UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
CURSO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DISSERTAÇÃO DE MESTRADO

USANDO REPLICAÇÃO PARA ESCALONAR APLICAÇÕES BAG-OF-TASKS EM GRIDS COMPUTACIONAIS

DANIEL PARANHOS DA SILVA
(MESTRANDO)

FRANCISCO VILAR BRASILEIRO
WALFREDO CIRNE
(ORIENTADORES)

CAMPINA GRANDE
FEVEREIRO – 2003

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
CURSO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

USANDO REPLICAÇÃO PARA ESCALONAR APLICAÇÕES BAG-OF-TASKS EM GRIDS COMPUTACIONAIS

DANIEL PARANHOS DA SILVA

Dissertação submetida à Coordenação de Pós-Graduação em Informática do Centro de Ciências e Tecnologia da Universidade Federal de Campina Grande, como requisito parcial para a obtenção do grau de Mestre em Informática.

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Redes de Computadores e Sistemas Distribuídos

FRANCISCO VILAR BRASILEIRO
WALFREDO CIRNE
(ORIENTADORES)

CAMPINA GRANDE
FEVEREIRO - 2003

SILVA, Daniel Paranhos da

S586U

Usando Replicação para Escalonar Aplicações Bag-of-Tasks em Grids Computacionais

Dissertação de Mestrado, Universidade Federal de Campina Grande, Centro de Ciências e Tecnologia, Coordenação de Pós-Graduação em Informática, Campina Grande, Paraíba, Março de 2003.

74 p. Il.

Orientadores: Walfredo da Costa Cirne
Francisco Vilar Brasileiro

Palavras Chave:

1. Sistemas Distribuídos
2. Redes de Computadores
3. Grid Computacional
4. Computação Paralela
5. Escalonamento
6. Replicação

CDU - 681.3.066D

Agradecimentos

Em primeiro lugar, gostaria de agradecer aos meus pais Geraldo e Nanci, pela compreensão quando precisei me afastar de casa para cursar o mestrado. Sem o apoio e o incentivo de vocês nada disso seria possível. Também agradeço, a todos os meus outros parentes, que mesmo estando longe, sei que torceram por mim.

Agradeço especialmente a minha namorada, Luana, por todo seu amor, dedicação e pela compreensão que mostrou ao longo dos últimos meses quando precisei me dedicar mais ao meu trabalho.

Queria também agradecer ao amigo Ricardo que dividiu o apartamento comigo durante o curso, e aos amigos que conheci em Campina Grande, Tiago, Renato, Rodrigo, Enio, Emir, Jefferson, Carol, Rafael, Wagner e Pasqueline. Muito obrigado pelo incentivo, confiança e ajuda em todos os momentos. Agradeço também a todos meus amigos de Salvador, porque sei que mesmo à distância torceram por mim.

Sinceros agradecimentos aos meus orientadores Walfredo da Costa Cirne e Francisco Vilar Brasileiro pela seriedade, competência, paciência e dedicação.

Agradecimentos ao pessoal do LSD (Laboratório de Sistemas Distribuídos) Elizeu, Lauro, Nazareno, Milton e Marcelo pela imensa ajuda e a todos os outros pela paciência quando eu precisava executar meus experimentos e deixava suas máquinas lentas.

Agradeço a todos professores e funcionários do Departamento de Sistemas e Computação da UFCG. Em especial a Aninha, Vera e Zeneide.

Concluir esse trabalho sem a colaboração de todos vocês seria uma tarefa muito difícil.

"*Saber que se sabe o que se sabe e que não se sabe o que não se sabe: eis a sabedoria*"

(Chen Lin)

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Escalonar tarefas independentes em ambientes heterogêneos, como Grids Computacionais, não é trivial. Para elaborar um bom plano de escalonamento, em ambientes assim, o escalonador geralmente precisa de algumas informações adicionais como velocidade de máquinas, suas cargas e tamanho das tarefas da aplicação. Esse tipo de informação nem sempre está disponível e muitas vezes é difícil de obter. Nesse trabalho, é proposta uma abordagem de escalonamento que não precisa de nenhum tipo de informação, mas ainda assim atinge uma boa performance. Essa abordagem usa replicação de tarefas para lidar com a natureza dinâmica e heterogênea sem depender de qualquer informação sobre máquinas e tarefas. Os resultados obtidos mostram que o uso de replicação de tarefas proporciona uma performance boa e estável com o custo de consumo adicional de recursos. Entretanto, ao limitar a replicação, o consumo adicional desses recursos pode ser controlado com uma pequena perda de performance.

Abstract

Scheduling independent tasks on heterogeneous environments, like Computational Grids, is not trivial. To make a good scheduling plan on this kind of environments, the scheduler usually needs some information such as host speed, host load, and task size. This kind of information is not always available and is often difficult to obtain. In this work, a scheduling approach that does not use any kind of information but still delivers good performance is proposed. This approach uses task replication to cope with the dynamic and heterogeneous nature of grids without depending on any information about machines or tasks. The results show that task replication can deliver good and stable performance at the expense of additional resource consumption. By limiting replication, however, additional resource consumption can be controlled with little effect on performance.

1 Introdução

Nos últimos anos, a disponibilidade de computadores poderosos e redes de alta velocidade tem aumentado consideravelmente. Este fato tornou possível agregar recursos dispersos geograficamente para a execução em larga escala de aplicações que consomem recursos de forma intensa. Essa agregação tem sido chamada de *Grid Computacional* [19]. Grids podem ser compostos por uma grande variedade de recursos: supercomputadores, *clusters*, SMPs (multiprocessadores simétricos), estações de trabalho, dispositivos de visualização, sistemas de armazenamento e bancos de dados e, instrumentos científicos (como telescópios), todos conectados através de combinações de LANs (*Local Area Network*) e WANs (*Wide Area Network*). Além disso, vários usuários podem usar simultaneamente estes recursos para executar uma variedade de aplicações paralelas. Isto caracteriza os Grids como ambientes heterogêneos e bastante dinâmicos.

A ciência está se tornando cada vez mais colaborativa e não é incomum encontrar colaborações distribuídas por vários países. Apesar da *Web* e o uso de correio eletrônico permitirem que estes grupos trabalhem juntos, seria muito melhor que eles pudessem compartilhar seus dados, computadores, equipamentos científicos e outros recursos de forma que se transformassem em um “laboratório virtual”. A plataforma Grid é uma promessa para prover essa abstração aos usuários.

Essa abstração é obtida pela tecnologia Grid através do provimento de protocolos, serviços e *kits* de desenvolvimento de *software* para permitir que os recursos sejam compartilhados de uma forma controlada e flexível. Esses protocolos e serviços providos são implementados com base em protocolos e serviços da Internet para dar suporte à utilização dos recursos computacionais acessíveis para o usuário através da Internet. Isso possibilita que aplicações paralelas sejam ro-

dadas no Grid sem que haja um limite de distância entre os recursos, já que estes podem estar acessíveis através da Internet.

Pelo fato de a Internet possuir uma alta latência na comunicação, o uso do Grid atualmente se restringe a aplicações que não necessitam de muita comunicação entre os processos durante a execução. Assim, as aplicações compostas por tarefas independentes, que não necessitam de nenhum tipo de comunicação entre si e que não dependem da execução de outras tarefas, são mais aplicáveis aos Grids Computacionais atuais. É claro que é possível se ter um Grid "privado" onde a rede utilizada não seja a Internet. Essa rede pode ser de alta velocidade permitindo que aplicações com alto grau de comunicação entre tarefas sejam executadas. Com o desenvolvimento da tecnologia de redes, isso poderá ser possível utilizando a própria Internet, provavelmente daqui a alguns anos. Entretanto, atualmente, a limitação de conectividade da Internet impõe limites para grande maioria dos esforços Grid.

Um modelo de aplicações caracterizado pela independência de tarefas que pode ser aplicado em Grids é o *Bag-of-Tasks* (BoT) [47]. Neste modelo não é necessário qualquer tipo de comunicação entre as tarefas durante o processamento e as tarefas não dependem da execução de outras. Uma forma de implementação bastante comum deste modelo consiste em possuir um servidor que gerencia uma fila de tarefas a serem distribuídas entre os processadores, assim que estes se tornem disponíveis. Após concluírem as tarefas, os processadores enviam os resultados ao responsável pela integração dos mesmos. O próprio servidor, muitas vezes, também tem a responsabilidade de integrar esses resultados em um único resultado geral. Esta implementação chama-se Workqueue. Ela tem a grande vantagem de ser simples. Infelizmente, ela não obtém uma performance muito boa.

O escalonamento de tarefas no Grid de forma eficiente é um grande problema que vem sendo estudado desde o surgimento do conceito de Grid. Muitos problemas inerentes à plataforma Grid e outros de natureza das próprias aplicações dificultam que esse escalonamento seja eficiente. Alguns desses problemas são: a grande heterogeneidade das máquinas, o compartilhamento de recursos entre vários usuários, a movimentação de dados pelo Grid e consistência dos

mesmos, particionamento da rede, entre outros. Os recursos computacionais precisam ser utilizados, senão da melhor forma possível, de uma forma próxima da melhor.

Existem muitos trabalhos desenvolvidos nessa área relacionados ao escalonamento de tarefas para aplicações rodando em ambientes distribuídos e heterogêneos, tais como Grids [1][9][10][21][22][26][28][32]. Casanova *et al.* propuseram algumas heurísticas de escalonamento para aplicações com tarefas independentes [9][10]. Uma dessas heurísticas, a Sufferage, apresentou bons resultados de performance nos experimentos realizados. No entanto, o problema principal com essas heurísticas propostas é que todas necessitam de informações sobre o ambiente e sobre a aplicação e nem sempre é possível obter essas informações. Um *framework*[1] para o escalonamento de aplicações em ambientes heterogêneos, chamado SmartNet, foi desenvolvido pela equipe de computação heterogênea de um centro da US Navy em San Diego [21][22]. O SmartNet foi desenvolvido para gerenciar aplicações e recursos em ambientes heterogêneos. Esse *framework* é capaz de coletar informações sobre o ambiente e aplicações para alimentar seus algoritmos e prover um bom escalonamento. O problema aqui, novamente, é que há um custo para a obtenção destas informações, especialmente em Grids Computacionais por serem amplamente distribuídos.

Para resolver o problema da dificuldade na obtenção de informações em ambientes heterogêneos e amplamente distribuídos, foi desenvolvido neste trabalho o algoritmo *Workqueue com Replicação* (WQR). O WQR é capaz de atingir uma boa performance mesmo sem usar qualquer tipo de informação sobre os recursos do Grid e tarefas da aplicação. Esse algoritmo é, basicamente, o clássico Workqueue, mas com a capacidade de replicar tarefas para lidar com a natureza dinâmica e heterogênea presente em Grids Computacionais. Quando uma tarefa é replicada, a primeira réplica que termina é considerada como a execução válida da tarefa e as outras réplicas são canceladas. Com essa abordagem, os efeitos da

[1] *Framework* é um conjunto de componentes estáticos e dinâmicos que constituem o esqueleto de uma classe de aplicações. Este esqueleto, incompleto, deve ser estendido pelo usuário com o propósito de gerar aplicações através da personalização de suas instâncias.

dinamicidade das cargas nas máquinas e da heterogeneidade de máquinas e tarefas são minimizados, sem a necessidade de informações sobre o ambiente e aplicação. Uma maneira de se pensar sobre o WQR é que ele permite trocar a necessidade em utilizar essas informações por um consumo extra de ciclos de CPU.

