre as propostas existentes, foi citado o modelo proposto por Foster et al. [FOS02b], por fazer parte de inúmeras discussões entre as comunidades acadêmicas e por ser a base de uns dos principais projetos de *middleware* de grade existente, o Globus *Toolkit* [GTK09].

O próximo capítulo demonstra algumas das características e conceitos que envolvem o escalonamento em grades. E, também, os desafios próprios e característicos deste ambiente, além de alguns algoritmos que, exemplificam o uso de técnicas que são utilizadas para obter desempenho satisfatório na execução das aplicações em grades e, além disso, serviram de base para o estudo do algoritmo o qual está sendo proposto neste trabalho.

3 ESCALONAMENTO DE TAREFAS EM GRADE

Há muito se discute o escalonamento de tarefas em sistemas paralelos e distribuídos, porém a abordagem de escalonamento em grades computacionais é recente, devido às características de escalabilidade e à capacidade de agregar recursos heterogêneos em escala global.

O escalonamento em um sistema distribuído consiste na alocação de tarefas de uma aplicação baseando-se ou não em informações sobre os recursos e, também, em gerenciar o envio e recebimento de tarefas durante a execução da aplicação. Registra-se, contudo, que esse tipo de escalonamento não difere do escalonamento em grades.

O desenvolvimento de políticas de escalonamento para grades computacionais constitui-se como uma grande área de pesquisa, devidos aos grandes desafios característicos deste ambiente. Uma política de escalonamento define um conjunto de regras, dizendo, por exemplo, quando e como obter informações dos recursos, como será realizada a distribuição das tarefas e quais recursos serão utilizados. Para implementar as regras de uma política são construídos os algoritmos de escalonamento [CIR07].

Uma política de escalonamento pode objetivar um bom desempenho na execução de uma aplicação sobre um determinado conjunto de recursos [SIL05]. Neste caso, como exemplo, considere-se uma aplicação paralela, a qual pode ser dividida em tarefas. Ela será escalonada para os processadores disponíveis de modo que o desempenho seria consideravelmente melhor que a execução com um só processador.

As políticas são muitas vezes denominadas de escalonadores de aplicação, que são responsáveis apenas pela execução da aplicação na grade, não controlam os recursos, apenas solicitam-nos aos chamados gerentes ou escalonadores de recursos.

Um escalonador de recursos é responsável por gerenciar o uso dos recursos entre os usuários e, neste sentido, exerce o controle ao definir quais estão disponíveis ou quais serão disponibilizados em determinado momento [CAL06].

Um escalonador de aplicação precisa dizer quais tarefas cada recurso executará, para então, realizar a execução da aplicação. Para utilizar os recursos ele solicita ao escalonador de recursos.

Muitas políticas de escalonamento existentes em grades são derivadas de políticas utilizadas em sistemas distribuídos, como exemplo o WQR que teve como base o WQ (*Work Queue*) [SIL03]. Isso faz com que o desenvolvimento de algoritmos específicos que estejam de acordo com as características das grades tenham grande relevância e

também possuam desempenho significativamente melhor quando comparados aos tradicionais.

Os algoritmos de grade, além de considerar o ambiente, também precisam, em muitos casos, considerar as características de um conjunto de aplicações específicas. Como exemplo, o WQR [SIL03], que é específico para aplicações do tipo BoT (*Bag-of-Task*). As aplicações do tipo BoT são independentes, ou seja, não existe uma ordem de dependência no processo de execução de suas tarefas, podendo ser executadas de forma aleatória.

Na seção seguinte, demonstra-se uma taxonomia proposta por [CAS88] que inicialmente classifica os algoritmos de sistemas distribuídos. Contudo, a apresentação pretende dar ênfase aos algoritmos de grade, de acordo com a proposta de [DON06].

3.1 Taxonomia para Algoritmos de Escalonamento

Casavant e Kuhl em [CAS88], ainda em 1988, propuseram uma taxonomia hierárquica para algoritmos de escalonamento em sistemas computacionais paralelos e distribuídos. A taxonomia é dita hierárquica, porque ordena as características de escalonamento em grupos. Dong [DON06], por sua vez, ressaltou neste modelo as classificações que são específicas para os algoritmos de grade. Na Figura 2, este subconjunto pode ser identificado pelos galhos em negrito/itálico.

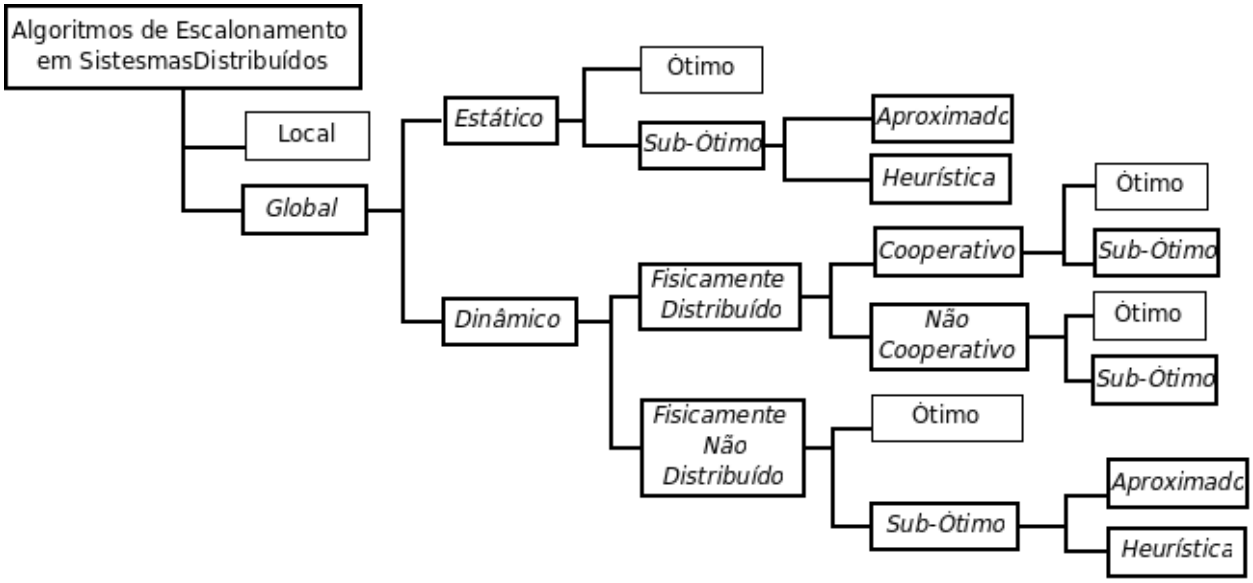

Figura 2 – Taxonomia Hierárquica para Algoritmos de Escalonamento

3.1.1 Local vs. Global

Em um nível mais alto, a distinção está delineada entre escalonamento local e global. O escalonamento local determina como os processos residentes em um único CPU (Processador) são alocados e executados enquanto que um escalonamento global utiliza-se de informações sobre o sistema para alocar processos para múltiplos processadores, para assim otimizar um sistema que objetiva mais desempenho. Certamente que uma grade configura-se dentro de um escalonamento global.

3.1.2 Estático vs. Dinâmico

O próximo nível da hierarquia (abaixo do escalonamento global) é a escolha entre escalonador estático e dinâmico. Esta escolha indica o tempo em que as decisões de escalonamento são tomadas.

No caso do escalonamento estático, as informações são obtidas antes do escalonamento da aplicação, pois não se obtém informações sobre as mudanças dinâmicas de estado do sistema que ocorrem durante o processo.

No caso do escalonamento dinâmico, a idéia básica é que se ofereça alocação de tarefas durante a execução da aplicação. Deste modo, é responsabilidade do sistema decidir onde o processo será executado.

O escalonamento dinâmico é usualmente aplicado quando se tem dificuldade em estimar o custo da aplicação e, também quando as tarefas da aplicação estão chegando dinamicamente em tempo real.

Tendo em vista os trabalhos de [REI05][SHI92], escalonamentos dinâmicos possuem quatro componentes básicos: uma política de transferência, uma de seleção, uma de localização e uma de informação. Sendo assim definidas:

- Política de transferência - é responsável por determinar se um recurso está em condições de participar de uma migração de processos, seja para receber uma nova tarefa, seja para transferir um de seus processos em execução.
- Política de seleção - uma vez determinado o recurso emissor de processos, deve-se escolher qual de seus processos deve ser transferido.
- Política de localização - sua função é encontrar recursos receptores de processos para os melhores emissores e vice-versa.
- Política de informação - determina quando e onde devem ser colhidos dados a respeito dos estados de outros recursos.

Tanto os algoritmos de escalonamento estático quanto dinâmico são largamente adotados em grades computacionais. Como exemplo, algoritmos estáticos são estudados em [BRA01] e dinâmicos estudados em [CHE02].

3.1.3 Ótimo vs. Sub-ótimo

No caso em que toda a informação do estado dos recursos e da aplicação é conhecida, tem-se, em alguns casos, um escalonamento Ótimo. Muitas vezes, no entanto, ter toda a informação é algo inalcançável, por exemplo, a dificuldade em estimar o tempo de execução de uma aplicação com precisão, e também, estimar a carga dos recursos selecionados que em muitos casos pode sofrer grandes variações.

Por dificuldades como estas que os algoritmos de escalonamento, geralmente, variam entre resultados Sub-ótimos que, por sua vez, podem ser obtidos através de Aproximação ou da Heurística.

3.1.4 Aproximação vs. Heurística

Um algoritmo de aproximação consiste em um algoritmo o qual segue as mesmas regras de um algoritmo ótimo, com a diferença de que não é necessário encontrar uma solução ótima, bastando uma solução que seja suficientemente boa em relação a que foi previamente definida como ótima.

Uma heurística consiste em um algoritmo que leva em consideração alguns parâmetros que afetam o sistema de uma maneira indireta. Esses parâmetros, porém, são mais simples de se calcular do que os parâmetros verdadeiros para análise de desempenho, por exemplo. É nesse ponto que a heurística representa boa alternativa para o escalonamento de processos.

3.1.5 Distribuído vs. Não-distribuído

No cenário dos escalonadores dinâmicos, as tomadas de decisões de um escalonamento global podem ser de um escalonador centralizado ou pode ser compartilhada por múltiplos escalonadores distribuídos. Em uma grade computacional, muitas aplicações podem ser enviadas ou requeridas para serem escalonadas simultaneamente, o que pode inviabilizar o uso de um escalonador centralizado.

Uma estratégia centralizada tem a vantagem de ser fácil de implementar, porém tem a possibilidade de virar um gargalo do desempenho.

3.1.6 Cooperativo vs. Não-Cooperativo

Caso seja adotado um algoritmo de escalonamento distribuído, a seguinte questão que precisa ser considerada é se todos os escalonadores envolvidos com o escalonamento da aplicação estão trabalhando de forma cooperativa ou independente (não-cooperativa). No modo não-cooperativo, os escalonadores individuais agem sozinhos como entidades independentes que não se preocupam diretamente com nenhum objetivo que resulte na melhoria do desempenho do resto do sistema. As decisões afetam somente o desempenho local de cada escalonador.

No modo cooperativo, cada escalonador responsabiliza-se por carregar fora sua própria porção do escalonamento da tarefa, mas todos os escalonadores estão trabalhando em comum para um amplo sistema global.

3.2 Aspectos Restritos às Grades Computacionais

A taxonomia hierárquica classifica os algoritmos de escalonamento do ponto de vista do sistema, como dinâmico ou estático ,distribuído ou centralizado, etc. Existem, porém, muitos outros aspectos demonstrados por [DON06] que não são cobertos por esta classificação e que estão sendo atualmente acoplados às novas mudanças no cenário de grades computacionais.

Alguns dos aspectos que recentemente estão sendo abordados são os objetivos funcionais, os algoritmos adaptativos e probabilísticos, a dependência entre as tarefas, os grandes volumes de dados durante o escalonamento e os requisitos de QoS. Apresentaremos, a seguir, cada um desses aspectos.

3.2.1 Objetivos Funcionais

Os algoritmos de escalonamento também podem ser classificados de acordo com o objetivo funcional do escalonamento e podem ser classificados em duas categorias: centralizado nos recursos e centralizado nas aplicações.

Os algoritmos de escalonamento centralizados nas aplicações têm por finalidade otimizar o desempenho de cada aplicação individualmente e, os centralizados nos recursos visam otimizar o desempenho e utilização destes.

3.2.2 Adaptativo e Probabilístico

Um algoritmo adaptativo é classificado como sendo aquele no qual os parâmetros utilizados em sua implementação mudam dinamicamente de acordo com o comportamento do sistema em resposta à política de escalonamento adotada. Um algoritmo que, durante a execução, altera o grau de importância de um parâmetro por achar que ele não está expressando corretamente o que deveria ser feito, de acordo com a política de escalonamento utilizada, é dito adaptativo. Logo, um algoritmo que sempre considera a mesma estrutura e grau de ponderação entre os parâmetros é dito não-adaptativo.

Um algoritmo probabilístico consiste na idéia de que, a partir do dado inicial, gerar aleatoriamente diversas soluções de um problema e, dentre estas soluções, selecionar a melhor. Neste método, o número de soluções geradas deve ser bastante para possibilitar que, entre o conjunto apresentado, tenha pelo menos alguma solução que aproxime da solução ótima.

3.2.3 Dependência e Não Dependência entre as Tarefas

Quando se considera a relação entre as tarefas de uma aplicação da grade, é comum encontrar a separação entre dependência e independência. Usualmente, dependência significa que possui uma determinada ordem entre as tarefas, e que, uma tarefa não pode iniciar antes que termine a outra com a qual possui relação de dependência.

As tarefas independentes não possuem relação de dependência e podem ser executadas em qualquer ordem. As aplicações denominadas *Bag-of-Task* são também do tipo independente. Este tipo de aplicação facilita a utilização em ambientes distribuídos geograficamente dispersos, como uma grade computacional.

3.2.4 Aplicações que Trabalham com muitos Dados

Existem aplicações que envolvem a geração de um enorme conjunto de dados numa escala de crescimento muito grande e, além disso, requerem armazenamento e sistemas de gerenciamento especializados.

Em uma grade, a localização onde uma aplicação será processada é usualmente selecionada em tempo real. Desse modo, o custo para transferir os dados de entrada de

onde estão armazenados para onde serão processados provavelmente varia dependendo de quais sites serão utilizados.

Em [NET04] é proposto um algoritmo de escalonamento de aplicações BoT que processam grandes quantidades de dados, chamado *Storage Affinity*.

3.2.5 Modelos não Tradicionais

Uma grade é uma espécie de sistema feito para um grande número de provedores autônomos de recursos e consumidores, os quais estão executando concorrentemente, mudando dinamicamente, interagindo com cada outro, porém com regras próprias [DON06]. Assim como na natureza e na sociedade humana, alguns sistemas possuem características similares. Em vista disso, novos modelos estão sendo propostos direcionados para as mudanças em grades computacionais, como economia de grades [BUY05] e outros modelos que são inspirados em fenômenos da natureza.

O uso dos métodos econômicos em escalonamento envolve um processo interativo entre o provedor de recursos e o usuário, de forma análoga a vários comportamentos de mercado, como barganha, oferta, leilão, dentre outros.

Os métodos de escalonamento baseados nas leis da natureza têm sido utilizados para melhorar os modelos existentes baseando-se em fenômenos naturais e tem, atualmente, demonstrado grande sucesso [ABR00].