Vale ser ressaltado que o WQR assume que as tarefas são idempotentes e não geram efeitos colaterais. Uma tarefa pode ser executada mais de uma vez sem que haja alteração no resultado final. Como exemplo de um possível efeito colateral, pode ser citado o caso onde duas réplicas incrementam um valor existente em um banco de dados. Entretanto, operações de atribuição de valores fixos podem ser realizadas sem problema. Essa hipótese é adequada a ambientes como Grids pelo fato dos recursos estarem bastante dispersos geograficamente. Assim, não é comum a utilização de aplicações em que tarefas façam acesso (leitura/escrita) a bancos de dados por serem bastante custosas.

A performance do WQR é similar e na maioria das vezes melhor que soluções baseadas no conhecimento total do comportamento do ambiente (praticamente impossível de se obter na prática), com o preço de consumir ciclos adicionais de CPU. Em alguns cenários, o consumo extra de ciclos do WQR é insignificante se comparado à quantidade de ciclos necessários para rodar a aplicação. Em situações em que isto não ocorre (basicamente quando a granularidade da aplicação é muito alta), o consumo extra de ciclos causado pela replicação pode ser controlado limitando essa replicação, uma solução com um impacto pequeno na performance alcançada pelo WQR. Além disso, frequentemente aplicações BoT usam ciclos que se tornariam ociosos [30]. Desse modo, trocar a necessidade do uso de informações sobre o ambiente e aplicações pelo consumo extra de CPU pode ser bastante vantajoso na prática.

O restante dessa dissertação é organizada da seguinte forma. A Seção 2 descreve o que são e como surgiram os Grids Computacionais, destacando sua importância e aplicabilidade em diversos ramos da ciência. A definição de aplicações *Bag-of-Tasks* é apresentada na Seção 3. Nessa seção, pode ser vista a importância dessa classe de aplicações com exemplos reais. Na Seção 4 podem ser vistos os problemas inerentes aos Grids Computacionais que dificultam o escalo-

namento de aplicações bem como algumas soluções para tais problemas. Na Seção 5, é apresentado o algoritmo WQR capaz de escalonar aplicações BoT com eficiência sem necessitar de informações sobre o ambiente e sobre as tarefas da aplicação. Ainda nessa seção, são apresentados resultados de experimentos cujo objetivo é a avaliação da performance do WQR em relação a outros bons algoritmos. Finalmente, na Seção 0, são apresentadas as considerações finais sobre o trabalho, bem como propostas para trabalhos futuros.

2 Grids Computacionais

Pressionada pelo aumento na complexidade dos problemas e impulsionada pelos avanços na tecnologia, a ciência atual é mais baseada na computação, análise de dados e colaboração mútua (como o esforço comum de pessoas que lidam com experimentos e problemas teóricos). Nos últimos anos, o poder computacional tem aumentado constantemente. De fato, por mais de três décadas, a velocidade dos computadores tem dobrado a cada dezoito meses (seguindo a Lei de Moore [13]) e é esperado que essa tendência continue por no mínimo mais uma década.

Entretanto, durante os últimos anos, a largura de banda nas redes de computadores aumentou em uma razão muito maior. Especialistas acreditam que a velocidade das redes dobra a cada nove meses. Ao mesmo tempo em que a performance de processadores e *hardware* de rede aumenta, seus custos de produção estão diminuindo consideravelmente. O outro fato interessante que vem sendo confirmado ao longo do tempo é que o poder computacional também não está evoluindo com a mesma rapidez do armazenamento. Enquanto o poder computacional dobra a cada dezoito meses, o armazenamento praticamente dobra a cada ano. Com a evolução na tecnologia, o custo para se ter um banco de dados com capacidade na ordem de *terabytes* (1024 *gigabytes*) já não é alto. Com menos de R$ 30.000,00 pode se montar um banco desta magnitude. Sabendo dessa tendência na evolução da capacidade de armazenamento, cientistas estão planejando experimentos físicos e simulações que geram arquivos de resultados enormes na ordem de *petabytes* (1024 *terabytes*). Essa grande quantidade de dados demanda mais recursos computacionais para ser analisada e ter esses recursos em um único lugar está se tornando impraticável.

Esses avanços que a tecnologia tem experimentado, principalmente na área de redes de computadores, levaram à idéia de utilizar computadores independentes e geograficamente dispersos conectados em rede como plataforma para execução de aplicações paralelas. Essa plataforma é comumente chamada de Grid Computacional. Sendo uma nova plataforma que permite a execução de aplicações paralelas, os Grids apresentam características diferentes das outras plataformas existentes. Devido a sua heterogeneidade (diversidade de computadores), compartilhamento com outros usuários e complexidade, os Grids tendem a apresentar maiores dificuldades para a execução de aplicações paralelas que plataformas tradicionais. Em geral, Grids são mais propícios para executarem aplicações onde não haja muita comunicação entre tarefas pelo fato de utilizarem a Internet como meio de interconexão.

Apesar de terem mais dificuldades que plataformas tradicionais para a execução de aplicações paralelas, os Grids permitem a interconexão de uma vasta gama de recursos criando uma espécie de ambiente virtual. Esse ambiente virtual é capaz de conectar pessoas, computadores, instrumentos científicos (e.g. telescópios) e bancos de dados proporcionando uma nova forma de colaboração entre comunidades. A Figura 1 exemplifica um Grid Computacional composto por vários recursos diferentes conectados através da Internet.

Um exemplo de como o Grid poderia ser útil é imaginar cientistas espalhados no mundo querendo realizar pesquisas a respeito do Universo. Para tal, poderia ser utilizado um supertelescópio localizado em um determinado ponto do planeta capaz de gerar uma gigantesca quantidade de dados a serem analisados. Sendo um volume muito grande de dados, adquirir um computador capaz de analisar esses dados de forma eficiente pode ser extremamente caro, se existir um computador que atenda às necessidades. Uma solução alternativa seria o particionamento e distribuição dos dados para que pudessem ser processados em paralelo por recursos computacionais pertencentes a um Grid. Depois de obtidos os resultados, estes poderiam ser disponibilizados através do Grid aos cientistas utilizando poderosas ferramentas de visualização. Dessa forma, os resultados poderiam ser avaliados e discutidos pelos cientistas em tempo real.

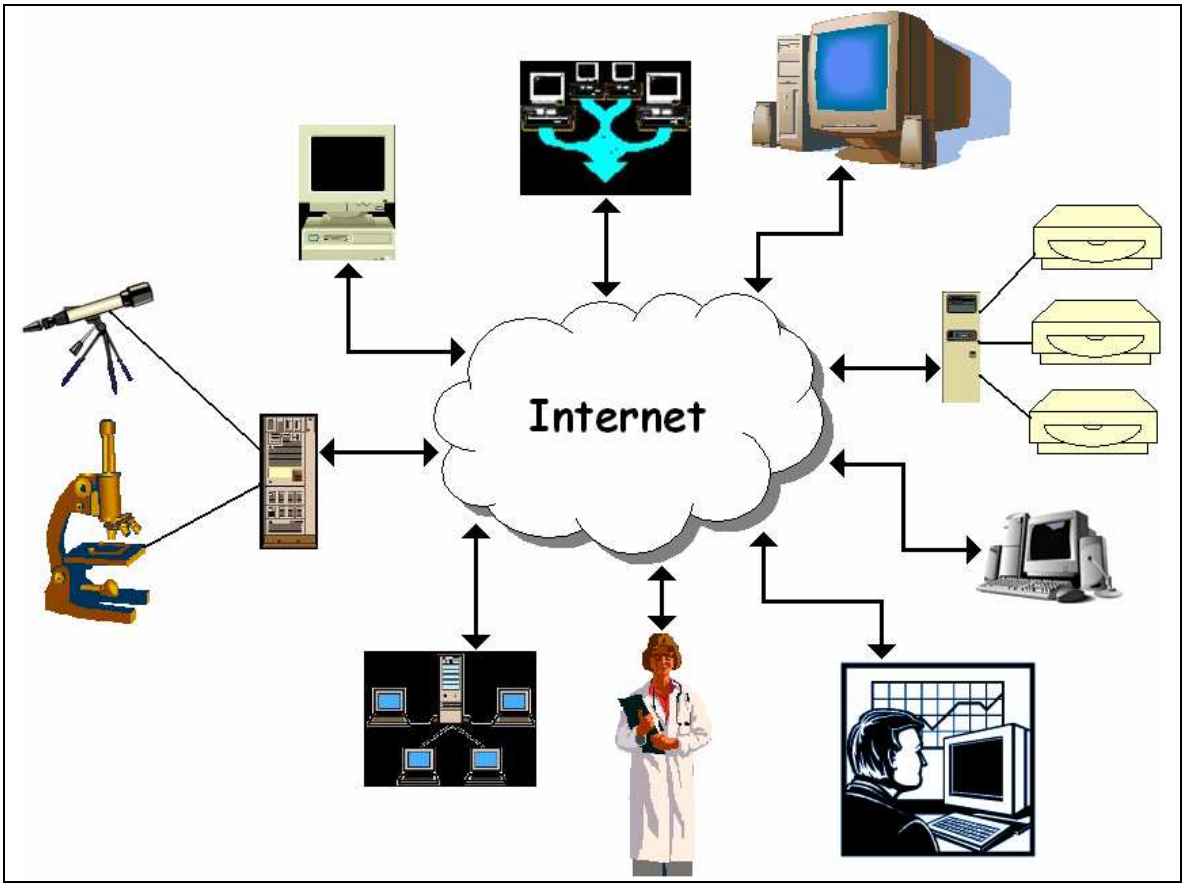

Figura 1. Grid Computacional.

2.1 Surgimento dos Grids

O estado da computação atualmente é bem parecido com o estado da eletricidade no início do século passado. Dispositivos que dependiam de energia elétrica eram desenvolvidos, mas o uso dos mesmos era dificultado porque os usuários tinham que ter seus próprios geradores. O que de fato foi revolucionário na época não foi a eletricidade. O grande avanço foi o desenvolvimento da rede elétrica e das tecnologias de transmissão e distribuição associadas. Este desenvolvimento tornou possível o fornecimento de um serviço confiável, de baixo custo e que fosse de fácil acesso aos usuários. Isso contribuiu para o surgimento de novas indústrias e o consequente desenvolvimento de novos dispositivos que utilizavam energia elétrica.

O termo Grid Computacional vem exatamente da analogia feita à rede elétrica (*Electric Power Grid*) e representa a infra-estrutura que permitirá avanços na

computação, de um modo geral, como ocorreu com a rede elétrica. Essa infra-estrutura é composta por *hardware* que é necessário para fazer a interconexão dos recursos e por *software* específico para monitorar e controlar o ambiente.

Apesar dos computadores estarem mais rápidos que nunca, mais poder computacional é necessário para solucionar grandes problemas. O Grid pode ajudar na solução desses problemas disponibilizando uma grande quantidade de recursos a um custo razoável. O Grid pode conectar recursos geograficamente distribuídos como *clusters*, supercomputadores, instrumentos científicos, repositórios de dados, dispositivos de visualização e pessoas.

O surgimento dos Grids Computacionais tornou-se possível através dos avanços que a tecnologia de rede vem alcançando nos últimos anos. Esses avanços permitiram a interconexão de computadores distribuídos geograficamente pelo planeta. Dessa forma, é possível imaginar a utilização de computadores amplamente distribuídos como plataforma de alta performance para a execução de aplicações paralelas em larga escala. Entretanto, é praticamente impossível usar de forma dedicada uma grande quantidade de recursos dispersos geograficamente para a execução de uma única aplicação. Esses recursos geralmente são compartilhados por outros usuários que também podem estar rodando suas aplicações. Sendo assim, para obter uma boa performance em um Grid, as aplicações devem ser executadas de forma eficiente mesmo rodando em conjuntos de recursos heterogêneos que sejam compartilhados por outros usuários.

2.2 Importância dos Grids

O principal objetivo dos Grids Computacionais é tornar possível a alocação de uma enorme quantidade de recursos para executar aplicações paralelas (e.g. centenas de milhares de computadores conectados através da Internet). Uma outra meta é atingir o objetivo principal com um custo muito menor do que alternativas tradicionais (baseadas em supercomputadores paralelos).

O Grid provê mecanismos de alta performance, seguros e escaláveis para o descobrimento e negociação de acesso remoto a recursos. Isso permite que recursos sejam compartilhados por comunidades científicas em uma escala jamais

vista e que grupos geograficamente distribuídos trabalhem em conjunto de maneiras que antes não eram possíveis.

Os ambientes Grid terão um papel de fundamental importância na ciência, com o potencial de melhorar a qualidade, a eficiência e reduzir o tempo das pesquisas. Esse impacto poderá ser sentido em um grande espectro das atividades experimentais e de simulação. No ramo da indústria, existe uma necessidade para o uso da tecnologia Grid para que as aplicações tenham maior capacidade de resolver problemas complexos, os quais não poderiam ser resolvidos sem a ajuda do Grid.

2.3 Aplicabilidade dos Grids

Portais Científicos

Esses portais [7][11] buscam métodos para resolver problemas complexos de uma maneira mais fácil através da invocação remota de pacotes de *software* sofisticados por navegadores *Web* ou através de clientes leves disponibilizados pelos portais. Esses pacotes podem ser executados remotamente em computadores de um Grid, por exemplo. Tais portais estão sendo desenvolvidos para auxiliar no ramo da biologia, física, computação, dentre outros. A criação de portais Grid não é uma tarefa complicada porque já existem diversas soluções com o propósito de facilitar essa tarefa [24][37].

O *GridPort* [37], por exemplo, é um projeto do *San Diego Supercomputer Center* (SDSC) que reúne uma coleção de serviços, *scripts* e ferramentas permitindo que desenvolvedores conectem interfaces *Web* a um Grid Computacional de forma transparente ao usuário. Os *scripts* e ferramentas provêm interfaces consistentes e segurança entre as infra-estruturas, sendo baseadas em tecnologias Grid bastante difundidas como *Globus* [23] e tecnologias *Web* que se tornaram padrão como CGI [51] e *Perl* [42]. O *GridPort* possui arquitetura aberta e foi desenvolvido de forma a ser flexível e capaz de usar outros serviços e tecnologias Grid. Para permitir que outros desenvolvedores e cientistas criem suas próprias páginas *Web* que acessem serviços de um Grid, o *GridPort* fornece um *framework* para facilitar

essa tarefa. Alguns portais científicos baseados na solução *GridPort* já estão em funcionamento como pode ser visto em [37].

Computação Distribuída

Em 2001, matemáticos dos EUA e da Itália reuniram uma série de computadores para resolver um problema de otimização chamado *Nug30* [38][52]. Este problema é do tipo NP-completo e, portanto o tempo gasto por um algoritmo preciso para resolvê-lo cresce exponencialmente com o tamanho do problema. O *Nug30* é um Problema de Atribuição Quadrática em que 30 unidades devem ser atribuídas para 30 locais. Essa atribuição deve ser realizada de uma forma que seu custo, que depende da distância entre os lugares e o fluxo entre as unidades, seja mínimo. Ao associar uma unidade X a um local A e uma unidade Y a um local B, o custo é calculado através da multiplicação do fluxo entre as unidades pela distância entre os locais. O custo global é calculado através da soma dos custos entre os pares de locais/fluxos atribuídos.

Dadas duas matrizes 30x30, $F = [f_{ij}]$ onde f_{ij} representa o valor do fluxo entre as unidades i e j, e $D = [d_{kl}]$ onde d_{kl} representa a distância entre os lugares k e l, o objetivo é encontrar uma permutação do conjunto S = {1, 2, ..., 30} que minimize o custo global das atribuições. Caso, por exemplo, em um problema de tamanho 4 a solução ótima fosse a permutação S = {4, 2, 1, 3} significaria que a unidade 4 deveria ser atribuída ao local 1, a unidade 2 ao local 2, a unidade 1 ao local 3 e a unidade 3 ao local 4. Dessa forma, a atribuição seria realizada obtendo o menor custo possível.

Para resolver o *Nug30*, um computador isolado poderia levar anos. Sendo assim, surgiu a idéia de agrupar vários computadores para solucionar o problema. Durante uma semana, a colaboração entre EUA e Itália conseguiu ter disponível uma média de 630 computadores e um máximo de 1006 para solucionar o *Nug30*. Com o avanço nas redes de computadores e tecnologias Grid, será possível resolver problemas cada vez maiores e mais complexos através do agrupamento de recursos computacionais.

Análise de Dados em Larga Escala

Muitos problemas científicos interessantes requerem a análise de grandes quantidades de dados. Para tais problemas, a agregação de computação distribuída e recursos para armazenamento são claramente de grande valor [25][36]. Além disso, o paralelismo natural inerente em muitos procedimentos de análise de dados torna possível o uso de recursos distribuídos de forma eficiente. Por exemplo, a análise de *petabytes* de dados gerados por experimentos científicos vai necessitar da união de milhares de processadores e milhares de *terabytes* de espaço em disco para guardar os resultados parciais. Por vários motivos, conseguir reunir esses recursos em um único lugar é praticamente impossível. Em projetos dessa grandeza, várias instituições devem estar envolvidas e elas podem prover os recursos necessários para esse tipo de experimento. Essas comunidades, além de compartilhar computadores e dispositivos de armazenamento, podem compartilhar procedimentos de análise e os próprios resultados.

Estações de trabalho podem ser conectadas através de redes de alta velocidade para formar um ambiente com poder computacional considerável. O programa *FightAIDSAtHome* da *Entropia* [16] reúne mais de 30.000 computadores para analisar o efeito de drogas contra a AIDS. Outro programa com esse mesmo objetivo é o *SETI@Home* [3][44] que será mais detalhado na Seção 3.2. O *SETI@Home* utiliza computadores em residências e escritórios espalhados pelo mundo para analisar sinais de radiotelescópios em busca de detectar sinais extraterrestres. Essa abordagem, apesar de apresentar algumas dificuldades, tem conseguido um enorme poder computacional, nunca obtido antes, e tem propiciado um envolvimento único das pessoas com a ciência.

Instrumentação Contínua

Instrumentos científicos como telescópios, microscópios eletrônicos, dentre outros geram fluxos de dados brutos que são arquivados para uma posterior análise. Uma análise de dados com características próximas a uma análise em tempo real pode aumentar a funcionalidade de um instrumento substancialmente [27][40].

Como exemplo pode ser citado um astrônomo estudando labaredas solares com um radiotelescópio. Os algoritmos utilizados para processar os dados e detectar as labaredas demandam muita computação. Executar esses algoritmos continuamente seria ineficiente para o estudo dessas labaredas porque elas são breves e esporádicas. Se o astrônomo puder solicitar recursos computacionais em larga escala e tê-los disponíveis sob-demanda, ele pode usar técnicas de detecção automática para ampliar as labaredas solares assim que elas ocorrerem. Além disso, o acompanhamento em tempo real do fenômeno pode possibilitar que o pesquisador tome ações dependendo do que está sendo observado (ex. iniciar a aquisição de um novo tipo de dado).

Trabalho Colaborativo

Pesquisadores geralmente não querem apenas agregar dados e poder computacional, mas também conhecimento. A formulação de problemas que contam com a colaboração de várias pessoas e análise de dados, por exemplo, são aplicações Grid de extrema importância [17][50]. Um físico, por exemplo, pode realizar uma simulação que gere *terabytes* de dados e pode querer que colegas espalhados pelo mundo visualizem os resultados da mesma forma e no mesmo momento, assim o grupo pode discutir os resultados em tempo real.

Aplicações apropriadas para Grids frequentemente vão conter alguns aspectos desse e de outros cenários. Por exemplo, um astrônomo pode desejar procurar por eventos similares em um arquivo internacional, discutir os resultados com colegas e requisitar computação distribuída para avaliar hipóteses alternativas.

Como foi visto nesse capítulo, as vantagens da utilização de Grids Computacionais têm atraído muita atenção para a área. O grande atrativo dessa idéia é a possibilidade de alocação de uma enorme quantidade de recursos para executar aplicações paralelas a um custo bastante inferior das soluções tradicionais, como as baseadas em supercomputadores paralelos. Muitos projetos e aplicações já foram postos em prática, consolidando a visão de Grids Computacionais.

3 Aplicações *Bag-of-Tasks* (BoT)

Com o surgimento de problemas complexos a serem resolvidos, sentiu-se a necessidade de particionar a aplicação de modo que ela pudesse ser computada por vários processadores paralelamente. Desse modo, a aplicação pode ser executada mais rapidamente do que se fosse executada em um único processador. Surgiu, então, o conceito de aplicações paralelas onde estas são compostas por diversas tarefas e cada tarefa é executada em um processador distinto. Algumas aplicações paralelas precisam que as tarefas se comuniquem para garantir o progresso consistente e outras não requerem nenhum tipo de comunicação entre tarefas.

A intensidade de comunicação entre as tarefas geralmente determina em que tipos de plataformas[2] a aplicação pode ser executada. Aplicações paralelas com alto grau de comunicação entre tarefas demandam plataformas dedicadas (tais como computadores maciçamente paralelos) de modo que os ganhos com o paralelismo não sejam anulados pelo custo de comunicação. Quando não há necessidade de comunicação entre as tarefas ou essa comunicação é pouca, as aplicações podem ser executadas em plataformas compostas por processadores compartilhados e interconectados por redes de longa distância, como Grids.

Um tipo de aplicação paralela bastante adequada para ser executada em Grids é chamado de *Bag-of-Tasks* (BoT). Essas aplicações são compostas por tarefas independentes. Sendo assim, não é necessário qualquer tipo de comunicação entre as tarefas durante o processamento e as tarefas não dependem da execução uma das outras. Por exemplo, o paradigma de aplicações BoT se aplica

[2] A plataforma de execução de uma aplicação paralela é composta pelos processadores utilizados pela aplicação e também pelas redes que conectam estes processadores.

a situações em que uma mesma função é executada inúmeras vezes com diferentes parâmetros. Uma tarefa é constituída pela aplicação de uma função com uma determinada entrada e o conjunto de todas as tarefas a serem computadas é chamado de *Bag-of-Tasks* (saco de tarefas).