3.2.6 Limitação de QoS

Em ambientes distribuídos heterogêneos não dedicados, requisitos de QoS (Quality of Service) se configuram como uma grande preocupação de muitas aplicações. A intenção em ter QoS pode variar de acordo com a preocupação de diferentes usuários. Ele pode ser um requisito de velocidade de CPU, tamanho de memória, largura de banda, versão de software ou prazo final. Em geral, QoS não é o último objetivo de uma aplicação, mas um conjunto de condições para executar uma aplicação bem sucedida.

3.3 Conclusões do Capítulo

As classificações apresentadas neste capítulo podem não atender a todos os algoritmos de escalonamento de grades existentes, porém, a maioria deles possui alguma característica relacionada com uma ou mais classificações apresentadas.

4 DESAFIOS NO ESCALONAMENTO EM GRADES

Diante dos grandes desafios encontrados em um ambiente de grade computacional quando se pretende desenvolver uma política de escalonamento, os objetivos propostos são os principais elementos que devem ser traduzidos para essa política [REI05].

O objetivo primordial do escalonamento de aplicações paralelas é garantir que as aplicações possam iniciar e terminar sua execução paralelamente. Apesar de este objetivo ser algo muito comum, os objetivos adicionais não podem prejudicá-lo e ainda precisam garantir que ele ocorra.

Os demais objetivos podem ser diversos, mas comumente envolvem:

- *Minimizar o tempo de resposta (Makespan*) - o tempo de resposta é o tempo de execução de uma aplicação, desde sua chegada até o término. Uma diminuição no tempo de resposta significa que a aplicação foi executada mais rapidamente.
- *Maximizar o throughput* - o *throughput* é o número de tarefas processadas por unidade de tempo. Esse objetivo está relacionado à exploração da capacidade do sistema.
- *Maximizar a utilização dos recursos* - esse objetivo representa o quanto da capacidade de um recurso está sendo utilizada e o quanto ainda pode ser aumentada para evitar que fique ocioso. Em um ambiente ideal, durante a execução de uma aplicação não se pode ter recursos ociosos.

Estes objetivos, muitas vezes, esbarram em desafios como a enorme quantidade de recursos que fazem parte do ambiente, sua diversidade ou heterogeneidade e o compartilhamento com outras aplicações.

E para que as tarefas de uma aplicação possam ser escalonadas de forma a serem executadas com eficiência estes problemas precisam ser tratados. As próximas seções detalham alguns dos principais desafios que devem ser considerados por uma política de escalonamento.

4.1.1 Escalabilidade

Uma das principais características de uma grade computacional é a possibilidade dela crescer indefinidamente, ou seja, pode-se facilmente agregar ao ambiente em um número muito grande, novos recursos. Cita-se, como exemplo de uma grade computacional com estas características, o projeto SETI@home [SET08], onde existem

milhões de computadores disponibilizando seus recursos ociosos para pesquisadores que buscam sinais de vida em outros planetas.

A grande quantidade de recursos faz com que o escalonador, caso adote um modelo centralizado, se converta em um gargalo para o sistema, não sendo possível atender a todos os computadores disponíveis.

O problema tende a ficar ainda mais crítico quando existe a necessidade de colher informações frequentes sobre o estado dos recursos.

4.1.2 Heterogeneidade dos Recursos

Em uma grade existe a possibilidade de que seja concentrado uma enorme quantidade e variedade de recursos, para então, serem utilizados na execução de uma aplicação. Dependendo da aplicação, pode-se abranger muito mais que somente recursos computacionais, como na abordagem da grade de serviços [FOS02b][BUY05], em que os recursos envolvem instrumentos científicos, computacionais, etc.

Quando se refere às grades computacionais, esta heterogeneidade encontra-se na arquitetura dos computadores, no sistema operacional, na velocidade do processador e da memória, na capacidade de armazenamento, na latência da rede dentre muitos outros.

A heterogeneidade é um fator crítico e determina o tipo de aplicação que certo ambiente pode ter melhor desempenho. Considera-se como exemplo um ambiente em que os processadores possuem bom desempenho, mas a rede de comunicação entre os recursos não possui. Assim sendo, dificilmente uma aplicação que necessita transferir grandes quantidades de dados durante a execução das tarefas terá resultados satisfatórios.

No entanto, nem sempre este fator é um problema, pois isto torna o ambiente mais dinâmico e faz com que o escalonador envie as tarefas para os recursos que julgar mais condizentes com o tipo da aplicação.

4.1.3 Compartilhamento de Recursos

Em uma grade computacional, devido sua grande heterogeneidade, há recursos que não são dedicados ao ambiente, o que provoca, muitas vezes, a preempção durante a execução de uma tarefa. Isto ocorre porque em um recurso compartilhado geralmente a prioridade é local. Quando determinada tarefa de uma aplicação remota está utilizando o recurso de um site pertencente a um domínio local e, uma tarefa de uma aplicação local é

encaminhada, ocorre a preempção da tarefa remota, porque a tarefa local possui maior prioridade.

Em ambientes em que mais de um usuário submete aplicações para serem executadas, ou, os computadores são compartilhados, pode ocorrer uma variação frequente nas características dinâmicas dos recursos, tornando mais complexo o escalonamento, principalmente quando se utiliza de políticas que não possuem um tratamento adequado para este tipo de alteração.

O escalonador precisa balancear a carga a ser distribuída para os recursos para evitar ou minimizar este problema. Uma técnica utilizada por muitas políticas é o uso de informações dinâmicas, que auxiliam em uma melhor distribuição das tarefas da aplicação entre os recursos. As informações podem estar relacionadas com a carga do sistema, à rede de comunicação, velocidade dos processadores, memória, dentre outros.

4.1.4 Transferência de Dados

Durante a execução de determinadas aplicações, existe uma grande quantidade de dados que precisam ser transferidos para os recursos nos quais suas tarefas serão executadas [NET04].

Ocorre também o que se chama de movimentação de dados, quando para obter ganho de performance no processamento, movimenta-se uma aplicação de um recurso para outro. Claro que a movimentação só ocorre se o tempo gasto com a movimentação somado com o tempo que o novo processador levará para executá-la for consideravelmente maior que o tempo gasto no estado local.

4.1.5 Integração dos Resultados

Após o particionamento de uma aplicação em diversas tarefas e encaminhadas para processamento, o próximo passo é organizar o recebimento das tarefas processadas e, por último, produzir o resultado final com a junção de todas as tarefas.

O grande problema neste processo, como citado anteriormente por meio do exemplo do SETI@Home, é o grande volume de tarefas que uma aplicação pode possuir, ou também a necessidade de expressivo processamento durante o recebimento e junção das tarefas para produzir o resultado, provocando assim uma sobrecarga no escalonador ou congestionamento na rede.

4.2 Algoritmos de Escalonamento em Grades

Alguns algoritmos de escalonamento tradicionais são utilizados em ambientes de grades computacionais, como o *Worqueue* e *Round-Robin*. Apesar de não ter um desempenho satisfatório nesse ambiente, eles servem como base para o desenvolvimento de políticas de escalonamento mais robustas e adaptadas com as características do ambiente e das aplicações.

De acordo com a taxonomia proposta por Casavant e Kuhl [CAS88], algoritmos que implementam escalonamento estático tomam todas as decisões relacionadas ao escalonamento antes de iniciar a execução da aplicação. Já algoritmos que implementam escalonamento dinâmico, algumas ou todas as suas decisões são realizadas durante a execução da aplicação.

A variação de carga nos recursos utilizados para execução de uma aplicação pode afetar significativamente o seu *Makespan*, caso a política de escalonamento realize o mapeamento somente de maneira estática. Da mesma forma, atribuir tarefas dinamicamente à grade, não considerando estas variações, pode ter os mesmos resultados. Ademais, a atualização frequente do estado dos recursos a cada nova atribuição de tarefa pode causar uma sobrecarga na comunicação do sistema.

Os algoritmos apresentados a seguir realizam o escalonamento dinâmico, por meio da utilização de informação sobre os recursos, sobre a aplicação, ou, lançando-se mão da replicação de tarefas quando a informação não estiver disponível.

4.2.1 Adaptive WQR

Este algoritmo, proposto por Nobrega [NOB08][NOB06], aproveita as vantagens do uso de replicação no escalonamento de tarefas, e também, as vantagens dos escalonadores que utilizam informações sobre os recursos e tarefas (denominados *bin-packing schedulers*) para tomar as decisões do escalonamento. Desta forma, acredita-se que a solução proposta seja adaptativa ao ambiente de grade quanto à disponibilidade de informações.

Ao utilizar-se de qualquer informação disponível, esta abordagem procura a melhor maneira para lidar com o dinamismo e a heterogeneidade do ambiente de grade.

Os recursos são divididos em dois grupos, a saber: os que possuem informação disponível e os que não possuem. Desta forma, as tarefas que são escalonadas para um

grupo não podem ser replicadas em outro. Para o processo de replicação são determinados os seguintes critérios:

- *Informação disponível para a Aplicação e Recursos*: Uma tarefa é replicada somente quando o seu tempo restante para terminar o processamento é maior que o tempo para executar a nova réplica no recurso disponível.
- *Informação disponível para Aplicação e indisponível para os Recursos*: As maiores tarefas são escalonadas inicialmente, e o processo de replicação ocorre idêntico ao WQR.
- *Informação indisponível para Aplicação e disponível para os Recursos*: A replicação de uma tarefa só acontece quando o recurso disponível possuir um desempenho melhor que o recurso no qual a tarefa está sendo executada.
- *Informação indisponível para Aplicação e Recursos*: funcionamento idêntico ao WQR.

O algoritmo *Adaptive* WQR procura minimizar o desperdício de processamento somente quando existe a disponibilidade de informação sobre os recursos ou tarefas.

4.2.2 XSufferage

Este algoritmo é baseado em heurísticas que dependem de informação disponível para realizar o escalonamento. Casanova et. al [], ao realizar a modificação de heurísticas dependentes de informação utilizadas em sistemas homogêneos, obtiveram melhor desempenho em ambientes heterogêneos. Baseando-se, então, no algoritmo Sufferage[CAS00], os referidos autores adaptaram-no e criaram a extensão chamada XSuferage.

O algoritmo original, o *Sufferage*, possui uma heurística que considera que determinada tarefa deve ser executada na máquina em que sofreria menos (menor tempo de execução). O *Sufferage* é o valor da diferença entre o melhor MCT (*Minimum Completion Time*) e o segundo melhor MCT de cada tarefa.

O algoritmo *XSufferage* adiciona também o MCT em nível de cluster, em que é computado o mínimo MCT de todos as máquinas do cluster. Deste modo, uma tarefa é enviada para o cluster que possuir o menor MCT, escolhendo também a máquina com o menor MCT. Adicionalmente, utiliza a idéia de que uma tarefa sofreria menos também, se for escalonada para um máquina ou *cluster* que já possuir uma porção maior dos dados de entrada de que ela necessita.

O *XSufferage* depende de informação tanto da tarefa, quanto dos recursos, restringindo sua utilização para ambientes que possuam estas informações disponíveis.

4.2.3 Dynamic FPTLF

O dinamismo e a heterogeneidade dos recursos presentes em uma grade levaram ao desenvolvimento de algoritmos dinâmicos como *Dynamic* FPLTF. Esta política é baseada na FPLTF(*Fastest Processor to Largest Task First*) e foi apresentada em [SIL03]. Como o próprio nome sugere, é muito similar a política Max-min2x[RSM05], em que as tarefas maiores, que necessitam de mais tempo de processamento, são encaminhadas primeiro aos processadores mais rápidos. A grande diferença é que isto é realizado de forma dinâmica pelo DFPLTF, o que torna possível sua utilização em ambientes dinâmicos como as grades computacionais.

O DFPLTF é uma política que precisa de informações para escalonar devidamente as tarefas. Ele necessita de três tipos de informações: Tamanho da Tarefa *(Task Size)*, Carga da Máquina *(Host Load)*, e Velocidade da Máquina *(Host speed)*. O valor de *Host Speed* possui um valor relativo, ou seja, uma máquina com velocidade de *Host Speed* = 2 executa uma tarefa duas vezes mais rápido do que uma com *Host Speed = 1*. O *Host Load* representa a parte da máquina que não está disponível para a aplicação. E *Task Size* representa o tempo necessário para uma máquina completar a execução de uma tarefa quando possui *Host Speed = 1*.

No início da execução, o algoritmo inicializa uma variável TBA (*Time to Become Avaiable*) para cada máquina com o valor zero (i.e TBA = 0), e as tarefas são organizadas em ordem decrescente. Uma tarefa é alocada à máquina que lhe oferecer o melhor tempo de execução (i.e *Completion Time*). O CT pode ser calculado pela seguinte equação: CT = TBA + *Task Cost*, onde, *Task Cost* = (*Task Size*/*Host Speed*)/(1 – *Host Load*).

Quando uma tarefa é alocada em uma máquina, o valor TBA correspondente dele é incrementado pelo *Task Cost*. As tarefas são alocadas de acordo com a quantidade de máquinas disponíveis antes do início da execução. Durante a execução, quando uma tarefa se dá por completa, todas as que não estiverem na fila de execução são escalonadas para uma máquina até que todas sejam completadas. De forma dinâmica, ocorre o processo de re-escalonamento de tarefas. Caso uma tarefa esteja sendo executada em uma máquina e outra máquina com melhor CT é disponibilizada, pode-se então, compensando os tempos gastos com deslocamento, transferir a execução da tarefa para esta.

4.2.4 Dynamic Max-Min2x

A Dynamic Max-Min2x [RSM05] é uma política de escalonamento global e dinâmica, voltada para aplicações do tipo BoT, baseada na atribuição dinâmica de tarefas. Ela recebe a extensão 2x porque toda tarefa tem a possibilidade de ser replicada para executar em duas máquinas simultaneamente.

Essa política reduz o desbalanceamento de carga entre os processadores através da aplicação de maiores prioridades às maiores tarefas. Com isso, as tarefas de uma aplicação são escalonadas conforme a disponibilização dos processadores. Esse mecanismo distribui um maior número de tarefas aos recursos com maior capacidade, os quais finalizam mais rapidamente os processamentos, e, tende a apresentar um melhor desempenho na execução das aplicações.

Outra característica desta política é que utiliza replicação de tarefas como meio para redução do tempo de resposta. Como a *Dynamic* Max-min2x consiste em uma política dinâmica, ela tende a igualar as cargas de trabalho entre os processadores, o que faz com que eles terminem os seus respectivos processamentos em intervalos de tempo similares. Isto gera, como consequência, um pequeno desperdício de ciclos de CPU quando comparado ao número de ciclos utilizados em replicação de políticas estáticas. Basicamente, ela consiste em atribuir as maiores tarefas aos melhores processadores e realizar um escalonamento dinâmico com replicação das tarefas.

Quando comparada com políticas como DFPLTF e WQR, a política *Dynamic* Max-min2x obtém uma média de resultados superiores às das outras políticas [REI05]. Alguns resultados, no entanto, mostram que as vantagens são maiores somente em determinadas condições. Ela tende a ser superior às outras políticas de forma proporcional a granularidade das tarefas submetidas ao sistema, ou seja, quanto maior o tamanho das tarefas, menor o número delas, e por consequência, maior o número de máquinas disponíveis para executá-las. Quando o ambiente possui um nível muito alto de heterogeneidade, ela também possui os melhores resultados, por apresentar desempenho constante.