Aplicações BoT são muito úteis a problemas onde é necessária a análise de uma enorme quantidade de dados. Geralmente, essa análise pode ser feita em paralelo através do particionamento dos dados para que possam ser analisados por diversos processadores simultaneamente, sem necessidade de comunicação entre as tarefas. Uma outra aplicabilidade desse tipo de aplicação ocorre na área de simulações. Frequentemente vários cenários precisam ser simulados, seja para identificar o melhor cenário, seja para garantir a validade estatística dos resultados. O processamento em paralelo dessa função com várias entradas diferentes proporciona uma maior velocidade na geração de resultados. Isso permite a execução de simulações complexas que poderiam levar anos se fossem executadas em um único processador. Aplicações BoT estão presentes em diversos ramos da ciência atual e vários exemplos práticos podem ser vistos.

3.1 Aplicações Monte Carlo

Métodos numéricos Monte Carlo [18] podem ser descritos como métodos de simulação estatística, onde a entrada da aplicação inclui componentes aleatórios. Cada entrada aleatória pode ser utilizada para rodar uma simulação independente do fenômeno estudado, caracterizando assim esse tipo de aplicação como *Bag-of-Tasks*. A conclusão sobre o processo estudado pode, então, ser obtida estudando-se a distribuição dos resultados de todas as simulações. Ou seja, simulações são executadas e o resultado do experimento é obtido através da distribuição do resultado observado. Em muitas aplicações, pode-se prever o erro estatístico do resultado final e assim pode ser estimado o número de simulações necessárias para obter uma dada precisão.

Os métodos Monte Carlo agora são usados em diversos campos, desde simulações de fenômenos físicos complexos como a penetração de radiação na

atmosfera da Terra, passando por simulações de processos nucleares até simulações de um jogo de bingo.

As simulações descritas na Seção 5 deste trabalho são um bom exemplo da utilização de métodos Monte Carlo. Essas simulações são realizadas de forma independente utilizando entradas compostas por componentes aleatórios e seus resultados são reunidos para serem analisados posteriormente.

3.2 Programa SETI@Home

A maioria dos programas de busca de inteligência extraterrestre (SETI – *Search for Extraterrestrial Intelligence*) existentes utiliza supercomputadores para analisar a enorme quantidade de dados gerada por radiotelescópios em tempo real. Entretanto, esses supercomputadores têm poder computacional limitado e não conseguem analisar profundamente os dados (sinais mais fracos são ignorados). Para uma análise mais detalhada, é necessário analisar o mais fraco sinal e para isso uma enorme quantidade de poder computacional é necessária.

Adquirir o poder computacional para realizar essa análise profunda pode tornar-se bastante custoso para os programas SETI. Adquirir um computador enorme para executar a análise dos dados seria extremamente caro, isso caso exista tal computador com a capacidade de processamento requerida. Uma solução alternativa para esse problema é a utilização de vários computadores menores através de um Grid Computacional, com um custo muito menor. Essa alternativa torna-se possível devido às características dos dados a serem analisados: de fácil distribuição pelo fato de poderem ser divididos em pequenos pedaços e poderem ser processados de forma independente.

Para conseguir reunir os recursos computacionais necessários, o programa *SETI@Home* [3][44] disponibiliza um protetor de tela que pode ser copiado por qualquer pessoa. Esse protetor de tela é capaz requisitar dados através da Internet, analisar esses dados e enviar de volta os resultados. A idéia de usar um protetor de tela é aproveitar o tempo ocioso das máquinas que, voluntariamente, se juntam ao sistema. Em Julho de 2002, o programa *SETI@Home* contava com

mais 3.8 milhões de processadores espalhados em 226 países, e computava em média a uma velocidade de 27 *Teraflops*[3].

O modelo computacional do *SETI@Home* é simples. Os dados a serem analisados são divididos em unidades de trabalho de tamanho fixo que são distribuídas, através da Internet, para o programa cliente (protetor de tela) rodando em vários computadores. Esse programa cliente computa um resultado (conjunto de sinais candidatos) utilizando técnicas conhecidas como *Data Mining*[4], retorna esse resultado para o servidor e requisita outra unidade de trabalho. Não é necessário qualquer tipo de comunicação entre os clientes, caracterizando o SETI como uma aplicação genuinamente *Bag-of-Tasks*.

Uma outra característica interessante que vale ser ressaltada no *SETI@Home* é a utilização de computação redundante. Cada unidade de trabalho é processada mais de uma vez para que seja possível detectar e descartar resultados de processadores falhos e de usuários mal intencionados. Um nível de redundância de dois ou três é suficiente para esse propósito [3].

3.3 Tomografia

A tomografia computadorizada permite a reconstrução de estruturas 3-D de um objeto baseando-se em projeções 2-D feitas em diversos ângulos do objeto. Essas imagens 2-D armazenadas são alinhadas e então utilizadas para reconstruir as estruturas 3-D usando técnicas de tomografia analíticas e iterativas [41]. Essa reconstrução pode ser muito custosa e necessitar de uma quantidade considerável de poder computacional para ser realizada.

Esse tipo de aplicação é caracterizado como BoT pelo fato de poder ser composto por tarefas independentes. Uma aplicação paralela real de tomografia, com essas características, é a GTOMO, usada pelo NCMIR (*National Center for*

---

[3] *Teraflop* é uma medida de velocidade de computador e equivale a um trilhão de operações de ponto flutuante por segundo.

[4] *Data Mining* pode ser definido como um conjunto de técnicas que envolvem métodos matemáticos, algoritmos e heurísticas para descobrir padrões e regularidades em grandes conjuntos de dados.

Microscopy and Imaging Research). Nessa aplicação, as tarefas recebem arquivos de entrada, realizam o processamento e geram arquivos de saída de forma independente.

Para executar essa aplicação, o NCMIR possuía um certo número de estações de trabalho (usadas como máquinas de uso pessoal e como plataforma de execução paralela) e também tinha acesso a supercomputadores, mas a performance da aplicação ainda podia ser melhorada. Em [46], Smallen *et al.* descrevem uma estratégia de alocação de estações de trabalho e supercomputadores para melhorar a performance de GTOMO. Os experimentos realizados com essa estratégia apresentaram melhores resultados que outras possíveis soluções.

3.4 Condor

Condor é um sistema de gerenciamento de recursos (*Resource Management System* - RMS) e *jobs* especializado para *jobs* que exijam computação intensa. Como outros sistemas, o Condor provê um mecanismo de gerenciamento de *jobs*, políticas de escalonamento, esquema de prioridades, monitoração de recursos e gerenciamento dos mesmos [48][49]. Usuários submetem seus *jobs* para o Condor, e Condor escolhe quando e onde executá-los baseado em uma política, monitora seu progresso, e finalmente informa ao usuário de seu término.

O propósito da criação do Condor é o de fornecer uma grande quantidade de poder computacional a médio e longo prazo (dias ou semanas) utilizando recursos ociosos na rede [30]. Os autores do sistema enfatizam que Condor objetiva obter uma alta vazão (*high throughput*) e não alto desempenho (*high performance*). Desse modo, Condor visa fornecer desempenho sustentável a médio e longo prazos, mesmo que o desempenho instantâneo do sistema possa variar consideravelmente.

Condor foi inicialmente concebido para funcionar em redes de computadores comuns (*Network of Workstations* - NOWs). Uma NOW que executa Condor é chamado de Condor Pool. A parte mais importante da arquitetura de um Condor Pool é o *Matchmaker*. O *Matchmaker* é responsável pela alocação de tarefas a máquinas pertencentes ao Pool, de forma que as necessidades de cada tarefa e

as restrições de uso de cada máquina sejam respeitadas. As necessidades de uma tarefa são especificadas pelo usuário no momento de sua submissão. Por exemplo, uma tarefa pode precisar de uma máquina Sun Sparc, rodando Solaris, com pelo menos 128MB de memória. As restrições de uso de uma dada máquina são especificadas por seu dono no momento da inclusão da máquina no Pool. O dono pode restringir o uso a determinados usuários/grupos e proibir outros. Desse modo, essas restrições permitem ao dono determinar como sua máquina será usada no Condor Pool. Tipicamente, o dono estabelece que sua máquina só deve ser usada quando estiver ociosa e que, ao utilizar a máquina novamente, qualquer aplicação Condor em execução é suspensa imediatamente.

Um aspecto interessante do Condor é que ambos usuários e donos de máquinas são representados no sistema por agentes de *software*. O Agente do Usuário envia as necessidades da tarefa para o *Matchmaker*. Similarmente, o Agente do Dono envia as restrições de uso do recurso ao *Matchmaker*. Ao efetuar o casamento entre tarefa e máquina, o *Matchmaker* notifica ambos agentes. A partir daí, o Agente do Usuário e o Agente do Dono interagem diretamente para realizar a execução da tarefa. A Figura 2 ilustra este protocolo.

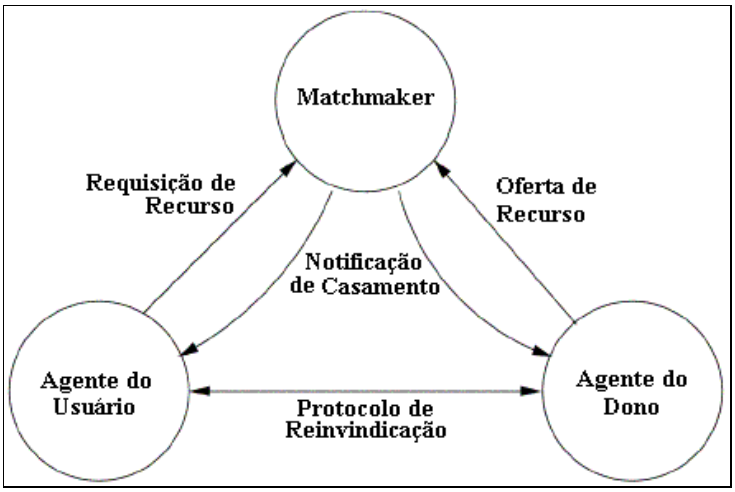

Figura 2. *Matchmaker* do Condor.

4 Escalonando Aplicações em Grids

Um dos grandes desafios existentes em sistemas computacionais paralelos e distribuídos é o desenvolvimento de técnicas eficientes para a alocação de tarefas de aplicações paralelas para os processadores. A alocação dessas tarefas é conhecida como escalonamento de aplicações paralelas. Esse escalonamento visa atingir um conjunto de objetivos relacionados com medidas de desempenho, por exemplo, minimizar o tempo médio de resposta e/ou maximizar a utilização dos recursos. Esses objetivos são muitas vezes conflitantes e o escalonador deve tomar decisões influenciadas por diversos fatores (tais como a velocidade e carga das máquinas, bem como o tamanho das tarefas).

De acordo com o tempo em que as decisões do escalonador são tomadas, os escalonadores podem ser classificados como estáticos ou dinâmicos. O escalonamento é dito estático quando os processadores em que as tarefas serão executadas são escolhidos no início da execução. Um bom representante da classe de escalonadores estáticos é o *Fastest Processor to Largest Task First* (FPLTF) [33]. Pelo fato dos escalonadores estáticos realizarem a alocação das tarefas para os processadores no início da execução da aplicação, estes são comumente usados com recursos dedicados. A dinamicidade e heterogeneidade de recursos presentes nos Grids Computacionais fazem com que a utilização de escalonadores estáticos em Grids não seja uma boa solução. Por outro lado, escalonadores dinâmicos (adaptativos) fazem suas decisões durante a execução da aplicação [1]. Diferentemente dos algoritmos estáticos, os dinâmicos podem atingir uma boa performance em Grids por causa da habilidade que eles possuem em se adaptar às mudanças no ambiente ocorridas durante a execução da aplicação.

4.1 Problemas no Escalonamento

Existem muitos problemas inerentes à plataforma Grid e outros de natureza das próprias aplicações que precisam ser resolvidos para que as tarefas dessas aplicações sejam escalonadas de forma a executarem eficientemente. A seguir serão apresentados alguns dos principais problemas detectados para se escalonar aplicações em Grids Computacionais.

Heterogeneidade dos Recursos

Uma das grandes dificuldades de se escalonar tarefas para serem executadas em um Grid se dá quando os recursos são bastante heterogêneos. Essa heterogeneidade é própria das diferentes arquiteturas de computadores. Algumas das variáveis que provocam essas diferenças são: velocidade de processadores, organização e interconexão dos mesmos, velocidade e tamanho de disco, tamanho e velocidade de memória, arquitetura interna de dados, sistema operacional, dentre outras [21]. Por exemplo, a taxa de execução de algumas tarefas pode ser diretamente proporcional apenas à velocidade do processador, enquanto outras podem depender da latência da rede. Isso faz com que determinados recursos funcionem bem para certas aplicações, mas não tão bem para outras.

Por outro lado, esse problema da grande heterogeneidade das máquinas no Grid pode ser visto como um ponto positivo. Isso porque as aplicações podem possuir diferentes tipos de tarefas com diferentes requisitos de execução. Como as máquinas são bastante heterogêneas, o escalonador pode atribuir as tarefas para as máquinas que possuam maior “afinidade” com as mesmas.

Compartilhamento de Recursos

Um dos maiores desafios em Grids Computacionais é obter performance para a aplicação. Isso acontece por causa das variações que ocorrem dinamicamente nas cargas sobre os recursos e na disponibilidade dos mesmos, já que estes recursos podem estar sendo compartilhados com outros usuários.

A utilização de máquinas por vários usuários para rodar suas aplicações pode ocasionar uma sobrecarga nas máquinas, prejudicando a performance des-

sas aplicações em geral. O escalonador tem como função balancear a carga a ser rodada em cada máquina para tentar evitar esse problema ao máximo. Essa tarefa pode ser facilitada através da obtenção de estimativas de performance dessas máquinas antes da atribuição das tarefas. Uma forma de obter essas estimativas será descrita posteriormente na Seção 5. Para tal, é utilizado um serviço de monitoração chamado NWS [53] (*Network Weather System*) que grava e prevê a performance de recursos computacionais e de rede de um sistema ao longo do tempo.

Além de balancear a carga nas máquinas, o escalonador também deve fazer o mesmo com a utilização das redes. Se uma determinada rota entre duas máquinas estiver sendo muito utilizada, pode ser benéfico escalonar outras tarefas em máquinas onde uma rota diferente possa ser utilizada. Isso pode evitar que as aplicações entrem em colapso com o congestionamento da rede.

Movimentação de Dados

Quando estão sendo executadas aplicações que trabalham com uma grande quantidade de dados, geralmente esses dados necessitam que sejam movimentados de uma determinada localização para outra. Essa movimentação tem como objetivo obter um ganho de performance no processamento. Nesse transporte de dados, deve ser levado em conta o tempo gasto com o tráfego dos dados pela rede e o tempo de processamento a ser gasto na máquina de destino. A soma desses dois tempos deve ser inferior ao tempo gasto na máquina inicial se os dados não fossem transportados [26].

Como a quantidade de dados pode ser enorme e o tamanho dos fragmentos (partições dos dados) também pode ser grande, o escalonador tem que ser preciso na sua decisão para que esse transporte de dados não acabe comprometendo a performance da aplicação. Para que isso não ocorra o escalonador tem que ter uma boa noção da topologia da rede e da localização dos dados.

Para o escalonador, o conhecimento de onde os dados estão localizados pode ajudar na alocação das tarefas para os processadores. Essa necessidade de saber a localização dos dados é justificada pela “proximidade” da máquina que

executará a tarefa em relação aos dados que vão ser utilizados. Vale ressaltar que essa “proximidade” se refere ao tempo gasto na recuperação dos dados por parte da máquina. Quanto menor o tempo gasto na recuperação desses dados, “mais próximos” eles estão.

Particionamento da Rede e Consistência dos Dados

A replicação dos dados possibilita uma maior disponibilidade, mas traz um problema, qual seja: a necessidade de manutenção de consistência das cópias dos dados. Este problema se torna crítico quando a rede é fragmentada em partições. Quando existem cópias de dados em partições diferentes e esses dados são modificados, surge o problema de tornar o estado dos dados consistente novamente quando o sistema se recuperar da falha [13]. Existe, ainda, a possibilidade de que processadores lentos demorem a responder as requisições feitas pelo escalonador. Isso faz com que a rede esteja “virtualmente particionada” mesmo não estando fisicamente, mas o efeito é o mesmo de um particionamento físico.

Gargalo na Integração de Resultados

Após a execução das tarefas pelas máquinas que compõem o Grid, há a necessidade de que seja feita a integração dos resultados obtidos ao final da execução de todas ou de um subconjunto das tarefas. Ao terminarem de executar as tarefas, as máquinas devem enviar seus resultados para o responsável pela integração e aguardar pela possível atribuição de novas tarefas. Quando o responsável pela integração estiver de posse do resultado da execução de todas as tarefas, ele faz uma “união” dos resultados gerando a saída final da computação.

O problema nessa integração de resultados é que o responsável por ela pode se tornar um gargalo do sistema. Isso pode ocorrer quando estiver chegando uma grande quantidade de dados provocando um congestionamento na rede ou quando o processamento necessário for grande causando uma sobrecarga no processamento da máquina que roda o consolidador.

4.2 Escalonando Aplicações Bag-of-Tasks em Grids

Uma forma de implementação bastante comum deste modelo consiste em possuir um servidor que gerencia uma fila de tarefas a serem distribuídas entre os processadores, assim que estes se tornem disponíveis. Após concluírem as tarefas, os processadores enviam os resultados ao responsável pela integração dos mesmos. O próprio servidor, muitas vezes, também tem a responsabilidade de integrar esses resultados em um único resultado geral.

A arquitetura básica deste modelo pode ser visto na Figura 3. Este modelo, porém, está suscetível a gargalos no servidor (gerenciador de tarefas), e poderá ser eficiente apenas se o trabalho realizado pelo servidor corresponder a uma pequena fração da quantidade total de trabalho e quando o número de processadores não for tão grande. A principal vantagem dessa técnica é que ela não precisa de informações sobre o ambiente e sobre a aplicação para realizar o escalonamento das tarefas. Por outro lado, sua performance não é muito boa comparada a outras técnicas.

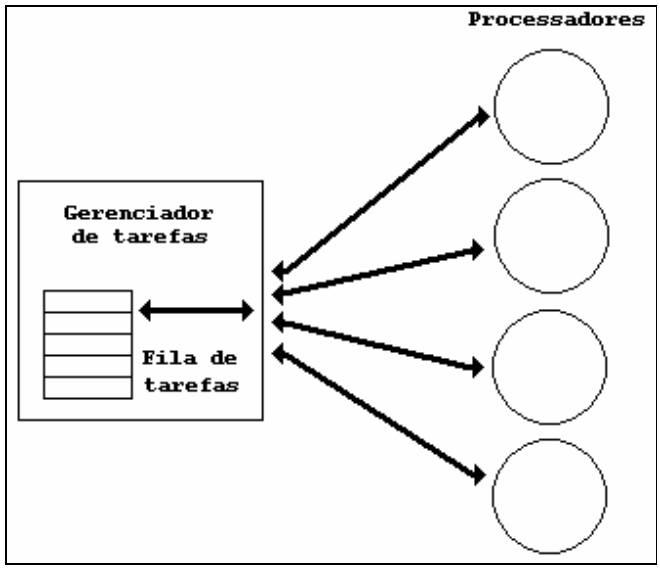

Figura 3. Arquitetura básica de uma implementação do modelo *Bag-of-Tasks*.

Escalonar aplicações em um ambiente heterogêneo e dinâmico como um Grid Computacional não é uma tarefa trivial. Algumas características que são intrínsecas de Grids devem ser consideradas durante o escalonamento, como a heterogeneidade dos recursos, a dinamicidade de cargas nas máquinas e na rede, a

largura de banda, latência e topologia da rede (principalmente quando a aplicação utiliza grandes quantidades de dados). Naturalmente, a dificuldade de escalonamento depende das características da aplicação alvo. Até mesmo o escalonamento de aplicações BoT não é uma tarefa trivial como poderia se pensar devido a sua simplicidade. Aplicações BoT são simples pelo fato de serem compostas por tarefas independentes que podem ser executadas em qualquer ordem e não necessitam de comunicação entre tarefas. Entretanto, escalonar tarefas independentes em Grids, embora seja mais simples do que a grande maioria das aplicações paralelas, ainda assim é difícil devido ao comportamento dinâmico e a heterogeneidade intrínseca dos recursos exibida na maioria dos Grids.

Ambientes Grid são tipicamente compostos por recursos compartilhados. Sendo assim, a concorrência criada por outras aplicações, rodando simultaneamente nesses recursos, causa atrasos e degrada a qualidade do serviço. Além disso, os recursos de um Grid são, de um modo geral, heterogêneos e não se comportam da mesma maneira para todas as aplicações. Atingir boa performance nesse tipo de situação geralmente requer o uso de boas informações (confiáveis e precisas) do ambiente e da aplicação para definir o plano de escalonamento.

Infelizmente, devido à ampla distribuição geográfica dos recursos em um Grid, é normalmente difícil obter informações confiáveis sobre todo o Grid. Essa ampla distribuição geográfica do Grid dificulta a instrumentação para obter informações de carga (máquinas e rede), as quais variam dinamicamente. Existem alguns esforços para instrumentar um Grid e obter tais informações (carga das máquinas, latência e largura de banda da rede) [20][31][53]. Alguns resultados iniciais são encorajadores [31][53], mas fazer tal monitoração em grande escala, como em um Grid, envolve sérios obstáculos como saber onde os monitores devem ser instalados e com que frequência devem executar as monitorações [20]. Além disso, às vezes não é possível instalar sistemas de monitoramento em certas máquinas do Grid. Alguns *sites* impõem restrições administrativas no que pode ser rodado e o tráfego de rede é comumente filtrado por *firewalls*. Finalmente, apenas o ato de monitorar não é suficiente. A fim de garantir um bom escalonamen-

to, previsão de performance futura baseada em valores obtidos através de monitoração é geralmente desejável, o que complica ainda mais o problema.

Além disso, é também difícil de se obter informações sobre as tarefas (tempo de execução em um determinado processador) para elaborar o plano de escalonamento. Às vezes, a única maneira de se obter tais informações é rodar uma tarefa em um dado processador. Naturalmente, essa estratégia funciona apenas em casos em que todas as tarefas têm a mesma complexidade ou que sejam muito parecidas. Uma tarefa pode ser executada em todos os processadores e seu tempo de execução em cada um deles pode ser usado para prever o tempo de execução das tarefas restantes.