4.3 Conclusões do Capítulo

Dentre os algoritmos de escalonamento apresentados, o *Adaptive* WQR e Max-min2x se caracterizam por utilizar uma técnica híbrida, o que lhes permite serem utilizados em ambientes com obtenção parcial de informação. O DFPTLF e *XSufferage*,

por serem dependentes de informações, são algoritmos complexos de serem implementados, uma vez que nem sempre as informações podem ser obtidas com precisão devido as próprias características do ambiente.

5 MIDDLEWARE OURGRID

O OurGrid é uma solução de *middleware* para construção de um ambiente de grade computacional par-a-par. Ele é desenvolvido a partir do projeto do MyGrid, ambos da Universidade Federal de Campina Grande – PB [OUR09].

Com o OurGrid pode-se utilizar os recursos compartilhados em torno de uma comunidade ou, entre os sites pertencentes a uma instituição.

A Figura 3, demonstra a estrutura de uma Comunidade do OurGrid (OurGrid *Community*).

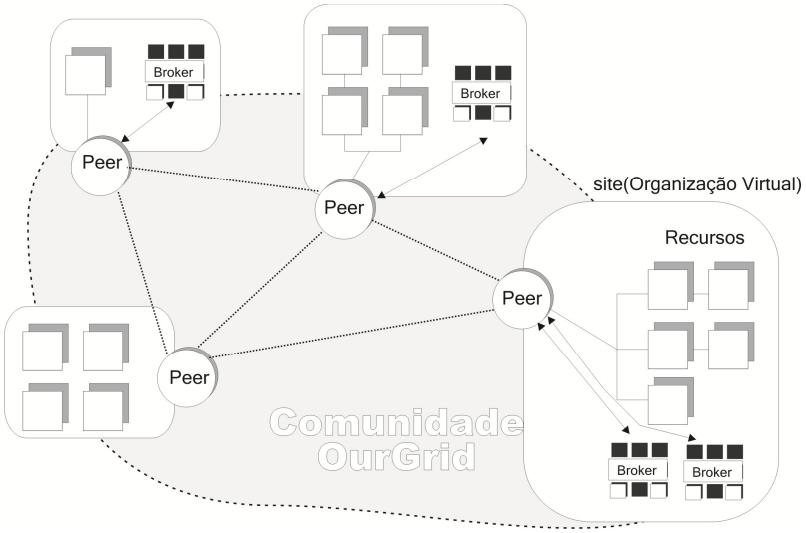

Figura 3 – Visão do compartilhamento de recursos na Comunidade do OurGrid

Cada site ou instituição que esteja participando da comunidade possui pelo menos um *Peer* que está interligado com os demais *Peers* existentes. Desta forma, é possível que um usuário execute sua aplicação utilizando os seus recursos e os recursos que são disponibilizados dentro da comunidade. Cada site pode possuir um ou mais usuários conectados em seu *Peer*, que por sua vez, conectados também a comunidade.

5.1 A Rede de Favores

O OurGrid implementa um mecanismo de segurança para evitar que usuários mal intencionados utilizem os recursos da grade de forma indiscriminada. Este mecanismo é denominado de Rede de Favores [AND03].

Na Rede de Favores, os usuários podem utilizar os recursos da grade, o que é considerado um favor. Contudo, ele fica em débito com os donos dos recursos, necessitando que o mesmo retribua o favor disponibilizando também os seus recursos

quando forem solicitados pelos usuários da grade. Desta forma, ocorre uma contabilização no uso dos recursos.

Este mecanismo faz que, caso os usuários queiram participar efetivamente da grade, também disponibilizem os seus recursos. Os usuários que mais disponibilizam os recursos possuem mais prioridade dentro da grade.

A seguir serão apresentados os componentes pertencentes a este *middleware* e suas funcionalidades dentro de um ambiente de grade.

5.2 Componentes

5.2.1 Broker

O *Broker*, antigo MyGrid, que antecede o projeto do OurGrid, surgiu como uma proposta de tornar mais acessível a utilização de ambientes de grade por parte dos usuários, ou seja, minimizar a complexidade de execução das aplicações. Ele é responsável por gerenciar a execução das aplicações. Através dele os usuários iniciam a execução de suas aplicações (*Jobs*).

Um *Job*, para o OurGrid, é um conjunto de tarefas independentes que podem ser executadas paralelamente. Cada tarefa pode possuir uma ou mais réplicas sendo executadas em um mesmo período de tempo.

Cada réplica possui três fases distintas: *init*, *remote* e *final*. Estas fases são descritas pelo usuário do *Broker* que submete a aplicação para execução. A fase *init* descreve-se os dados que devem ser transferidos para o *Worker* remoto; na fase *remote*, (remote), descreve-se os comandos serem executados pelo *Worker*; na fase *final*, descreve-se os dados de retorno que devem ser transferidos de volta para o *Broker* que enviou a réplica.

Deste modo, o *Broker* permite que, de forma simples, o usuário se preocupe somente com sua aplicação, tornando transparente o acesso aos recursos e as formas como são alocados durante a execução das tarefas. Ele é a fronteira entre o usuário e recursos da grade, provendo as funcionalidades básicas de submissão de sua aplicação, gerenciamento do envio, execução e retorno da mesma.

A alocação dos recursos para execução da aplicação é também uma tarefa do *Broker*. Quando determinado usuário submete sua aplicação para que seja executada, ele faz a requisição ao(s) *Peer*(s) dos *Workers* necessários e aguarda o retorno para que, através de seu componente interno chamado *Scheduler*, inicie a execução da aplicação.

O componente *Scheduler* é responsável por todas as decisões do escalonamento, uma vez que, através dele são implementados os algoritmos de escalonamento.

5.2.1.1 *Scheduler*

O *Scheduler* é um componente interno do *Broker*, e descreve os métodos básicos que são necessários para realizar o escalonamento e, seu funcionamento, depende do algoritmo utilizado.

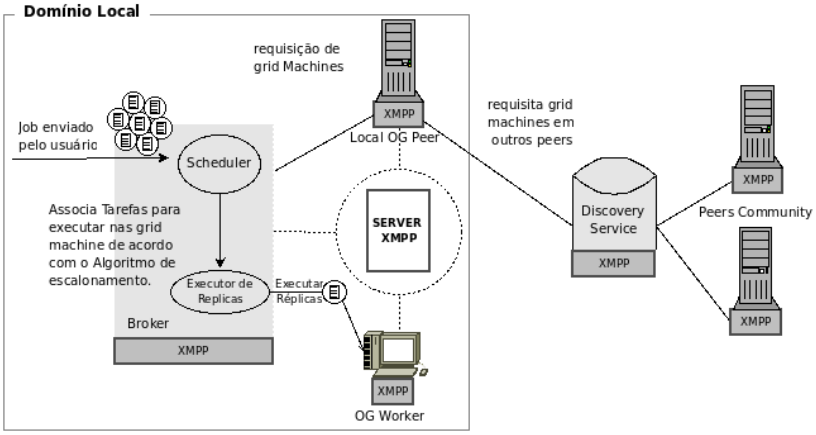

Figura 4 – Processo de escalonamento no OurGrid

O esquema da Figura 4 demonstra o envio de um *Job* (aplicação) pelo usuário e o funcionamento básico do processo de escalonamento do *Scheduler*

Este funcionamento segue as definições do algoritmo de escalonamento WQR do OurGrid. i) Inicialmente o *Broker* repassará para o *Scheduler* o *Job*. ii) Em seguida o *Scheduler* calculará o número de *Workers* necessários e retorna para o *Broker*, que envia o pedido para o *Peer* local. iii) O *Peer* local tentará atender a solicitação de acordo com a especificação requerida (Sistema Operacional, Cpu, Memória, etc), e, caso não possua o número suficiente, buscará junto aos demais *Peers* da Comunidade e, conforme for recebendo os *Workers, o Peer* Local os repassará para o *Broker*. iv) O *Broker* receberá os Workers e repassará para o *Scheduler*, que irá fazer o casamento das tarefas (i.e. das réplicas das tarefas) do *Job* e retornará para executor de réplicas, que por sua vez, iniciará o processo de execução das réplicas. Como o escalonamento é dinâmico, sempre que uma réplica termina sua execução ou lhe é entregue um novo *Worker*, e o *Scheduler* realiza um novo casamento. v) O Réplica Executor transformará todas as tarefas da aplicação em réplicas. Desta forma não existe uma tarefa principal, e sim um determinado número de réplicas que serão executadas concorrentemente nos recursos disponíveis.

Atualmente, o *Scheduler* do *Broker* tem somente implementado o algoritmo de escalonamento WQR. Este algoritmo realiza o escalonamento de forma dinâmica utilizando a replicação de tarefas.

5.2.1.2 WorkQueue Replication

O algoritmo WQR (*WorkQueue With Replication*) em sua formulação original, seleciona as tarefas em uma ordem arbitrária e realiza o escalonamento para os processadores assim que estiverem disponíveis [CIR07][SIL03]. Quando não existem mais tarefas a serem escalonadas, são criadas as réplicas de tarefas em execução para serem escalonadas para os processadores que ainda estão disponíveis ou que venham estar. Quando uma tarefa termina sua execução, todas as suas outras réplicas que foram criadas serão canceladas.

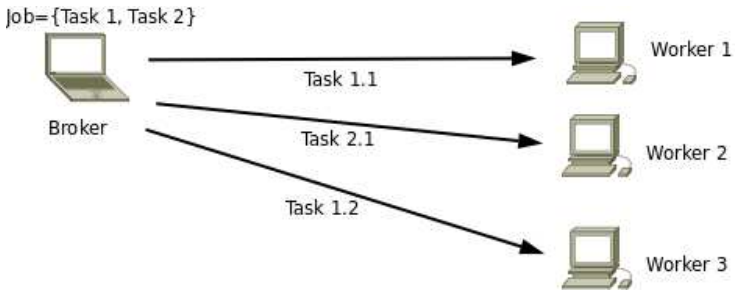

Figura 5 – Replicação de Tarefas com o WQR

A Figura 5 demonstra um pequeno exemplo do WQR para execução de uma aplicação (*Job*) composta por duas tarefas (*Task*s), dentro de um cenário fictício que contém três recursos (*Workers*) disponíveis para processamento das tarefas. Inicialmente, o WQR enviará uma réplica de cada tarefa para os recursos (*Task* 1.1 e 2.1); no entanto, como ainda existe um recurso disponível, será criada uma réplica de uma das *Task* já escalonadas e enviada para este (*Task* 1.2).

O WQR não utiliza nenhuma informação, não tendo conhecimento das características dos recursos, como velocidade do processador, memória disponível, espaço em disco, etc, nem da aplicação, como tamanho dos dados de entrada, custo computacional, tamanho dos dados de saída. O bom desempenho deste algoritmo está relacionado à criação de réplicas.

Durante a execução de uma aplicação dentro de uma grade, é possível que existam recursos que sejam mais eficientes que outros, seja pela velocidade do processador, seja pela carga do processador disponível para executar a aplicação em determinado momento. Esta característica, comum em grades, contribui para que

algoritmos como WQR atribuam tarefas para processadores muito lentos, podendo afetar o desempenho total da aplicação. A replicação de certa forma faz com que mesmo que isso ocorra durante o escalonamento, uma tarefa pode ser escalonada novamente, aumentando sua probabilidade de ser escalonada para um recurso que tenha um melhor desempenho. De fato, isto pode ocorrer, como os resultados apresentados por Silva [SIL03], em que os tempos de execução da aplicação estão próximos de algoritmos que utilizam informação sobre os recursos.

O estudo realizado por Anglano e Canonico [AGL05] adicionou um mecanismo de tolerância a falhas e reinício automático de tarefas ao WQR, resultando no WQR-FT (*WorkQueue with Replication Fault Tolerant*).

No algoritmo WQR original, se uma tarefa falhar, não existe nenhuma ação determinada, consequentemente uma ou mais tarefas podem não completar sua execução. Neste caso, o WQR-FT adiciona o mecanismo de reinicio automático (*automatic restart*) de tarefas (em caso de falhas), para que cada tarefa tenha pelo menos uma réplica em execução até que seja completada. Adicionalmente, o mecanismo de *checkpoint* guarda o estado da computação de cada réplica em execução, para que em caso de falha, o reinicio da tarefa ocorra a partir do último *checkpoint* armazenado.

5.2.2 Peer

O *Peer* é o componente responsável por coordenar o acesso e fazer a contabilização do uso dos *Workers* que estão conectados a ele. Este acesso dentro do OurGrid pode ser realizado de dois modos, primeiro, quando ele permite que um usuário do *Broker* se conecte diretamente a ele, sendo neste caso os recursos considerados locais para o usuário e, segundo, através do acesso indireto, quando o *Peer* está conectado a outros *Peers* através do *Discovery Service* (i.e. está em uma comunidade), e fornece seus *Workers* para um usuário que está conectado em um *Broker* remoto.

A contabilização dos recursos é realizada para garantir os preceitos da Rede de Favores, em que um site o qual disponibiliza seus recursos possa ter uma prioridade maior em caso de concorrência com outro que não disponibiliza ou que na contabilização é inferior ao seu concorrente.

Dentro da estrutura do OurGrid, é possível que um site tenha um ou mais *Peers* e que os mesmos sejam acessados diretamente ou indiretamente pelos usuários da grade.

5.2.3 Worker

Os *Workers* são responsáveis pelo processamento das réplicas encaminhadas pelo *Broker*. Um *Worker*, inicialmente, está cedido para o *Peer* local, porém, pode ser requisitado por um *Peer* remoto quando estiver ocioso.

Cada *Worker* possui um detector de ociosidade e, caso seja habilitado, só estará disponível quando a máquina na qual ele estiver instalado ficar ociosa. Caso contrário, o *Worker* irá compartilhar o processamento da máquina mesmo que ela esteja em uso.

5.2.4 Discovery Service

Um *Discovery Service* (DS) permite que os *Peers* que estejam conectados a ele possam se comunicar, compartilhando os *Workers* entre si. Um exemplo de uso é a comunidade do OurGrid, em que é possível conectar ao seu DS e utilizar os recursos nela disponibilizados.

5.2.5 Commune

O OurGrid em sua versão atual (4.1.4) não utiliza mais a tecnologia Java RMI para comunicação entre objetos remotos. O Sistema de comunicação agora é realizado pelo Commune (antigo JIC [LIM06]), um *middleware* dirigido a eventos que utiliza o protocolo XMPP(*eXtensible Messaging and Presence Protocol*).

Cada componente do OurGrid precisa estar conectado a um servidor XMPP para funcionar, como pode ser visto através da Figura 4. Basicamente, cada domínio necessita de um servidor instalado. Esta tecnologia é considerada *firewall friendly*, ou seja, é necessário somente que se libere a porta do servidor XMPP para que todos os componentes do OurGrid se comuniquem.

5.3 Segurança

O OurGrid é uma solução de grade *free-to-join*, o que em um nível global, permite que um usuário de um site possa fazer parte da grade e utilizar os recursos compartilhados sem a necessidade de enfrentar tramites burocráticos com uma ou mas instituições, contudo, não é necessário realizar a autenticação entre os sites.

Em um nível local a um site, o *Peer* é responsável pelo controle do acesso aos recursos da grade, tanto os locais, quanto os remotos. Como demonstra a Figura 6, existe

uma camada de segurança que é implementada por ele. Primeiramente, para que um *Worker* se conecte a um Peer precisa conhecer a chave pública dele e também, deve estar registrado no *Peer* o seu nome de acesso ao servidor XMPP, para que assim possa oferecer seus recursos à grade.

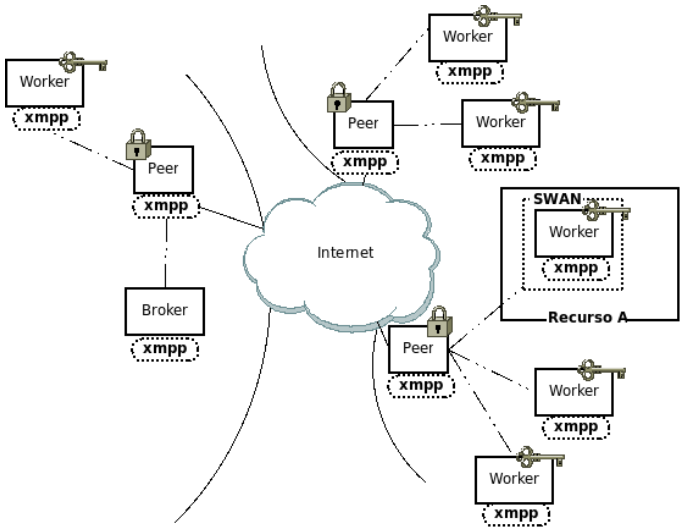

Figura 6 – Aspectos de Segurança do OurGrid

Em um Peer, além de uma lista com os nomes do *Workers* que ele tem acesso, também possui o registro dos *Brokers* que podem conectar a ele. Desta forma, um usuário somente pode enviar uma aplicação à grade se o *Broker* estiver cadastrado em algum Peer.

5.4 Conclusões do Capítulo

O OurGrid permite facilmente que determinada instituição disponibilize seus recursos à grade e que possa usufruir também dos recursos que são compartilhados. Os componentes são de fácil instalação e utilizam a linguagem de programação Java, o que o torna uma solução multiplataforma.

A nova arquitetura do *middleware*, com a adoção do Commune, permite uma maior facilidade de instalação e configuração, uma vez que todos os componentes se comunicam através do protocolo XMPP.

A versão atual do OurGrid (4.0) ainda está em desenvolvimento e, por isso, o *framework* de escalonamento que se esperava para esta versão ainda não está totalmente pronto, tornando o processo de inclusão de escalonadores mais complexo.

O algoritmo de escalonamento disponibilizado se restringiu apenas ao WQR, não sendo mais utilizado o algoritmo *Storage Affinity*, utilizado nas versões anteriores para aplicações que trabalham com grandes quantidades de dados. A arquitetura atual

possivelmente permitirá a adoção de escalonadores não só escritos na linguagem Java, a qual é utilizada em seu desenvolvimento, mas também de escalonadores escritos em diversas outras linguagens.

No próximo capítulo será apresentado o algoritmo de escalonamento que está sendo proposto neste trabalho e também, uma breve descrição da implementação do algoritmo no *Broker* do OurGrid.

6 ALGORITMO DE ESCALONAMENTO TMRORR

A proposta deste algoritmo está baseada no estudo dos algoritmos de escalonamento citados na seção 4.2 e no WQR (seção 5.2.1.2). O objetivo deste algoritmo é atender a um ambiente em que a quantidade de recursos pertencentes ao domínio local é igual ou superior ao que lhe é fornecido pela grade.

Um domínio local tem a vantagem de permitir o acesso aos recursos de forma menos burocrática. Uma aplicação que tenha um requisito de QoS e que necessite que este seja atendido terá uma maior facilidade de executar nos recursos locais, pois estes estão mais acessíveis dentro da hierarquia da grade. Para exemplificar essa situação tomemos o seguinte: uma aplicação que tenha como requisito um recurso com Sistema Operacional Gnu/Linux, com 1024 Mb de memória RAM disponível, e execute um script na linguagem Perl[1] versão 5.1.0, dificilmente terá a possibilidade de solicitar aos administradores de todos os sites da grade que atendam a estes requisitos, pois o trâmite burocrático pode levar um tempo muito grande e, mesmo assim, não satisfazer a todas as necessidades requisitadas.

Sites com estas características podem atender especificamente a um grupo diferente de usuários, com diferentes características em suas aplicações e, mesmo assim, participar de uma grade global em que os recursos externos atendem aos requisitos da aplicação.

Algumas das temáticas que, a partir do estudo do WQR e de outros algoritmos, motivaram a busca de uma nova política que se adequasse às características desse ambiente são: replicação de Tarefas; uso de informação no escalonamento e escalonadores adaptativos (utilizam uma técnica híbrida); e previsão de desempenho.

6.1.1 Motivação

6.1.1.1 Uso de Replicação

O uso de replicação tem demonstrado bons resultados no processo de escalonamento de tarefas em ambientes de grade sem uso de informação, reduzindo muitas vezes o *Makespan* da aplicação, contudo, causa um uso desnecessário de processamento, chegando a *300%* a mais do que a própria aplicação consome [CIR07].

[1] Perl é uma linguagem de Programação multiplataforma. O Significado Perl é “Practical Extraction and Report Language. A versão estável atualmente é a 5.10”[PER09].

No estudo realizado em [FAL07], é demonstrada a sobrecarga causada pelo uso de algoritmos de replicação (especificamente WQR) no escalonamento de tarefas em ambientes nos quais não se utiliza nenhuma informação.

O uso de replicação pode ser justificado pela melhora no desempenho na execução das aplicações em ambientes com pouca ou nenhuma informação disponível. Em ambientes, portanto, em que os recursos são utilizados de forma compartilhada e existe ainda uma contabilização de uso, deve-se ter um controle maior referente ao número de réplicas criadas a fim de conter o desperdício com réplicas que são processadas e não são utilizadas. Certamente, as réplicas estão ocupando um tempo de processamento que poderia ser útil em outra aplicação.

6.1.1.2 Uso de Informação

Os algoritmos que dependem de informação sobre todos os recursos da grade precisam lidar com o desafio de se obter informação de forma confiável. O problema, ou o desafio, é obter frequentemente as informações de todos os recursos da grade, sem causar uma sobrecarga na rede durante a execução das tarefas. Atualmente, grades como o OurGrid possuem domínios interligados através da internet, com um uma conexão considerada de baixa velocidade, o que certamente prejudica a obtenção freqüente de informação sobre os recursos da grade. Porém, internamente em um site, os recursos disponibilizados à grade estão interligados através de redes de média e alta velocidade (100 Mbit/s, 1Gbit/s, etc.).

Em [DER06], há a utilização de infra-estrutura de grade para execução de determinadas aplicações, uma vez que os recursos locais utilizados são rápidos e interligados por redes de alta velocidade, o que proporciona mais segurança e confiabilidade, caso seja necessário obter informações constantes.

As informações sobre os recursos, muitas vezes, estão limitadas a infra-estrutura de obtenção de informação sobre a aplicação, o que pode tornar-se uma tarefa muito mais complexa em um ambiente de grade. Para estimar o custo computacional de uma tarefa é necessário, na maioria das vezes, realizar a execução da mesma ou deixando a cargo do dono da aplicação fornecer as informações necessárias, o que pode, neste caso, não ser muito confiável.

Obter informações somente sobre os recursos e utilizar técnicas de replicação para corrigir possíveis mapeamentos ineficientes, quando não se pode estimar o custo de uma tarefa, pode vir a ser uma solução no escalonamento de tarefas em ambientes com estas características.

6.1.1.3 Previsão de Desempenho

Uma das alternativas para reduzir o tráfego de informações na rede relacionado à obtenção de informação é a utilização de sistemas que, baseados em estatísticas de uso, podem fazer uma previsão de desempenho de determinado recurso por um determinado tempo.

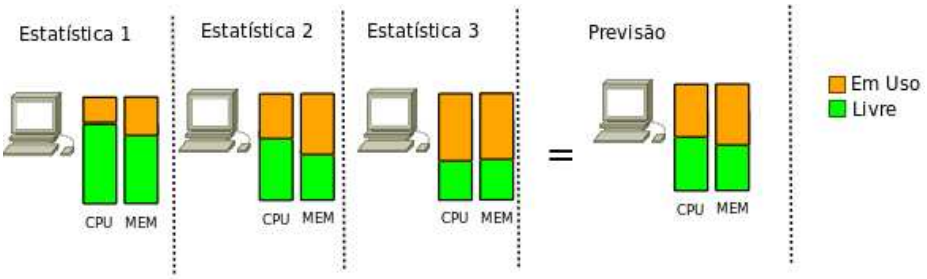

Figura 7 – Previsão de Carga em um recurso utilizando estatísticas de uso.

Considere, a partir da Figura 7, um recurso que é disponibilizado à grade e que seu uso é compartilhado. Durante determinados períodos ele pode estar em uso, todavia, é possível após certo tempo, prever de forma estatística o quanto deste recurso estará disponível dentro de um determinado intervalo de tempo. Esta previsão é repassada para o escalonador decidir se faz ou não o mapeamento de uma aplicação para executar nesse recurso.

Estas previsões podem ser obtidas com o uso de ferramentas que monitoram os recursos da grade, como o NWS (*Network Weather Service*) e Ganglia [WOL99][YAN07].

Existem dois tipos de informações que podem ser obtidas por meio dos recursos da grade, as estáticas e as dinâmicas. Informações estáticas não sofrem mudanças constantes, como a velocidade do processador (i.e freqüência em Ghz), memória instalada. Porém, informações dinâmicas precisam ser colhidas frequentemente, pois sofrem constantes alterações, como carga dos processadores (CPU *Load*), memória em uso, latência da rede, etc. As previsões são geradas com base nas informações dinâmicas.

A Figura 8 demonstra as informações que podem ser obtidas em duas situações diferentes. Quando existe uma ferramenta que realiza o monitoramento dos recursos, é possível obter informações dinâmicas e até mesmo utilizar previsões sobre determinado atributo. Por outro lado, em recursos em que não se tenha meios de prover informações dinâmicas, basicamente as informações que estão disponíveis são informações estáticas.

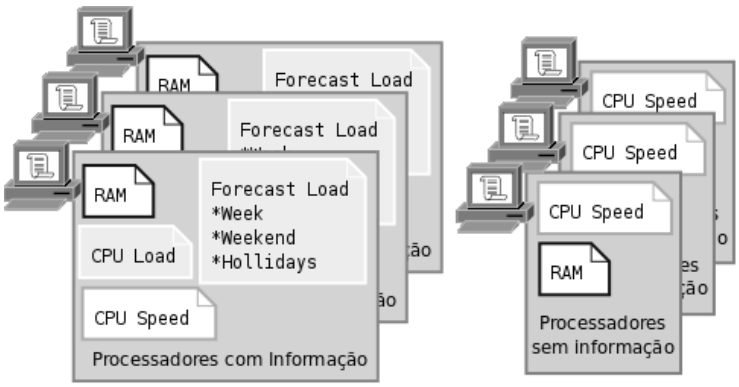

Figura 8 – Possíveis Informações presentes nos Recursos

6.1.1.4 Algoritmo TMRorR

Com base nos objetivos deste trabalho, este algoritmo visa atender a uma política de escalonamento em que se tenha uma decisão diferenciada durante o mapeamento de tarefas relacionadas à propriedade dos recursos. Quando estes pertencem ao domínio local existem limitações no uso da replicação, permitindo somente que ela ocorra em casos de falhas de execução. A justificativa para este procedimento está relacionada às questões de infra-estrutura. Localmente tende a ser mais propício a adoção de ferramentas de monitoração e previsão de informações dinâmicas dos recursos do que em outros domínios da grade. Como dito anteriormente, por questões burocráticas, a instalação destas ferramentas é mais factível.

O TMRorR utiliza as informações somente sobre os recursos, compensando a não utilização de informação sobre a aplicação com o uso de replicação. Inicialmente, as informações utilizadas estão relacionadas diretamente com as características físicas da máquina, como Velocidade do Processador, Memória e Carga do Processador e também, informações estatísticas, como a previsão de carga do processador. O intuito em utilizar informações estatísticas é que o escalonador possa utilizar estas informações. Como exemplo, o escalonador poderá escolher entre os processadores disponíveis aquele que tenha uma maior porção de carga livre em um determinado período de tempo, enviando uma tarefa para ser processada.

Portanto, após receber a quantidade de recursos solicitados para execução da aplicação, os recursos são organizados da seguinte forma: i) os processadores pertencentes ao domínio local são enfileirados a partir daqueles que possuem informação disponível, inicialmente pelos que apresentam as melhores propriedades (Processador mais rápido, maior quantidade de memória, menor previsão de carga), e ii) os

processadores advindos de outros domínios são enfileirados obedecendo aos mesmos critérios, caso tenha informações disponíveis.

Formalmente, cada processador $P_m = \{Cpu_{speed}, Mem, Forecast_{CpuLoad}\}, \forall P_m \in D$, onde D é o domínio. Um domínio pode ser composto por um conjunto de sites, onde $D_i = \{S_1..S_n\}$.

Estas propriedades são comuns a todos os processadores da grade, porém, quando o seu valor não é atribuído por quaisquer que sejam os motivos, seja por não haver informação disponível, seja por não ter tido tempo suficiente para fazê-lo, os escalonadores as tratarão como nulas. Em uma dada comparação entre dois processadores, há aquele que possui as propriedades devidamente informadas e o outro não. Certamente o primeiro terá prioridade na fila.

As propriedades de cada processador são utilizadas para determinar sua prioridade na fila. Utilizando como exemplo a relação das propriedades $(P_{CpuSpeed} \; x \; P_{CpuLoad})$, tem-se a porção do processador que está disponível, podendo enfileirá-los pelos melhores. Porém, como a propriedade $P_{CpuLoad}$ é alterada frequentemente, podem surgir equívocos no enfileiramento. Para isso utiliza-se também a propriedade $P_{ForecastCpuLoad}$, que possui uma previsão de carga do processador.

O valor de $P_{forecastLoad}$ é o valor médio da percentagem de carga do processador prevista para um determinado intervalo de tempo e que é obtida com base nas estatísticas de uso do processador que são colhidas previamente.

Inicialmente, as tarefas são distribuídas para os processadores locais, obedecendo à ordem da fila e, posteriormente, caso existam tarefas para serem escalonadas, estas são mapeadas obedecendo à fila dos recursos pertencentes a outros domínios.

A Figura 9 demonstra o esquema de envio de tarefas pelo algoritmo TMRorR. Neste exemplo existem 5 tarefas e 6 recursos. Primeiramente, as tarefas são enviadas para os recursos locais e, em seguida, as demais tarefas são enviadas para os recursos externos.

A tarefa 5 é replicada para um recurso externo que está disponível. Os recursos locais não recebem réplicas, pois eles tendem sempre a executar uma tarefa que ainda não foi executada localmente. Desta forma, caso a tarefa 2 seja finalizada e o recurso se torne disponível, ele terá a possibilidade de executar somente as tarefas 4 ou 5, que estão sendo executadas em recursos externos. O número máximo de réplicas que podem ser

criadas para cada tarefa, com exceção daquelas criadas para suprir as falhas ocorridas, é de 2.