Apesar das dificuldades em obter informações confiáveis sobre os recursos do Grid e sobre as tarefas, a maioria dos esforços no estudo do escalonamento de aplicações compostas por tarefas independentes em ambientes heterogêneos e dinâmicos assumem que essas informações confiáveis estão disponíveis [4][5][6][9][10][21][22][32]. Por exemplo, heurísticas de escalonamento dinâmico foram estudadas por Maheswaran *et al.* [32], que concluíram que a escolha das heurísticas de mapeamento dependem de parâmetros como a heterogeneidade das tarefas e máquinas e a taxa de chegada das tarefas.

São poucos os estudos referentes ao escalonamento de tarefa em ambientes heterogêneos sem a utilização de informações sobre o ambiente (velocidade, carga das máquinas) e a aplicação (tamanho das tarefas). James *et al.*, por exemplo, comparam alguns algoritmos de escalonamento nestes cenários [29]. Os resultados mostraram que quando informação está disponível, um algoritmo continuamente adaptativo é melhor para aplicações com tarefas que tenham complexidades com um alto grau de simetria em torno de uma média. Quando os tempos de execução não são fáceis de serem previstos, James *et al.* descobriram que é melhor usar um algoritmo do tipo *First-Come First-Serve*.

Neste trabalho é apresentado um algoritmo dinâmico chamado *Workqueue com Replicação* (WQR) para o escalonamento de aplicações BoT em ambientes Grid. Essa solução não necessita de nenhum tipo de informação sobre o ambiente e a aplicação. O WQR será detalhado mais adiante na Seção 5. Para comparar a

performance desta solução, foram considerados apenas algoritmos dinâmicos pelo fato de serem mais adequados a Grids. Os algoritmos escolhidos para a comparação foram: *Dynamic Fastest Processor to Largest Task First* (Dynamic FPLTF), *Sufferage* e *Workqueue*. Esses algoritmos foram escolhidos porque são bastante conhecidos e estudados. O FPLTF é um algoritmo estático que apresenta boa performance em ambientes dedicados [33]. O Dynamic FPLTF é o resultado de uma modificação feita para torná-lo adaptativo e consequentemente utilizável em ambientes Grid. O Sufferage tem mostrado atingir boa performance em ambientes heterogêneos e dinâmicos [1]. O Workqueue foi obviamente escolhido pelo fato do WQR ser uma extensão dele.

Dynamic FPLTF

O Dynamic FPLTF tem uma boa habilidade de se adaptar à dinamicidade e heterogeneidade do ambiente. O Dynamic FPLTF precisa de três tipos de informações para escalonar as tarefas devidamente: `Task Size`, `Host Load` e `Host Speed`. `Host Speed` representa a velocidade relativa da máquina. Uma máquina que tem `Host Speed = 2` executa uma tarefa duas vezes mais rapidamente que uma máquina com `Host Speed = 1`. `Host Load` representa a fração de CPU da máquina que *não* está disponível para a aplicação (fração de CPU que está sendo usada por outros usuários e aplicações). Vale ser ressaltado que `Host Load` varia com o tempo, dependendo da carga que está sendo imposta à máquina por outros usuários e aplicações. Finalmente, `Task Size` é o tempo necessário para uma máquina com `Host Speed = 1` completar a tarefa quando `Host Load = 0`.

No início do algoritmo, o tempo para se tornar disponível `TBA` (*Time to Become Available*) de cada *host* é iniciado com `0` e as tarefas ordenadas por tamanho em ordem decrescente. Deste modo, a maior tarefa é a primeira a ser alocada. Cada tarefa é alocada para o *host* que provê o menor tempo de execução `CT` (*Completion Time*) para ela, onde:

`CT = TBA + Task Cost`

$$\text{onde Task Cost} = \frac{\text{Task Size} / \text{Host Speed}}{1 - \text{Host Load}}$$

Quando uma tarefa é alocada para uma máquina, o valor do TBA correspondente a este *host* é incrementado com `Task Cost`. As tarefas são alocadas até que todas as máquinas dos Grid estejam em uso. Após isso, a execução da aplicação é iniciada. Quando uma tarefa é completada, todas as tarefas que não estão rodando no momento são desescalonadas e re-escalonadas até que cada máquina do Grid tenha pelo menos uma tarefa alocada. Esse esquema continua até que todas as tarefas sejam completadas.

Essa estratégia tenta minimizar os efeitos da dinamicidade do Grid. Uma máquina que, em um primeiro momento, é muito rápida e pouco carregada recebe mais tarefas que uma máquina também muito rápida, mas com bastante carga. O problema, neste caso, aconteceria quando uma máquina pouco carregada passasse a ter muita carga. Essa variação na carga poderia comprometer a execução de toda a aplicação já que as tarefas escalonadas para essa máquina iriam executar muito mais lentamente que o previsto. O processo de re-escalonamento de tarefas tenta corrigir esse problema alocando as tarefas prioritariamente para as máquinas mais rápidas do momento.

Esse esquema consegue atingir uma boa performance, como poderemos ver na seção 5.2, mas ele é complicado para ser implementado na prática. O Dynamic FPLTF precisa de muita informação sobre o ambiente (tamanho das tarefas, carga e velocidade das máquinas). Geralmente, esse tipo de informação é difícil de se obter com precisão e frequentemente não está disponível devido a restrições administrativas, tornando a solução inviável na maioria dos casos.

Sufferage

A idéia por trás do Sufferage [1][10] é que uma máquina é designada para executar uma tarefa que “sofreria” mais se aquela máquina não fosse atribuída para ela. O valor *sufferage* de uma tarefa é definido pela diferença entre seu segundo melhor e seu melhor *Completion Time* CT (calculado da mesma forma mos-

trada acima). Como pode ser visto na Figura 4, o algoritmo Sufferage tem basicamente um laço enquanto com um laço para dentro.

```
(01) enquanto ainda existe tarefas no “saco de tarefas” {
(02)    todas as máquinas são marcadas como “não-usadas”
(03)    para cada tarefa no “saco” {
(04)       encontrar as máquinas que proporcionam o melhor e o segundo
             melhor Completion Time
(05)       valor_sufferage = segundo melhor CT - melhor CT
(06)       se a máquina que proporciona o maior valor_sufferage
             para a tarefa está marcada como “não-usada” {
(07)          aloca a tarefa para a máquina
(08)          remove a tarefa do “saco”
(09)          marca a máquina como “usada”
(10)       }
(11)       senão {
(12)          se o valor sufferage da nova tarefa é maior que o
              valor sufferage da tarefa previamente designada para a
              máquina{
(13)              desaloca a tarefa previamente designada
(14)              coloca esta tarefa novamente no “saco”
(15)              aloca a nova tarefa para a máquina
(16)              remove a nova tarefa do “saco” aloca
(17)          }
(18)       }
(19)    }
(20) }
```

Figura 4. Algoritmo Sufferage.

O laço enquanto é necessário para garantir que todas as tarefas sejam escalonadas. No início desse laço, todos as máquinas são marcadas como “não-usadas”. Após isso, o laço para é iniciado. Neste laço, o valor *sufferage* de cada tarefa do “saco de tarefas” (conjunto de tarefas a serem executadas) é calculado e uma tarefa é possivelmente designada para a máquina que lhe proporciona o melhor *Completion Time*. Se a máquina está marcada como “não-usada”, a tarefa é designada para ela, removida do “saco” e a máquina se torna “usada”. Se a má-

quina está marcada como “usada”, os valores *sufferage* da tarefa previamente designada para aquela máquina e da nova tarefa a ser alocada são comparados. A tarefa que fica naquela máquina é a que possui o maior valor *sufferage*, a outra retorna para o “saco de tarefas” e não é considerada novamente neste laço `para`. Depois do término do laço `para`, a próxima iteração do laço `enquanto` inicia e o processo é repetido até que todas as tarefas sejam escalonadas.

Naturalmente, os valores *sufferage* de cada tarefa variam durante a execução da aplicação por causa da dinamicidade das cargas intrínseca em ambientes como Grids. Para lidar com isso, nos experimentos realizados neste trabalho o algoritmo acima é invocado várias vezes durante a execução da aplicação. Toda vez que uma tarefa é completada, todas as tarefas que ainda não começaram a rodar são desescalonadas e o algoritmo é invocado novamente, usando os valores *sufferage* atuais. Consequentemente, o algoritmo roda novamente escalonando as tarefas restantes, mas desta vez com a nova carga das máquinas. Esse esquema é repetido até que todas as tarefas sejam completadas.

O Sufferage, de certa forma, privilegia as tarefas maiores escalonando-as antes das menores. Poderia se pensar que o comportamento do Sufferage é igual ao do Dynamic FPLTF, alocando as maiores tarefas inicialmente por tenderem a “sofrer” mais. Entretanto, nem sempre isso ocorre como poderá ser visto a seguir em um exemplo de seu funcionamento.

Exemplo:

Supondo que existam inicialmente 5 tarefas a serem executadas com os seguintes tamanhos (i.e., tempo de execução em uma máquina com velocidade 1, sem carga): 1000, 600, 400, 200 e 100 segundos. Três máquinas (A, B e C) estão à disposição para a execução das tarefas e suas características estão descritas na Tabela 1. A Velocidade Relativa Real (VRR) representa o valor da Velocidade Relativa da máquina descontando o efeito da carga que a máquina sofre por outros usuários e aplicações. O tempo de execução de uma tarefa é calculado através da divisão do seu custo pela VRR. Se, por exemplo, a tarefa de 1000 segundos fosse alocada para o *host* B, gastaria 1000 segundos para ser completada. Entretanto,

se esta mesma tarefa fosse alocada para o *host* A, gastaria apenas 500 segundos. Isso se deve ao fato da Velocidade Relativa Real do *host* A (VRR = 2) ser duas vezes maior que a do *host* B (VRR =1).

Host	Velocidade Relativa	Carga	Velocidade Relativa Real
A	5	60 %	2
B	2	50 %	1
C	1	20 %	0,8

Tabela 1. Características das máquinas.

Tarefa	Host	CT no Laço 1	Valor Sufferage	CT no Laço 2	Valor Sufferage	CT no Laço 3	Valor Sufferage
1 (1000)	A	500 (1º)	500	X	X	X	X
	B	1000 (2º)		X		X	
	C	1250		X		X	
2 (600)	A	300 (1º)	300	500+300=800	150	X	X
	B	600 (2º)		600 (1º)		X	
	C	750		750 (2º)		X	
3 (400)	A	200 (1º)	200	500+200=700	100	500+200=700 (2º)	200
	B	400 (2º)		400 (1º)		600+400=1000	
	C	500		500 (2º)		500 (1º)	
4 (200)	A	100 (1º)	100	500+100=600	50	500+100=600 (2º)	350
	B	200 (2º)		200 (1º)		600+200=800	
	C	250		250 (2º)		250 (1º)	
5 (100)	A	50 (1º)	50	500+50=550	25	500+50=550 (2º)	500
	B	100 (2º)		100 (1º)		600+100=700	
	C	125		125 (2º)		50 (1º)	
		Tarefa 1 ➔ Máquina A		Tarefa 2 ➔ Máquina B		Tarefa 5 ➔ Máquina C	

Tabela 2. Exemplo do algoritmo Sufferage em funcionamento.