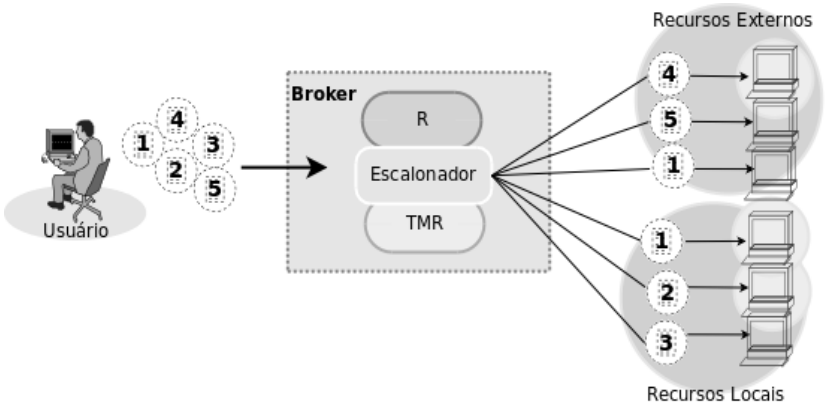

Figura 9 – Exemplo de escalonamento do algoritmo TMRorR

O algoritmo da Figura 10 demonstra, em alto nível, o funcionamento do TMRorR. No início do escalonamento (1) solicita-se os processadores (recursos) disponíveis de acordo com o número de tarefas. Posteriormente, (2) seleciona-se os processadores pertencentes ao domínio local. Depois (3) os processadores pertencentes a outros domínios. A seguir, é realizado o cálculo da confiabilidade (4) para então enfileirar os recursos (5). As tarefas são primeiramente escalonadas para os recursos pertencentes ao domínio local (6) e, caso restem tarefas para escalonar, são mapeadas (7) para os recursos pertencentes ao(s) domínio(s) externo(s). Caso contrário, são replicadas as tarefas que estão sendo executadas nos recursos locais (8), limitado a um número máximo de 2 réplicas por tarefa.

Quando um processador local ficar livre, será encaminhada uma tarefa que não tenha réplicas locais (9), e quando um processador externo ficar livre, será encaminhado a ele uma tarefa que não tenha sido executada ou que não tenha excedido o número máximo de réplicas (10).

1 Obter recursos P_m
2 **Selecione** P_i; /* Recursos do domínio local */
3 **Selecione** P_j ; /* Recursos de domínios externos */
4 **Calcular** C_i ; /* *Confiabilidade* */
5 **Ordenar** P_j e P_i;
6 **Escalonar Tarefas** T = {T1..Tr} para P_i;

7 **Se** *r > 0* , **Escalonar** *T = {T1..Tr}* para *Pj*;
8 Senao Replica Tarefas para *Pj,* limitando *r = 2x*;
9 **Quando** P_i ficar livre, executa T_r que ainda não tem réplicas locais.
10 **Quando** P_j ficar livre, executar T_r que esteja na fila ou que *r < 2x*;

Figura 10 – Funcionamento do Algoritmo TMRorR

O algoritmo TMRorR não é dependente da informação, mas a utiliza quando está disponível para ambos os recursos (i.e. local e externo). Além disso, procura discriminar a quantidade de réplicas criadas a afim de diminuir o desperdício com réplicas.

6.2 Implementação do TMRorR no OurGrid

A versão atual do OurGrid (4.1.4) não possui, ainda, de forma explícita, a disponibilidade de um *framework* de escalonamento, porém o código está sendo refatorado para que isto ocorra, tornando mais simples o processo de implementação de novos algoritmos. Não existe também nenhuma documentação acerca do código da versão atual, uma vez que todo o OurGrid foi refatorado utilizando o Commune para comunicação entre os objetos remotos.

6.2.1 Desafios

Ao idealizar um algoritmo de escalonamento para grades computacionais é necessário levar em consideração as características que são inerentes a estes ambientes, como heterogeneidade, escalabilidade e compartilhamento de recursos. Especificamente na implementação do algoritmo dentro da estrutura do OurGrid, o desafio estava relacionado a obtenção de informações sobre os recursos, pois o *middleware* não disponibiliza uma arquitetura de obtenção de informações.

6.2.2 Implementações realizadas

A implementação[2] foi baseada nas funcionalidades citadas na seção 6.1.1, e realizada utilizando a versão do *branch* "*heuristic_refactoring*" disponível no repositório[3] do OurGrid.

[2] Parte do desenvolvimento do algoritmo ocorreu no LSD – Laboratório de Sistemas Distribuídos – UFCG, com apoio da equipe de desenvolvimento do OurGrid.

O algoritmo de escalonamento é um componente interno do Broker. Todos os métodos que fazem parte das funcionalidades básicas do algoritmo e sua interação com o Broker estão descritas na interface SchedulerIF. Estes métodos básicos envolvem questões como: requisição de *Workers*, envio de uma réplica para execução, cancelamento de uma réplica, recebimento de um *Worker* enviado pelo Peer etc. Todo algoritmo a ser desenvolvido para o OurGrid deve implementar a interface SchedulerIF.

Os métodos adicionados para o desenvolvimento do algoritmo TMRorR atendem as seguintes funcionalidades:

- Organizar os *Workers* recebidos em um grupo local e outro remoto;
- Ordenar cada grupo, iniciando pelos *Workers* que apresentem maior confiabilidade (Velocidade do Processador x Carga do Processador, Previsão de Carga).
- Realizar o casamento das tarefas, inicialmente pelos *Workers* locais e, posteriormente pelos *Workers* remotos. Esses métodos garantem também a prioridade de uma tarefa ser mapeada para um *Worker* local, nos casos que, um *Worker* local e um remoto estejam disponíveis e a tarefa não tenha réplicas locais ou não ultrapasse o limite de 2 réplicas de modo geral.

Para que o algoritmo suportasse trabalhar com informações dinâmicas foram necessárias pequenas modificações em algumas classes comuns aos demais componentes do OurGrid, por exemplo a classe WorkerSpec que contém as informações de um *Worker*.

Com a utilização de um sistema de obtenção de informações, a idéia é que as informações dinâmicas dos *Workers* podem ser atualizadas na classe WorkerSpec, sendo repassadas a seus respectivos *Peers*, que por sua vez as atualizam junto ao *Broker* que estiver alocando os *Workers*, e estes, por fim, entregam-nas ao escalonador (TMRorR).

As implementações foram realizadas atendendo as funcionalidade propostas pelo algoritmo, com intuito de realizar os experimentos utilizando a arquitetura do OurGrid, comparando os resultados do algoritmo com o WQR.

Nos experimentos realizados (seção 7.2), estes atributos foram definidos manualmente com base nas características do ambiente utilizado, uma vez que, durante a realização dos mesmos todos os *Workers* estavam dedicados a execução, não possuindo alteração nas informações dos valores dinâmicos.

---

[3] Disponível em: ourgrid.cvs.sourceforge.net Acessado em: fev 2009.

O algoritmo ficou dependente de um sistema de monitoramento e previsão (e.g. Gmond), o que é esperado para OurGrid, porém, não estava disponível até o momento em que os experimentos foram realizados.

6.3 Conclusões do Capítulo

O TMRorR possui algumas diferenças em relação aos algoritmos Adaptive WQR e Max-min2x, apresentados na seção 4.2. Ele não utiliza informação sobre as tarefas e a replicação possui um controle diferenciado, ou seja, as réplicas são limitadas a 01 (1x) local e 2 (2x) externas. Este controle não permite a ocorrência de replicação de uma tarefa que já esteja sendo executada localmente, mas a de uma tarefa que esteja na fila ou sendo executada em um recurso externo, em situações em que se têm uma quantidade de tarefas superior a quantidade de recursos. Seu funcionamento não é dependente de informação, pois esta é tratada quando está disponível, contudo, considera-se que os recursos locais, em sua maioria, possuam informação disponível.

A implementação do algoritmo TMRorR teve praticamente todas suas funcionalidades atendidas, porém, ficou dependente de uma arquitetura[4] de obtenção de informações dinâmicas dos *Workers*.

A não existência de documentação sobre a versão atual do OurGrid foi um dos maiores entraves no processo de desenvolvimento, o que demandou muito tempo, uma vez que o estudo de seus componentes foi realizado diretamente observando o seu código fonte.

Como o OurGrid passou por uma reestruturação completa, ainda estava em fase de amadurecimento, e consequentemente, era comum o surgimento constante de falhas nos componentes.

[4] Através do branch *heuristic refactoring*, do cvs do ourgrid (ourgrid.cvs.sourceforge.net), é possível verificar a inclusão de uma arquitetura obtenção de informações utilizando o Gmond.

7 AVALIAÇÃO DE DESEMPENHO

Neste trabalho foram realizadas duas formas de avaliação de desempenho, a simulação e experimento. Através da simulação foi possível determinar as características de um ambiente de grade e definir diversos cenários de testes, os quais seriam praticamente impossíveis de serem controlados em um ambiente real. Os experimentos foram realizados utilizando os componentes do próprio OurGrid, em um ambiente real, contudo, dentro de um cenário construído especificamente para os testes.

7.1 Simulações

As simulações foram realizadas com intuito de verificar o desempenho dos algoritmos WQR, TMRorR e DFPTLF na execução de aplicações BoT de acordo com os cenários estabelecidos abaixo, e também, a criação de réplicas, pelos algoritmos que utilizam a replicação de tarefas.

O simulador utilizado neste trabalho foi GridSim *Toolkit* [SUL07]. Ele possibilita a modelagem e simulação de uma grande variedade de recursos heterogêneos, como máquinas monoprocessadas e multiprocessadas, com memória compartilhada e distribuída (e.g. estações de trabalho, SMPs e *clusters*). Também suporta a modelagem de redes de conexão com diferentes características e configurações, usuários, aplicações e escalonadores dentro de um ambiente de grade.

O GridSim permite a criação de aplicações BoT (aplicações com tarefas independentes) e, permite também, o mapeamento de tarefas nos recursos (escalonamento), o que torna possível realizar simulações com algoritmos de escalonamento para este tipo de aplicação.

Para que fosse possível realizar as simulações era necessário descrever o modelo da Grade, que corresponde a estrutura física dos equipamentos e da rede, o modelo da aplicação, que corresponde às características das aplicações (custo computacional, tamanho dos arquivos a serem transferidos pela rede, etc.). Abaixo são descritos os modelos de forma detalhada.

7.1.1 Modelo da Grade

O modelo de grade ao qual este trabalho é dirigido denomina-se de grade computacional, com foco nas aplicações que consomem muito processamento (*CPU-Intensive*).

A grade *G* é composta por um conjunto de sites, onde $G = \{S_1..S_n\}$, e $n \geq 2$. Especificamente, neste trabalho, a referida grade é composta por um site local e outro externo. O site local S_i denotado por $S_i=\{R_1..Rx\}$, e $x > 0$, e o externo por $S_j = \{R_1..R_y\}$, e $y > 0$, em que R representa os recursos pertencentes a cada site.

Cada recurso possui um conjunto de processadores, em que $R = \{P_1..P_m\}$, $m \geq 0$.

7.1.1.1 Topologia da Rede

A Topologia da rede, demonstrada na Figura 11, abrange a interligação entre dois sites, um local outro externo. A interligação entre os sites é realizada através de uma conexão similar a da rede Internet.

O site local possui dois *clusters*, um contendo 12 máquinas, denominado recurso interno "*A*", e outro, com 16 máquinas, denominado recurso interno "*B*". O recurso externo "*C*" possui um *cluster* composto por 06 máquinas.

O *cluster* "*A*" é composto por processadores Pentium III de 1 Ghz, 256 Mb de memória RAM e, rede de comunicação *Fast Ethernet* 100 Mbit/s, e o "*B*", com processadores Pentium IV de 1.6Ghz., 256 MB de memória RAM e rede também de 100Mbit/s.

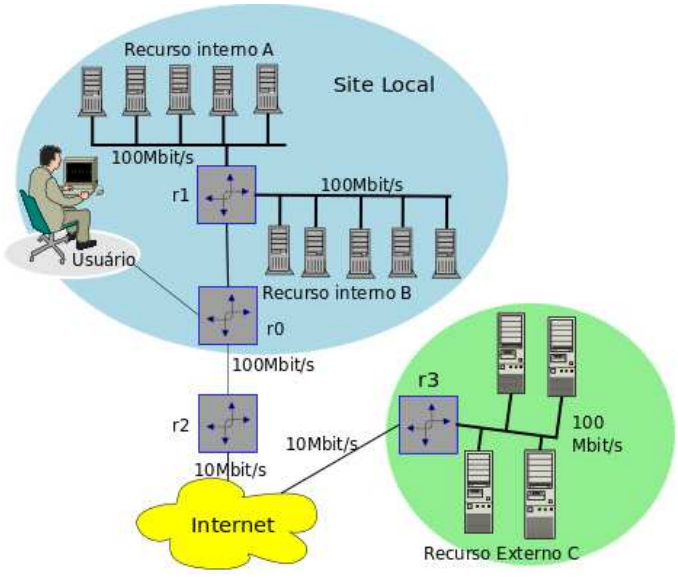

Figura 11 – Topologia da Rede utilizada na Simulação

O recurso externo "*C*" é composto por processadores Pentium IV de 1 Ghz, 256 MB de RAM e rede de conexão de 100 Mbit/s. A rede de conexão interna é de 100Mbit/s.

A conexão entre os sites a velocidade é de 10 Mbit/s. Para todas as conexões foi considerado um retardo de 10 milissegundos (latência).

7.1.2 Modelo da Aplicação

O modelo da aplicação corresponde somente à aplicações BoT, em que, uma aplicação (*Job*) paralela *J* é composta por um conjunto finito de T_n tarefas, onde $J = \{T_1..T_n\}$, sendo $1 < n < \infty$.

Cada tarefa T_n possui um conjunto de atributos - T_n = *{Length, InputSize, OutputSize}* – referentes à sua carga computacional, a quantidade de dados de entrada e a quantidade de dados de saída, respectivamente.

O *Length* é o número de instruções que a tarefa possui, medida em MIPS (*Million Instructions Per Second*) enquanto que quantidade de dados de entrada e saída é medida em *bytes*. Porém, aqui utiliza-se o tamanho em segundos, deixando a cargo do simulador converter este tempo para MIPS.

Baseando-se nos parâmetros utilizados por [NOB06][SIL03][FUJ04][] nos testes de simulação de seus algoritmos e, considerando a topologia da rede definida, juntamente com o modelo da grade, definiu-se os valores dos parâmetros da aplicação de teste.

Foi definido um tamanho único para aplicação, que representa a soma do tempo de suas tarefas, que é de *2812500* segundos. Este tempo foi determinado considerando os valores determinados para cada tarefa para que fosse possível simular diferentes tipos de aplicações (diferentes quantidades e tamanho de tarefas). Uma aplicação pode ter suas tarefas com os valores *2400s*, *4800s*, *38400s* e *307200s*.

Deriva disto a possibilidade de realizar simulações com uma aplicação que tenha uma quantidade pequena de tarefas em contraposição com um alto custo computacional, até simulações nas quais se tem uma grande quantidade de tarefas com baixo custo computacional.

Na Tabela 1, demonstra-se o número médio de tarefas de acordo com o tamanho das tarefas, e também, a relação tarefa/processador ($\frac{T_m}{P_m}$) no caso em que todas as máquinas estejam disponíveis. Cabe lembrar, porém, que esta relação não está determinando necessariamente quantas tarefas estarão sendo executadas em cada máquina.

Tabela 1 – Granuridade da Aplicação e Relação *Tarefa/Processador*

***Tamanho do Grão* (s)**	***Número de Tarefas***	***Tarefa/Processador***
2400	1172	34
4800	586	17
38400	73	2
307200	9	0.2

Ao utilizar os valores de referência de aplicações de exemplos do OurGrid, estabeleceu-se, conforme Tabela 2, o tamanho dos dados de entrada e saída das tarefas. Esta definição torna-se pertinente uma vez que existem recursos (e.g. recurso externo C) que estão interligados por uma rede de conexão com alta latência (i.e. Internet), podendo assim ter impacto significativo no *Makespan* da aplicação.

Tabela 2 – Dados de Entrada e Saída

Aplicação	*Dados de Entrada*	*Dados de Saída*
Pequena	0 – 50 Kb	0 – 20 Kb
Média	50 – 500 Kb	50 – 80 Kb

7.1.3 Análise dos Resultados

7.1.3.1 Testes com Heterogeneidade

A heterogeneidade é uma das principais características de um ambiente de grade, sendo também, um dos grandes desafios enfrentados por algoritmos de escalonamento. Com intuito de tornar sua representação no ambiente de simulação mais próxima de um ambiente real, optou-se por realizar testes que correspondessem a situações em que o conjunto de informações relacionadas às tarefas e aos processadores também fossem heterogêneos.

Para a aplicação determinou-se alguns níveis de variação da carga (custo) computacional das tarefas, e para cada experimento utilizou-se por vez *0%*, *25%*, *50%* e *75%* de variação, tendo como base os trabalhos de [SIL03][NOB08][NET04]. Estes níveis determinam uma distribuição uniforme do custo computacional das tarefas de acordo com a percentagem utilizada. Para exemplificar, considere-se uma aplicação com tarefas de grão pequeno (i.e. 2400s) e se estabeleça o nível 3 (i.e. *50%*) de variação. Resulta-se disso, que uma carga computacional (*Length*) terá uma distribuição uniforme $U_{(1200,3600)}$. Quando a variação é *0%* as tarefas serão todas homogêneas, sendo $U_{(2400)}$.

Em relação aos recursos, a variação está relacionada com a porção do processador que está sendo compartilhada durante toda a execução da aplicação, variando entre 0%, 25% e 50%. A distribuição é feita de forma similar ao que foi realizado com as tarefas.

As simulações relacionadas com o custo computacional das tarefas (i.e. heterogeneidade das tarefas) foram realizadas de acordo com a granularidade das tarefas. Conforme se aumentava o número de tarefas, o tamanho médio das tarefas era decrescido. Desta forma, como apresentado na Tabela 1, quanto menor o número de tarefas maior era o custo computacional de cada uma. Ao elevar-se o nível de heterogeneidade do conjunto de tarefas ocorria que, seguindo um distribuição normal, determinadas tarefas diminuíam o custo computacional na mesma proporção que aumentava de outras.

As simulações foram realizadas obedecendo a um mesmo conjunto de dados e nível de heterogeneidade para cada algoritmo. E também, não foram considerados os custos na obtenção da informação dentro do ambiente, de modo que o algoritmo DFPLTF teve seu funcionamento pleno, servindo de base para comparação com os outros algoritmos.

O gráfico da Figura 12 demonstra os tempos médios de execução obtidos com os algoritmos para um conjunto de 9 tarefas em todos os níveis de heterogeneidade determinados anteriormente. Como era de se esperar, o DFPTLF possui os melhores resultados. O seu desempenho é reflexo do uso de informações sobre os processadores e sobre o custo computacional das tarefas, cujos processadores mais rápidos recebem sempre as tarefas maiores. Realiza, assim, sempre um mapeamento eficiente.

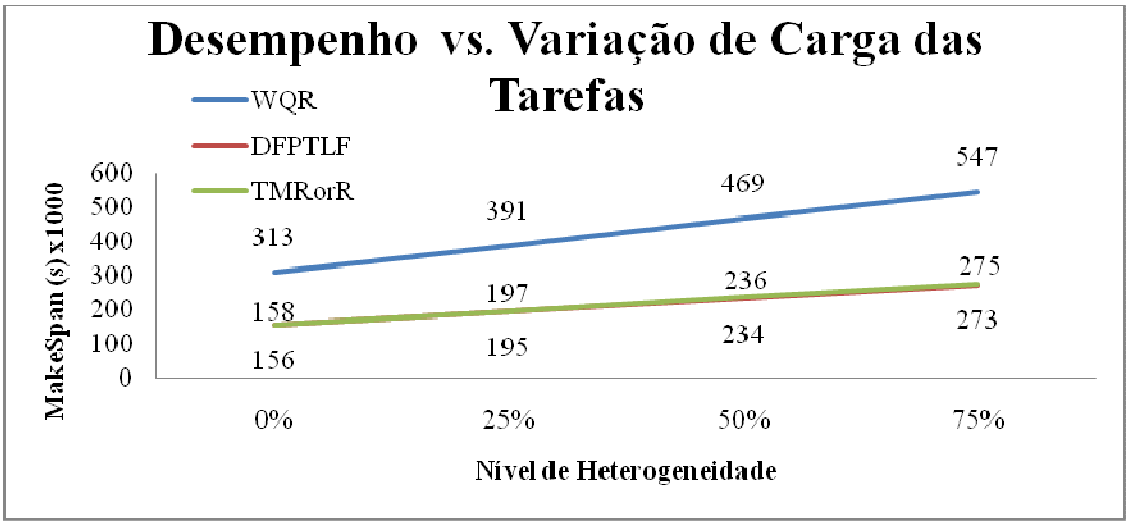

Figura 12 – Gráfico de desempenho dos Algoritmos para 9 Tarefas, em todos os níveis de heterogeneidade.

O TMRorR obteve um desempenho muito similar ao DFPTLF, o que pode ser facilmente compreendido. O Algoritmo possui o conhecimento dos melhores processadores, desta forma, dentro do universo de processadores requeridos foram selecionados os mais rápidos. Como a relação ($\frac{T_n}{P_m} \leq 1$) era baixa, de apenas 0.2, a execução foi realizada entre os processadores mais rápidos, não diferindo neste caso do DFPTLF, que também realizou a seleção dos processadores mais rápidos. O não conhecimento do tamanho da tarefa não interferiu no desempenho do TMRorR.

O número de réplicas criadas pelo WQR-2x chegou a 200% devido ao número de recursos disponíveis, já o algoritmo TMRorR chegou a 66%.

Conforme a relação ($\frac{T_n}{P_m}$) é crescente, os tempos de execução dos algoritmos diminuem, devido ao custo computacional de cada tarefa que também diminui. Quando $\frac{T_n}{P_m}$ > 1, ou seja, aumenta-se o número de tarefas, cada processador inicialmente receberá no mínimo uma tarefa, e os processadores mais rápidos certamente irão compensar os processadores mais lentos, processando um número maior de tarefas.

O gráfico da Figura 13 demonstra o desempenho dos algoritmos com o número máximo de tarefas ($\frac{T_n}{P_m}$ = *34*). O desempenho do algoritmo TMRorR esteve bem próximo do desempenho do WQR-2x. Uma das respostas para a queda de seu desempenho está relacionado ao número de réplicas.

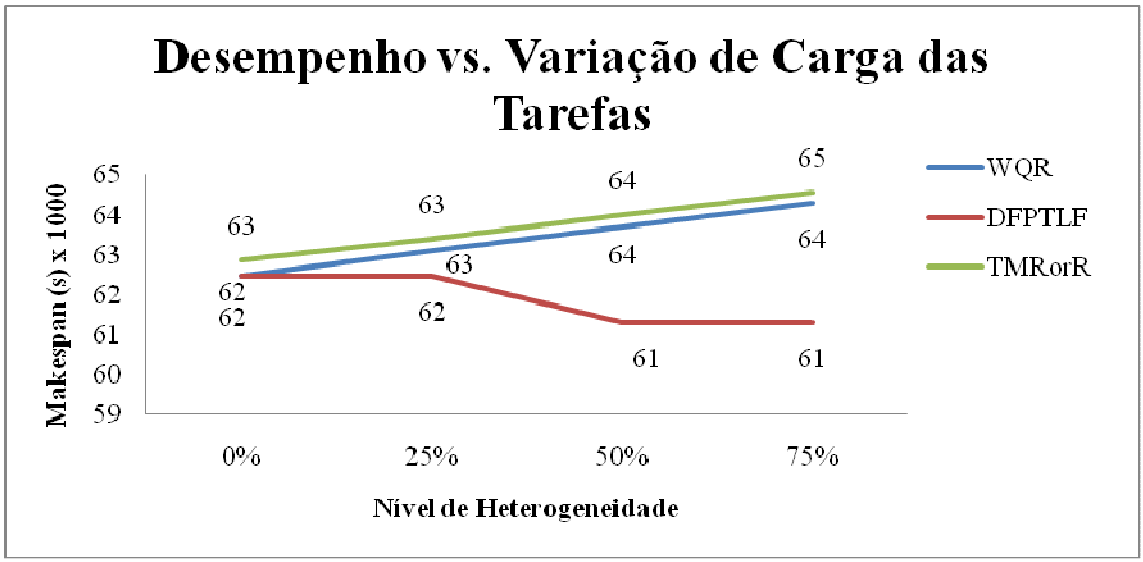

Figura 13 – Gráfico de desempenho dos algoritmos para 1172 tarefas

O TMRorR criou em média 9 réplicas, enquanto o WQR chegou a criar 49. Isso certamente influenciou no tempo de execução da aplicação, uma vez que essas réplicas são criadas nas etapas finais de execução da aplicação, onde $(\frac{T_n}{p_m} < 1)$, possibilita que uma tarefa seja replicada, podendo ser concluída mais rapidamente a depender do recurso ao qual foi escalonada.

A heterogeneidade demonstrou ter uma grande influência nos resultados dos algoritmos que não utilizam informação de todo o ambiente, como WQR-2x e TMRorR. Quando existe informação disponível, tanto para os recursos quanto para as tarefas, o mapeamento foi realizado de forma eficiente pelo algoritmo DFPTLF.

7.1.3.2 Testes com a Variação da Carga dos Processadores

Estes testes foram realizados somente com o WQR e o TMRorR. Nos resultados apresentados na Figura 14, o WQR-2x obteve um bom desempenho nos casos em que a carga dos processadores estava em *0%* e a relação tarefa por máquina era $\frac{T_n}{p_m} < 1$. Porém, com a alteração na variação (25% e 50%) seu desempenho ficou prejudicado.

O aumento da heterogeneidade (i.e variação da carga) favoreceu o desempenho do TMRorR. Conforme aumenta a variação da carga, favorece as escolhas do algoritmo que são sempre baseadas nos processadores mais rápidos, o qual seleciona sempre os melhores entre aqueles que estão disponíveis.

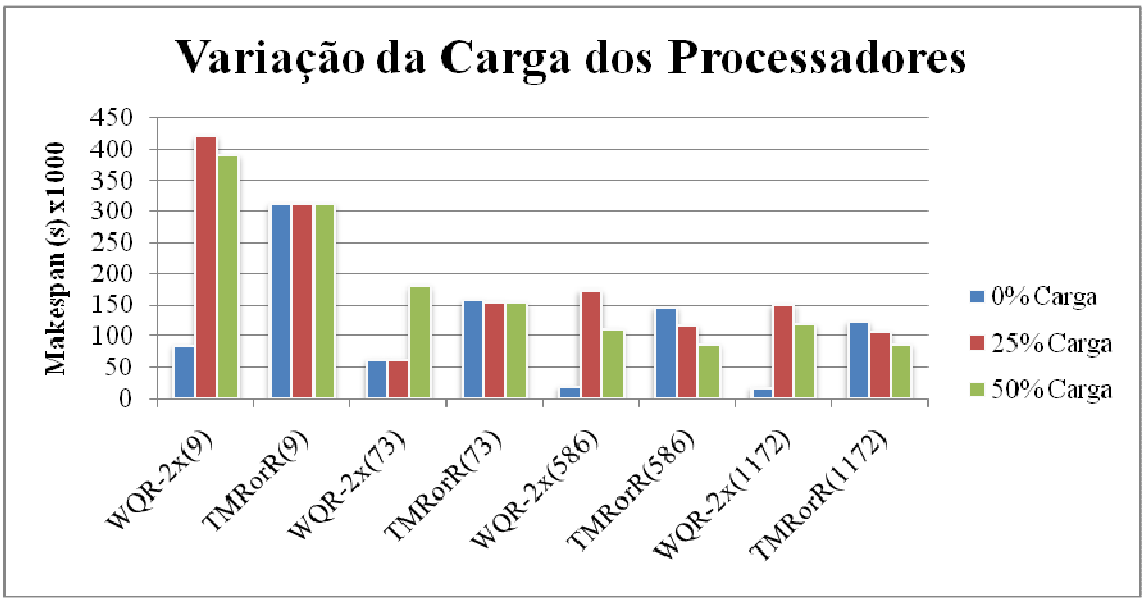

Figura 14 - Tempos de Execução relacionados com a variação de carga dos Processadores

O gráfico da Figura 15 demonstra o número médio de réplicas criadas em cada simulação para cada algoritmo, juntamente com o desvio padrão e a percentagem correspondendo ao número de tarefas simuladas.

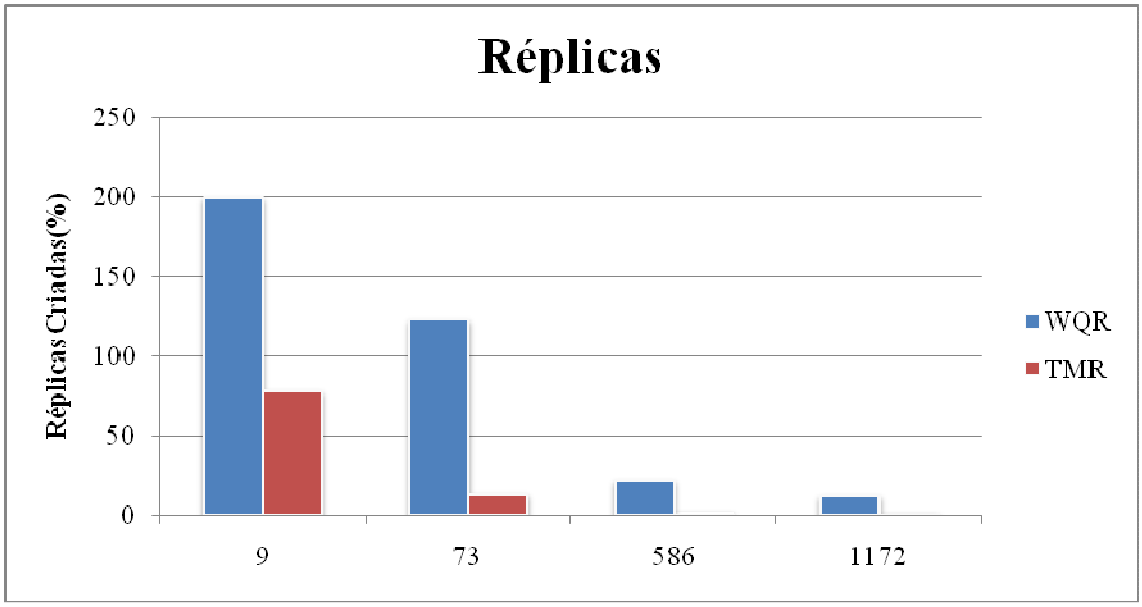

Figura 15 – Réplicas criadas pelos algoritmos durante as simulações

O número de réplicas é crescente conforme diminui a relação $\frac{T_n}{P_m}$. Porém, através das restrições adotadas pelo TMRorR, o número de réplicas criadas não ultrapassa a quantidade inicial de tarefas.