A Tabela 2 mostra com detalhes o comportamento do Sufferage no decorrer do escalonamento de uma aplicação. No primeiro momento (Laço 1), nenhuma tarefa foi alocada ainda e o *Completion Time* das tarefas é calculado baseado apenas na velocidade relativa da máquina e sua carga. Ao final deste laço, a Tarefa 1 é escolhida para ser alocada para o *host* A por ter proporcionado o maior valor

sufferage (segundo menor CT – menor CT) entre as tarefas. A maior tarefa foi alocada comprovando o privilégio dado pelo Sufferage às tarefas maiores.

No Laço 2, novamente, a maior tarefa é alocada por ter o maior valor *sufferage*. Pode-se notar também, que já neste laço, o CT das tarefas no *host* A é calculado levando em consideração o provável tempo necessário para executar a Tarefa 1, alocada anteriormente. No laço seguinte, a Tarefa 5 (menor tarefa dentre as restantes) é escalonada para o *host* C. Diferentemente do Dynamic FPLTF, nesse caso. A tarefa que sofre mais é a menor e não a maior. O fato das outras máquinas já possuírem tarefas alocadas faz com que o valor *sufferage* das tarefas menores tenda a ser maior. O exemplo mostrado acima comprova a afirmação de que nem sempre o Sufferage escolhe a maior tarefa disponível para ser escalonada.

Note que o grande problema deste algoritmo é o mesmo do Dynamic FPLTF. Ele necessita de muita informação para calcular os CTs das tarefas. Necessita ter conhecimento do tamanho das tarefas, da carga e da velocidade das máquinas. Em um Grid, essas informações são difíceis de serem obtidas com precisão e nem sempre estão disponíveis.

Workqueue

O Workqueue é um escalonador que não se baseia em conhecimento no sentido em que ele não precisa de nenhum tipo de informação para o escalonamento das tarefas. As tarefas são escolhidas de maneira arbitrária do “saco de tarefas” e enviadas aos processadores, tão logo eles se tornem disponíveis. Depois que a tarefa é completada, o processador envia de volta o resultado e o escalonador atribui uma nova tarefa ao processador. Em outras palavras, o escalonador inicia com o envio de uma tarefa para cada máquina disponível. Logo que uma máquina completa a execução de sua tarefa, o escalonador atribui outra tarefa àquela máquina (desde que ainda existam tarefas a serem processadas).

A idéia por trás dessa abordagem é que mais tarefas serão escalonadas para as máquinas rápidas/desocupadas enquanto as máquinas lentas/carregadas processarão uma carga menor de trabalho. A grande vantagem aqui é que o

Workqueue não depende de informações de performance. Infelizmente, o Workqueue não consegue atingir uma performance equiparável aos escalonadores que possuem conhecimento sobre o ambiente, como será visto na seção 5.2. O problema com o Workqueue surge quando uma tarefa grande é alocada para uma máquina lenta perto do final da execução da aplicação. Como o Workqueue não possui um mecanismo para evitar essa situação, isso pode ocorrer com certa frequência. Dessa maneira, o término da execução da aplicação pode ser consideravelmente atrasado por depender do término desta tarefa.

5 Escalonamento com Replicação de Tarefas

Devido à dificuldade em se obter boas informações sobre a performance de máquinas no Grid e sobre as tarefas das aplicações, foi decidido, neste trabalho, a concepção de um escalonador que não se baseasse em informações do ambiente. A solução proposta é chamada de *Workqueue com Replicação* (WQR), uma vez que ela basicamente adiciona a replicação de tarefas ao algoritmo Workqueue clássico. Entretanto, sua performance é equivalente a de soluções que têm um grande conhecimento sobre o ambiente (que não são factíveis na prática). O preço pago é o consumo de mais ciclos de CPU.

O algoritmo WQR usa a replicação de tarefas para lidar com a heterogeneidade das máquinas e tarefas, e também com a variação dinâmica na disponibilidade dos recursos devido à carga gerada por outros usuários no Grid. Vale ser ressaltado que o WQR assume que as tarefas são idempotentes, não geram efeitos colaterais, como alterações em bancos de dados (ex. incrementos de valores), de modo a prevenir a inconsistência de dados que as réplicas poderiam gerar. Essa hipótese é válida em ambientes como Grids pelo fato das máquinas estarem bastante dispersas geograficamente. Em ambientes assim, não é muito comum a utilização de aplicações cujas tarefas façam acesso (leitura/escrita) em bancos de dados por ter um alto custo. Em Grids, aplicações BoT tipicamente se comunicam por arquivos de entrada e saída.

O início da execução do WQR é igual a do Workqueue clássico e continua da mesma forma até que o “saco de tarefas” esteja vazio. Nesse momento, no Workqueue comum, as máquinas que completam suas tarefas ficariam desocupadas durante o resto da execução da aplicação. Usando a abordagem de replicação, essas máquinas são designadas a executar réplicas de tarefas que ainda estão rodando. As tarefas são replicadas até que um número máximo de réplicas

predefinido seja atingido. Para expressar o limite de réplicas do algoritmo é utilizado um complemento em seu nome, WQR 2x, por exemplo, permite até 2 réplicas (2 tarefas iguais em execução), WQR 3x permite até 3 réplicas e assim por diante. Quando uma réplica de uma tarefa termina, suas outras réplicas são canceladas para evitar o gasto de ciclos de CPU sem necessidade.

Com essa abordagem, a performance tende a ser melhorada em situações em que tarefas grandes atrasam o término da aplicação por terem sido alocadas para máquinas lentas/carregadas. Quando uma tarefa é replicada, há uma chance maior dessa réplica ser escalonada para uma máquina mais rápida e/ou menos carregada.

O ponto positivo dessa solução é que ela não utiliza informações sobre as máquinas (velocidade e carga) e tamanhos das tarefas para realizar o escalonamento. O ponto negativo é que para atingir uma boa performance essa abordagem gasta mais CPU com as réplicas de tarefas que são canceladas (a CPU gasta não foi útil para o processamento da aplicação). Como poderá ser visto na Seção 5.2, quando a granularidade da aplicação (número de máquinas disponíveis por tarefa) é baixa, o WQR gasta poucos ciclos de CPU a mais comparado à quantidade de ciclos necessária para executar a aplicação (menos de 7% mais). Com uma alta granularidade de tarefas, o consumo extra de recursos do WQR pode ser significante (podendo ultrapassar 100% nos experimentos realizados). Nesses casos, a replicação pode ser limitada a 2 réplicas, mantendo o desperdício de ciclos abaixo dos 50% e, ainda assim, atingindo uma boa performance.

Em resumo, WQR tenta minimizar o problema do escalonamento de tarefas independentes em ambientes heterogêneos, evitando a dependência de qualquer tipo de informação, como tamanho de tarefa e performance de processador. Esse tipo de informação é bastante útil para elaborar um bom plano de escalonamento. Entretanto, obter essas informações pode ser difícil devido à grande distribuição geográfica dos recursos natural dos Grids, a limitações administrativas e ao desconhecimento do tempo de execução antes que tarefas executem. Além disso, quando é possível obter essa informação, as previsões para recursos computacionais e de rede nem sempre são precisas e confiáveis.

5.1 Experimentos Realizados

Para avaliar a performance dos algoritmos foi utilizada a técnica de simulação. Para executar os experimentos, foi usado o SimGrid [45]. Este *software* provê funções básicas para a simulação de aplicações distribuídas em ambientes heterogêneos e dinâmicos como Grids. Essa ferramenta foi desenvolvida para o estudo de algoritmos de escalonamento e políticas de gerenciamento de recursos em Grids Computacionais.

Neste trabalho, assume-se que os tempos de transferência de dados através da rede são desprezíveis, ou seja, assume-se que as aplicações a serem analisadas lidam com poucos dados de entrada/saída (i.e., as tarefas são basicamente consumidoras de CPU). Alternativamente pode também se assumir que, quando a aplicação utiliza grandes quantidades de dados, estes são previamente distribuídos pelo Grid [1][43]. Essa estratégia de distribuição prévia dos dados tem a finalidade de diminuir ao máximo as transferências de dados durante a execução da aplicação. Outro ponto a ser destacado neste trabalho é que os algoritmos baseados em informações sobre o ambiente e as aplicações (Sufferage e Dynamic FPLTF) são alimentados com informações perfeitas, situação que é praticamente impossível no mundo real.

O objetivo principal dos experimentos é comparar a performance de diferentes escalonadores para aplicações BoT em Grids Computacionais. Os experimentos visam avaliar a influência da heterogeneidade das máquinas do Grid (diferentes velocidades), a heterogeneidade das tarefas (diferentes tamanhos) e a granularidade da aplicação (diferentes relações máquinas por tarefa). A heterogeneidade máquinas possui cinco níveis (desde um Grid homogêneo a um Grid bastante heterogêneo) como poderá ser visto ainda nessa seção. A heterogeneidade de tarefas também possui cinco níveis (desde uma aplicação homogênea, i.e. tarefas com mesmo tamanho, a uma aplicação bastante heterogênea) como será detalhado mais adiante nessa seção. Em relação à granularidade da aplicação, esta é determinada a partir de uma variação no tamanho médio das tarefas. Quatro grupos principais de aplicações são gerados e as médias do tamanho das tarefas desses grupos são 1.000, 5.000, 25.000 e 125.000 segundos, seguindo uma esca-

la exponencial. Todas aplicações geradas para os experimentos possuem o mesmo tamanho. Sendo assim, a aplicação cujo tamanho médio de tarefas é 1.000 segundos (grão pequeno) terá muito mais tarefas que a aplicação com tamanho médio de tarefas de 125.000 segundos (grão enorme).

Todos os experimentos têm um valor fixo para o poder do Grid (soma da capacidade dos processadores de todas as máquinas) e tamanho da aplicação. O valor fixo para o poder do Grid é 1.000. O Grid poderia ser composto, por exemplo, por 500 máquinas com poder 2. O poder da máquina indica o quão rápido uma máquina pode executar uma tarefa, sendo um valor relativo a uma máquina com poder 1. Uma máquina com poder 2 pode executar uma "tarefa de 10 segundos" em 5 segundos. O valor fixo para o tamanho da aplicação é de 3.600.000 segundos. Em um mundo perfeito[5], esta aplicação levaria exatamente 1 hora (3.600 segundos) para ser completada em um Grid de poder 1.000. A estratégia de fixar os valores do poder do Grid e do tamanho da aplicação tem como objetivo deixar que os tempos de execução da aplicação sejam influenciados apenas pelos escalonadores. Em cada um dos experimentos, todos os algoritmos são executados com o mesmo conjunto de máquinas e tarefas.

Variação na Granularidade das Aplicações

Ao variar o tamanho médio das tarefas (1.000, 5.000, 25.000 e 125.000 segundos), o número de tarefas das aplicações também está sendo variado (em valores aproximados: 3.600, 720, 144 e 29 tarefas respectivamente em cada grupo) e consequentemente a relação máquinas por tarefa é alterada como pode ser visto na Tabela 3. Isso acontece porque o número de máquinas no Grid mantém-se praticamente constante em todos os experimentos com sua média oscilando sempre em torno de 100 máquinas.