7.1.3.3 Testes com Falhas na Execução das Tarefas

Na realização dos testes com a ocorrência de falhas durante a execução das aplicações considerou-se que o escalonador é responsável por decidir se re-escalona ou não uma réplica de uma tarefa em casos de falha, sabendo que uma tarefa pode possuir mais de uma réplica em execução.

Foram determinados três níveis de falhas, sendo 10%, 25% e 50%, relativos ao número de tarefas de uma aplicação. Para exemplificar, considere um nível de 50% para uma aplicação que possua 586 tarefas, deste modo, poderão ocorrer no máximo 293 falhas.

O TMRorR teve um desempenho melhor que o WQR-2x para as aplicações com 9 e 73 tarefas e praticamente igualando os resultados a partir deste momento. Este bom desempenho deve-se ao número de tarefas o qual é menor ou bem próximo ao número de processadores disponíveis. Isso lhe deu a vantagem de replicar uma tarefa que falhou

para um dos processadores disponíveis que seja mais rápido, contribuindo para um melhor desempenho.

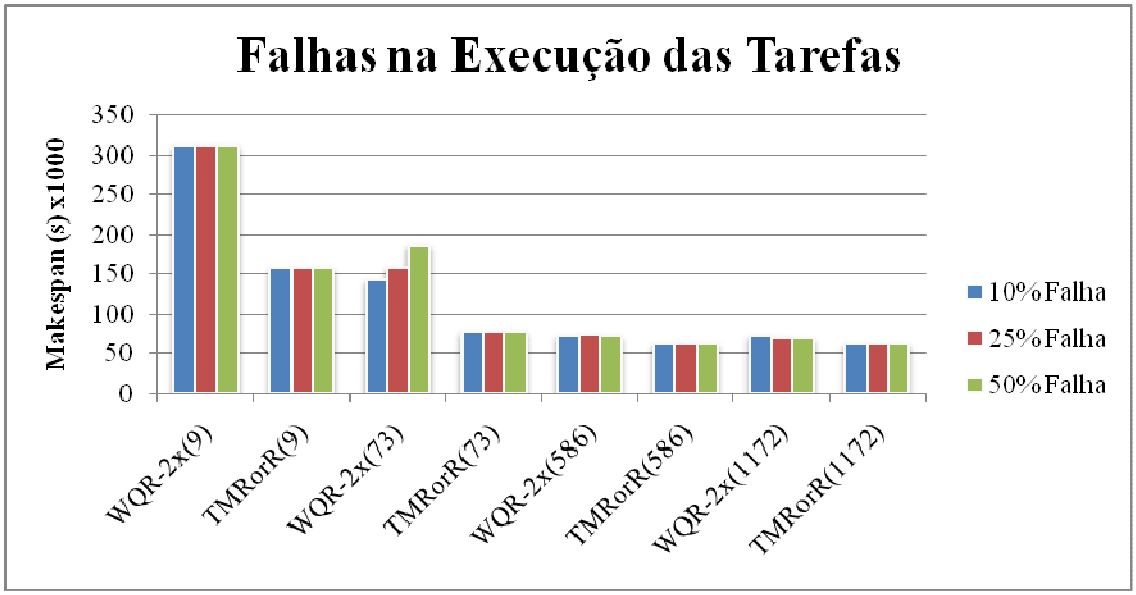

Figura 16 – Tempos de Execução relacionados com as falhas na execução das tarefas

O algoritmo TMRorR cria um número menor de réplicas que o WQR-2x, devido a restrição de que uma tarefa somente é replicada dentro do site local em caso de falha. Os resultados demonstraram que isto diminuiu a quantidade total de réplicas que são criadas, reduzindo o desperdício sem que, de um modo geral, tenha se uma perca considerável de desempenho em relação ao WQR.

A próxima seção demonstra os resultados dos experimentos realizados com os algoritmos TMRorR e WQR, em um ambiente de grade real.

7.2 Experimentos

Os experimentos foram realizados utilizando uma grade constituída pelos componentes do OurGrid, como demonstrado na seção 7.2.1. Os algoritmo TMRorR e WQR-2x estavam implementados em *Brokers* distintos[5] e, cada um deles possuía acesso dedicado e exclusivo à grade durante a execução das aplicações.

O número de recursos pertencentes à grade era fixo e todos estavam dedicados somente a execução das aplicações submetidas pelo *Broker*. A execução das aplicações foram repetidas pelo menos duas vezes a fim de que fosse obtido um valor médio dos resultados.

[5] Os testes com o WQR foram realizados com a versão original do Broker (4.1.4) e o TMRorR com a versão a qual ele foi implementado (4.2.1).

7.2.1 Grade de Teste

A grade de teste é composta por 2 sites, $G = \{S_1, S_2\}$. O site S_1 pertence ao domínio local e o site S_2 pertence ao domínio externo. Cada site é representado pelo componente *Peer* do OurGrid, e são interligados através de um DS (*Discovery Service*).

7.2.1.1 Topologia da Rede

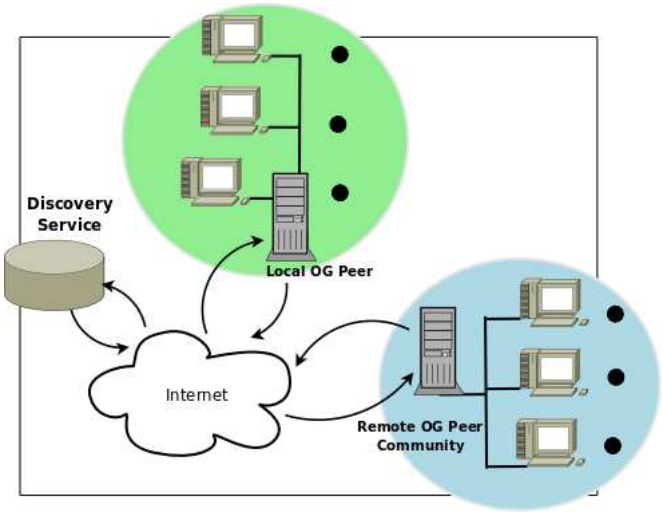

Figura 17 – Topologia da Grade utilizada nos experimentos

A topologia de rede utilizada nos experimentos é demonstrada pela Figura 17. O *Peer* do site local e o *Peer* do site externo estão conectados ao DS através da rede Internet.O *Peer* local é composto por 10 (dez) recursos e o externo por 04 (quatro).

A velocidade dos processadores foi medida em *Bogomips* [DOR96], de modo que, os processadores locais possuíam em média *3198 Bogomips*, enquanto os processadores externos *2574*. Em ambos os casos a rede de conexão entre os recursos era de *100* Mbit/s e a conexão com o DS utilizando um canal dedicado de *512* Kbps.

Todos os experimentos foram realizados durante o período em que toda a infra-estrutura estava dedicada à execução das aplicações, assim como não houve compartilhamento da rede nem os processadores com outros serviços ou usuários.

7.2.2 Aplicação de Teste

A definição das características da aplicação de teste teve como base os exemplos disponibilizados pelo OurGrid de modo que obedecesse a duas condições primordiais: primeira, da possibilidade de ser alterada para adequar à quantidade de tarefas necessárias e, a segunda, de que fosse possível alterar seu custo computacional.

Definiu-se então uma aplicação que se caracterizava por realizar o cálculo de uma multiplicação entre matrizes quadráticas e que armazenasse os resultados em arquivos. O tamanho da matriz foi definido conforme Tabela 3, sendo igual para todas as tarefas da aplicação, sem importar com a quantidade definida.

Tabela 3 – Características da Aplicação utilizada nos Experimentos

Número de Tarefas	*Matriz*
10	500x500
25	500x500
50	500x500

O tempo total de execução de cada tarefa da aplicação incluía além de seu tempo de processamento, a gravação em arquivo das duas matrizes principais e da matriz resultante. A execução de uma única tarefa em um único processador com 3200 *Bogomips* teve o tempo médio de execução de 67 segundos.

7.2.3 Análise dos resultados

Os experimentos realizados objetivaram verificar o desempenho dos algoritmos e a criação de réplicas na execução das aplicações. Em todos os experimentos os processadores estavam dedicados à execução, ou seja, eles estavam com aproximadamente *0%* de carga.

O gráfico da Figura 18 demonstra os resultados obtidos nos experimentos com a aplicação contendo *10*, *25* e *50* tarefas, com *10* processadores pertencentes ao site local e *4* pertencentes ao site externo.

O algoritmo TMRorR obteve um desempenho melhor para a aplicações com 10 tarefas, porém os demais tempos ficaram bem próximos do WQR-2x para 25 e 50 tarefas.

A diferença maior de desempenho entre os algoritmos está relacionada às aplicações com menos tarefas. Isto pode ocorrer devido a forma como o TMRorR faz a seleção dos processadores, encaminhando as tarefas primeiramente para os processadores mais rápidos. Como o número de tarefas era menor que o número de processadores, inicialmente as tarefas sempre foram encaminhadas para os melhores processadores do grupo.

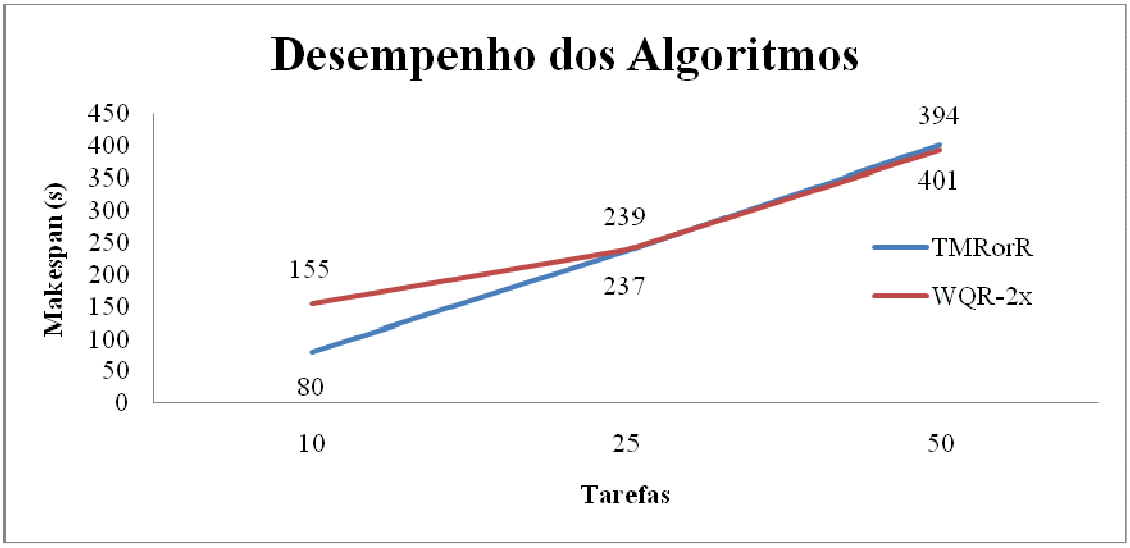

Figura 18 – Gráfico do Desempenho dos Algoritmos na Execução das Tarefas com maior número de processadores pertencentes ao domínio local.

As informações referentes à carga e previsão de carga não influenciaram nos resultados, uma vez que todos os processadores estavam dedicados a execução e as mesmas estavam definidas como 0%.

O gráfico da Figura 19 demonstra os resultados obtidos com a execução das aplicações em que realizou-se a inversão do número de processadores entre o site local e o site externo, com isso, passando para 4 e 10, respectivamente.

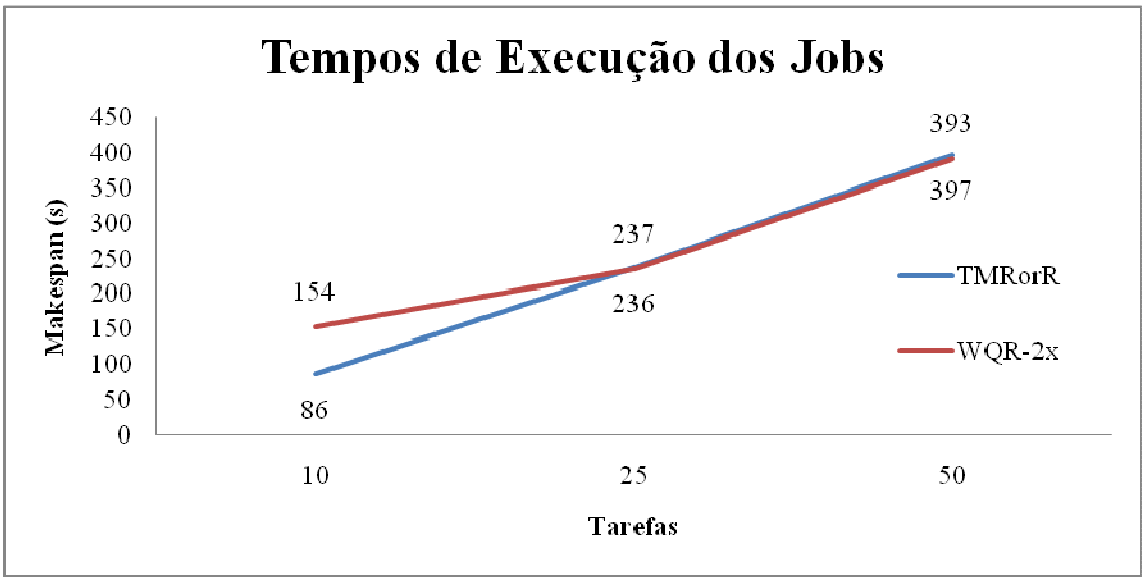

Figura 19 - Gráfico do Desempenho dos Algoritmos na Execução das Tarefas com maior número de processadores pertencentes ao domínio Externo.

Os resultados obtidos são bem próximos dos experimentos anteriores, tendo um pequeno acréscimo em ambos os algoritmos para aplicações com 10 tarefas. Para as

aplicações com 50 tarefas, o TMRorR obteve um pequeno decréscimo no tempo médio de execução, que pode ser justificado pela redução do número de recursos locais, que refletiu na execução de um número muito maior de tarefas nos recursos externos.

Um dos objetivos do trabalho, no desenvolvimento de um algoritmo de escalonamento, era realizar o controle da criação de réplicas. Para tanto, durante a execução das aplicações procurou contabilizar o número de réplicas criadas pelos algoritmos.

O algoritmo TMRorR não permite que mais de uma réplica da mesma tarefa seja executada localmente, contudo, permite que uma tarefa que tenha uma réplica sendo executada em um recurso externo possa executar outra réplica em um recurso local. Esta restrição diminui o número de réplica a serem criadas de um modo geral, como pode ser observado no gráfico da Figura 20.

Durante a realização dos experimentos, tanto o WQR, quanto o TMRorR, realizavam o tratamento de falhas na execução das réplicas. Quando ocorria de uma réplica de uma tarefa falhar executava-se uma nova réplica. Em alguns casos uma tarefa somente completava sua execução quando era escalonada para outro recurso, o que contribui para um aumento no número de réplicas de ambos os algoritmos.

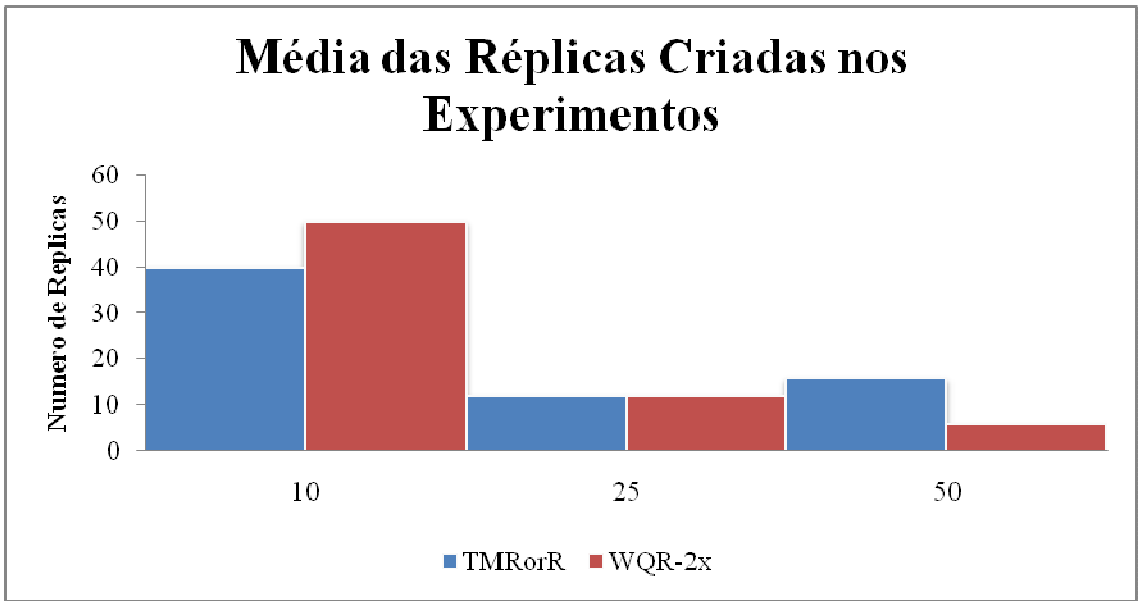

Figura 20 – Proporção média (%) das réplicas criadas nos experimentos

Mesmo com a ocorrência de falhas, a qual não puderam ser controladas, o algoritmo TMRorR possui valor médio da percentagem de criação de réplicas melhor que o WQR em dois casos, para execução da aplicação com 10 e 25 tarefas. Para 50 tarefas a ocorrência de falhas prejudicou uma marcação exata do número de réplicas que foram

criadas. As ocorrência de falhas não estava relacionada com a aplicação, mas sim com o componente de execução (i.e. Worker).

7.3 Conclusões do Capítulo

Os resultados apresentados nas simulações foram mais abrangentes, pois o ambiente permitia controlar todas as variáveis do ambiente, além de prover todas as funcionalidades necessárias para execução plena do algoritmo TMRorR. Os resultados das simulações mostraram que é possível obter alguns resultados melhores que o WQR e, ainda assim, economizar na criação de réplicas dentro da grade.

Os dados de entrada e saída definidos no modelo da aplicação, na seção 7.1.2, tiveram um impacto muito pequeno nos resultados das simulações, não interferindo no tempo total de execução.

Os experimentos por sua vez não foram tão abrangentes. As limitações do ambiente e a complexidade em estar controlando as variáveis de teste não permitiam que fossem atendidos todos os requisitos necessários à execução. Despendia-se de um tempo muito grande para que fosse possível repetir muitas vezes os experimentos, além de que, necessitava que toda a estrutura utilizada estivesse dedicada aos experimentos, uma vez que, não foi possível a utilização de uma arquitetura de monitoramento que pudesse colher as informações dinâmicas que eram consideradas pelo TMRorR.

Mesmo com estas ponderações, os resultados dos experimentos foram satisfatórios, e demonstraram que é possível diminuir o tempo de execução de determinadas aplicações. Salienta-se, contudo, que os resultados parecem indicar que se o controle do ambiente for mais completo e atender mais cenários, os resultados poderão ser melhores.

8 CONCLUSÃO E TRABALHOS FUTUROS

8.1 Conclusões

Um ambiente de grade possui muitas características que, de acordo com o algoritmo de escalonamento utilizado, terá diferentes resultados. A escalabilidade, a heterogeneidade e o compartilhamento de recursos são algumas das características que podem determinar o desempenho de um algoritmo.

As grades que possuem uma infra-estrutura similar ao TeraGrid [TER09], o qual possui uma rede de conexão de alta velocidade interligando os recursos das OV´s, permitem que seja possível a utilização algoritmos que sejam dependentes de informação. Porém, ambientes em que as OV's são interligadas através da Internet e, em alguns casos, por canais da baixa velocidade, as informações a serem obtidas dos recursos podem ser incompletas ou até mesmo inexistentes, o que torna necessário utilizar algoritmos que estejam preparados para ambientes com estas características.

Algoritmos que utilizam a replicação de tarefas, como o WQR do OurGrid, possuem bons resultados quando aplicados ao ultimo ambiente, isto porque não dependem de informação, contudo, qualquer informação que esteja disponível é desprezada.

Este trabalho estudou o funcionamento do WQR e propôs o algoritmo TMRorR para que pudesse, além de outras coisas, utilizar as informações que por ventura estivessem disponíveis na grade.

Nas simulações realizadas com o GridSim, pode-se fazer a comparação dos resultados com os algoritmos TMRorR, DFPTLF e WQR, contudo, nos experimentos, devido a limitações do ambiente de testes e, a complexidade envolvida na implementação do algoritmo no middleware OurGrid, realizou-se somente a comparação com o TMRorR e WQR.

Na maioria dos resultados, o TMRorR obteve um desempenho melhor que o WQR e, em muitos casos, reduzindo também o desperdício de ciclos de CPU com a redução do número de réplicas que foram criadas.

8.2 Trabalhos Futuros

Com o desenvolvimento do trabalho surgiram algumas necessidades que podem ser úteis para complementar o estudo atual ou até mesmo serem aproveitadas em novos estudos. Essas necessidades ficam como trabalhos futuros:

- Utilizar uma arquitetura de monitoramento e previsão de informações para o pleno funcionamento do Algoritmo durante sua execução dentro de uma ambiente real. Atualmente está em desenvolvimento uma arquitetura de monitoramente baseada no Gmond que faz parte do aplicativo Ganglia[YAN07], utilizado para colher informações dinâmicas sobre o estado dos recursos.
- Realizar experimentos mais abrangentes, incluindo novas aplicações e uma quantidade maior de recursos.

REFERÊNCIAS

[ABR00] Abraham, A; Buyya, R; Nath, B. “Nature's Heuristics for Scheduling Jobs on Computational Grids”. In: 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000), 2000, pp. 45-52.

[AGL05] Aglano, C.; Canonico, M. “Fault-Tolerant Scheduling for Bag-of-Task Grid Applications”. In: Advances in Grid Computing, European Grid Conference, 2005, pp. 630-639.

[AND03] Andrade, N.; Cirne, W.; Brasileiro, F.; Roisenberg, P. “OurGrid: An approach to easily assemble grids with equitable resource sharing”. Computer Science, v. 2862/2, 2003, p. 61-86.

[BRA01] Braun, R.; Siegel, H.; Beck, N.; Boloni, L.; Maheswaran, M.; Reuther, A.; Robertson, J.; Theys, M.; Yao, B.; Hensgen, D.; Freund, R. “A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems”. Journal of Parallel and Distributed Computing, vol.61, No. 6, 2001, pp. 810-837.

[BUY02] Buyya, R.; Krauter, K; Maheswaran, M. “A Taxonomy and Survey of Grid Resource Management System for Distributed Computing”. Software – Pratice and Experience, v.32, n.2, 2002, p135-164.

[BUY05] Buyya, R.; Abramson, D.; Venugopal, S. “The Grid Economy”. Proceedings of the IEEE, Vol. 93, No. 3, 2005, pp. 698-714.

[CAL06] Calheiros, R. N. “Um Escalonador Orientado a Sites para Grades Computacionais”, Dissertação de Mestrado, Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2006. 71p.

[CAS00] Casanova, H.; Legrand, A.; Zagorodnov, D; Berman, F. “Heuristics for Scheduling Parameter Sweep Applications in Grid Environments”. In: 9th heterogeneous Computing Workshop (HCW'00), 2000, pp. 349-363.

[CAS88] Casavant, T.; Kuhl, J. “A Taxonomy of Scheduling in General-purpose Distributed Computing Systems”. IEEE Trans. on Software Engineering Vol. 14, No.2, Fev 1988, pp. 141-154.

[CHE02] Chen, H; Maheswaran, M. “Distributed Dynamic Scheduling of Composite Tasks on Grid Computing Systems”. In: 16th International Parallel and Distributed Processing Symposium, 2002, pp. 88--97.

[CIR03] Cirne, W. “Grids Computacionais: Arquitetura, Tecnologias e Aplicações”. 3a Escola Regional de Alto Desempenho, Sociedade Brasileira de Computação, 2003. p. 103–134.

[CIR07] Cirne, W; Brasileiro, F; Paranhos, Daniel; Góes, L. Fabrício, W.; Voorsluys, W. “On the efficacy, efficiency and emergent behavior of task replication in large distributed systems”. Journal of Parallel Computing, vol. 33, No. 3, 2007, pp. 213--234.

[DAN05] Dantas, M. “Computação Distribuída de Alto Desempenho: Redes, Clusters e Grids Computacionais”. Ed. Axcel Books, 2005, pp. 288.

[DER06] De Rose, C. A. F. ; Ferreto, T.; Saikoski, K. ; Cirne, W. “GerpavGrid: using the Grid to maintain the city road system”. In: 18th Symposium on Computer Architecture and High Performance Computing, 2006, p. 73-80.

[DON06] Dong, F.; Akl, S. G. “Scheduling Algorithms for Grid Computing: State of the Art and Open Problems”. Technical Report List for 2006 - Queen's University School of Computing, 2006, pp. 29.

[DOR96] Dorst, W. van; “The Quintessential Linux Benchmark”. Linux Journal. Capturado em: http://www.linuxjournal.com/article/1120. Maio 2009.

[FAL07] Falavinha-Junior, J. N.; Manacero-Junior, A.; Oliveira, L. J.; Boccardo, D. R. “Avaliação de algoritmos de escalonamento em Grids para diferentes configurações de ambiente”. In: WPerformance, 2007, p. 505-524.

[FOS01] Foster, I.; Kesselman, C.; Tuecke, S. “The Anatomy of the Grid: Enabling Scalable Virtual Organizations”. International Journal of High Performance Computing Applications, vol. 15, No. 3, 2001, pp. 200-222.

[FOS02a] Foster, I.; Kesselman, C.; Nick, J. M.; Tuecke, S. “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration”. Open Grid Service Infrastructure WG, Global Grid Forum, Jun 2002. Capturdo em: http://www.globus.org/research/papers.html

[FOS02b] Foster, I. “What is the Grid? A Three Point Checklist”. GridToday, vol. 1, No. 6, pp. 52.

[FUJ04] Fujimoto, N.; Hagihara, K. “A comparison among grid scheduling algorithms for independent coarse-grained tasks”. In: Symposium on Applications and the Internet-Workshops, 2004, pp. 674.

[GTK09] Globus Tolkit. Capturado em: http://www.globus.org, Maio de 2009.

[LIM06] Lima, A.; Cirne, W.; Brasileiro, F.; Fireman, D. “A case for event-driven distributed objects”. In: 8th International Symposium on Distributed Objects and Applications, Out 2006, pp. 1705-1721.

[NET04] Neto, E. L. S. “Escalonamento de Aplicações que Processam Grandes Quantidades de Dados em Grids Computacionais”, Dissertação de Mestrado, Programa de Pós-Graduação em Informática, UFCG, 2004, 82p.

[NOB06] Nóbrega-Junior, N. “Avaliação de Heurísticas de Escalonamento de Aplicações Bag-of-Task em Grids Computacionais Adaptativas à Disponibilidade de Informação”, Dissertação de Mestrado, Programa de Pós-Graduação em Informática, UFCG, 2006. 71p.

[NOB08] Nobrega, N.; Assis, L.; Brasileiro, F. “Scheduling CPU-Intensive Grid Applications Using Partial Information”. In: 37th International Conference Parallel Processing, 2008, pp. 262-269.

[OUR09] Ourgrid. Capturado em: http://ourgrid.org,Março 2009.

[PER09] Perl Programming Language. Capturado em: http://www.perl.org, Abril de 2009.

[REI05] Reis, V. Q. “Escalonamento em grids computacionais: estudo de caso”, Dissertação de Mestrado, Programa de Pós-Graduação em Ciências da Computação e Matemática Computacional, USP, 2005. 112p.

[RSM05] Reis, V.; Santana, M.; Santana, R. H. C.; Mello, R. F. “Uma Política de Escalonamento Orientada à Redução do Tempo de Resposta de Aplicações Paralelas”. In: WSCAD - Workshop em Sistemas Computacionais de Alto Desempenho, vol. 1, 2005, PP. 1-8.

[SET08] Seti@Home. Página oficial do projeto. Capturado em: http://setiathome.berkeley.edu, Outubro de 2008.

[SHI92] Shivaratri, N. G.; Krueger, P.; Singhal, M. “Load Distributing for locally distributed Systems”. Computer, v.25, n.12, 1992, pp. 33-44.

[SIL03] Silva, D. P. “Usando Replicação para Escalonar Aplicações Bag-of-Task em grades Computacionais”. Dissertação de Mestrado, Programa de Pós-Graduação em Informática, UFCG, 2003, 85p.

[SIL05] Silva, J. F. ; Gaspary, L. P. “PeGAC: Uma Arquitetura de Controle de Acesso baseado em Políticas para Grades Computacionais Peer-to-Peer”. In: I Workshop de Peer-to-Peer, vol. 1, 2005, pp. 37-48.

[SUL07] Sulistio, A.; Cibej, U.; Venugopal, S.; Robic, S.; Buyya, R. “A Toolkit for Modelling and Simulating Data Grids: An Extension to GridSim”. Concurrency and Computation: Practice and Experience, vol. 20, 2008, pp. 1532-1609.

[TER09] Teragrid. Site do Projeto. Capturado em: http://www.teragrid.org, Maio de 2009.

[WOL99] Wolski, R.; Spring, N. T.; Hayes, J. “The Network Weather Service: a distributed resource performance forecasting service for metacomputing”. Journal of Future Generation Computing Systems, vol. 15, no. 5, Out 1999, pp. 757-768.

[YAN07] Yang, C; CHEN, T; CHEN, S. “Implementation of Monitoring and Information Service Using Ganglia and NWS for Grid Resource Brokers”.In: 2nd IEEE Asia-Pacific Service Computing Conference, 2007, pp. 356-363.