[5] Neste cenário as máquinas estão 100% livres e livres de carga. Além disso, o escalonamento é feito de tal forma que todas as tarefas terminam simultaneamente.

Tamanho médio das tarefas (segundos)	Quantidade de tarefas	Máquinas por tarefa (Granularidade)
1000	3.600	0,028
5000	720	0,14
25000	144	0,69
125000	29	3,45

Tabela 3. Granularidade das Aplicações.

Essas relações indicam a quantidade de máquinas disponíveis para executar cada tarefa. O objetivo principal dessa variação na relação máquinas por tarefa é avaliar o impacto na performance dos algoritmos em situações onde existam muito mais tarefas que máquinas (100 máquinas para 3600 tarefas) e depois alterar essa situação de forma gradativa até chegar o ponto onde existam mais máquinas que tarefas (100 máquinas para 29 tarefas).

Simulação da Heterogeneidade das Máquinas

A velocidade das máquinas de cada experimento varia de acordo com uma distribuição uniforme de tal forma que a média de velocidade de todas as máquinas do Grid seja aproximadamente 10. Essa definição tem como intuito fazer com que todos os Grids utilizados nos experimentos tenham praticamente a mesma quantidade de máquinas para que o escalonador seja o principal responsável pelo desempenho da aplicação.

Para a definição de uma faixa de valores para a velocidade das máquinas do Grid, a Lei de Moore [13] foi utilizada como fonte de inspiração. A Lei de Moore diz que a velocidade das máquinas praticamente dobra a cada 18 meses. Para que se pudesse cobrir até aproximadamente seis anos de vida útil das máquinas,

os níveis de heterogeneidade definidos foram 1, 2, 4, 8 e 16[6]. O nível de heterogeneidade do experimento é definido pela constante **hm** e representa a variação máxima de velocidade das máquinas. Em particular, **hm** = 1, significa que o Grid é homogêneo (todas as máquinas têm velocidade 10). Quando **hm** é igual a 2, a velocidade das máquinas varia de acordo com a distribuição uniforme **U(9, 11)** (2 unidades de diferença e mantendo 10 de média). Para **hm** = 4, a velocidade das máquinas varia de acordo com a distribuição uniforme **U(8, 12)** (4 unidades de diferença e mantendo média 10) e dessa mesma forma seguem as distribuições para os outros graus de heterogeneidade (8 e 16). As máquinas são adicionadas ao Grid de acordo com cada distribuição até que a soma das suas velocidades atinja o poder do Grid que sempre é 1.000 (um total de aproximadamente 100 máquinas).

A carga de cada máquina é simulada através de informações obtidas pelo NWS (*Network Weather System*) [53] em sistemas reais. O NWS é um serviço de monitoração distribuído que grava e prevê a performance de vários recursos computacionais e de rede ao longo do tempo. O serviço é baseado em um conjunto de sensores de performance que reúnem informações necessárias para realizar as previsões através de modelos numéricos. O uso do NWS gera *traces*, arquivos que contém a largura de banda e latência de componentes de rede, percentual de CPU disponível nas máquinas, memória não-paginada, entre outros dados. O Simgrid, então, usa os *traces* relativos ao percentual de CPU disponível para fazer com que as máquinas do experimento se comportem de forma similar a máquinas reais.

A Figura 5 mostra um gráfico com a especificação da *workload* utilizada para simular a carga nas máquinas dos experimentos. Os traces apresentam uma característica típica de ambientes acadêmicos onde a maioria das máquinas passa a maior parte do tempo ociosa (cerca de 50% de acordo com o gráfico). Uma

[6] No primeiro momento **hm** é 1, Após 18 meses seu valor é dobrado para 2, 18 meses depois é dobrado para 4 e assim por diante até que chegue em 16, totalizando 72 meses (6 anos). Seis anos é um período razoável para ser analisado, pois as máquinas sofrem uma depreciação de 20% ao ano.

outra parte (aproximadamente 40%) é utilizada para edição de textos, programação, navegação na Internet, entre outras tarefas que não ocupam muito tempo da CPU. Os 10% restantes é composto por máquinas que realmente estão gastando CPU, executando aplicações de computação intensiva como simulações, por exemplo.

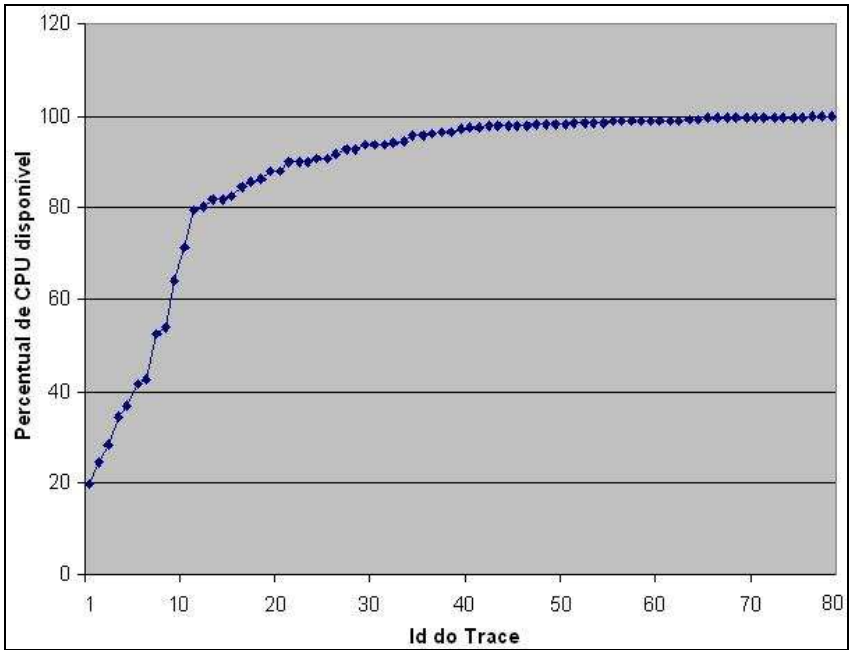

Figura 5. *Workload* dos *traces* utilizados para simular a carga nas máquinas.

Simulação da Heterogeneidade das Tarefas

É válido lembrar que os possíveis tamanhos médios das tarefas são: 1.000, 5.000, 25.000 e 125.000 segundos. Para simular a heterogeneidade das tarefas em cada um dos grupos, o tamanho das tarefas varia, mas a média de tamanho permanece a mesma do grupo. A variação no tamanho das tarefas pode ser de 0%, 25%, 50%, 75% e 100% relativo ao tamanho médio das tarefas do grupo. As aplicações onde as tarefas têm tamanho médio de 5.000 segundos (tempo de execução em uma máquina com velocidade 1), por exemplo, seria subdividido em cinco grupos. Em um grupo todas as tarefas teriam o mesmo tamanho (5.000 segundos, variação de 0%), sendo, portanto tarefas homogêneas. No grupo onde a variação é de 25%, o tempo de execução de cada tarefa varia de acordo com a

distribuição uniforme **U(4375, 5625)** (de 12,5% menos que 5.000 a 12,5% mais que 5.000). Para o grupo onde a variação é de 50%, os tempos de execução das tarefas seguem a distribuição uniforme **U(3750, 6250)**, (de 25% menos que 5.000 a 25% mais que 5.000). Os mesmos cálculos são feitos para definir a distribuição dos tempos de execução das tarefas dos outros subgrupos, sempre com o intuito de manter a mesma média do grupo principal. As tarefas são adicionadas à aplicação de acordo com cada distribuição até que a soma de seus tamanhos atinja o tamanho da aplicação que sempre é 3.600.000 segundos.

Com os experimentos definidos dessa forma, podemos avaliar não apenas a influência da heterogeneidade dos recursos do Grid e das tarefas da aplicação, mas também um outro ponto de bastante relevância a ser analisado que é a granularidade das aplicações.

5.2 Análise de resultados

Essa seção mostra o desempenho do WQR através de uma análise comparativa com os algoritmos Sufferage, Dynamic FPLTF e Workqueue. A Tabela 4 mostra o tempo médio que cada algoritmo levou para a execução das aplicações. Nessa tabela estão sumariados os dados relativos aos quatro tipos de aplicações e seus cinco níveis de heterogeneidade, bem como os cinco níveis de heterogeneidade das máquinas. Existem 25 cenários distintos (combinações dos 5 níveis de heterogeneidade de máquinas com os 5 níveis de heterogeneidade de tarefas) onde cada grupo de aplicação deve ser executado. Para cada cenário foram executadas aproximadamente 80 simulações, totalizando aproximadamente 2000 simulações para cada grupo de aplicações. Somando os 4 grupos de aplicações, foram realizadas um total de 7885 simulações. As simulações foram feitas em paralelo usando MyGrid[7] [12][35][39], que até então utilizava o Workqueue como escalonador de tarefas.

[7] MyGrid provê um ambiente de execução global que possibilita a execução de aplicações paralelas do tipo Bag-of-Tasks em qualquer máquina que o usuário tiver acesso.

	Sufferage	Dynamic FPLTF	Workqueue	WQR 2x	WQR 3x	WQR 4x
Média	13530.26	12901.78	23066.99	12835.70	12123.66	11652.80
Desvio Padrão	9556.55	9714.08	32655.85	10739.50	9434.70	8603.06
Média Geométrica	10982.35	10341.13	16262.94	9320.29	9012.52	8836.56

Tabela 4. Resultados gerais dos algoritmos.

Ao apresentar os dados de forma tão resumida, a média do Workqueue é bastante prejudicada, porque a depender do tipo de aplicação, os tempos de execução das aplicações podem ser consideravelmente elevados, dominando a média. A média geométrica de cada um dos algoritmos foi calculada com a intenção de reduzir a influência de valores extremos, de modo a obter números mais condizentes com a realidade [34].

O que se pode perceber através de uma rápida análise da Tabela 4 é que o WQR 4x teve o melhor desempenho entre os algoritmos e o Workqueue teve o pior rendimento. O primeiro ponto que vale ser ressaltado é que ao usar a estratégia de replicação sobre o Workqueue, a melhora na performance deste algoritmo é considerável, chegando a quase o dobro, em média. O segundo ponto a ser ressaltado é que mesmo não tendo a melhor performance, o WQR 2x pode ser uma solução muito boa para o escalonamento de tarefas em ambientes onde não seja possível obter informações das máquinas no Grid e das tarefas. A performance do WQR 2x é equivalente a dos algoritmos que usam este tipo de informação (Sufferage e Dynamic FPLTF) para realizarem os escalonamento. Além disso, o WQR 2x tem uma vantagem sobre o WQR 4x que é, em média, desperdiçar menos ciclos de CPU pelo fato de replicar menos as tarefas como poder ser visto na Tabela 5.

	WQR 2x	WQR 3x	WQR 4x
Média	23.55%	36.32%	48.87%

Tabela 5. Média de ciclos de CPU desperdiçados.

Entretanto, com o alto desvio padrão dos algoritmos observado na Tabela 4 (próximos ou até superiores à própria média), estes sugerem que as médias não devem ser analisadas absolutamente, como resultados finais. A análise dos dados deve ser realizada de uma forma mais detalhada como será visto mais adiante.

Impacto da Granularidade da Aplicação

A Figura 6 mostra a média aritmética dos tempos de execução obtida por cada um dos algoritmos de escalonamento para os quatro tipos de aplicações. Cada ponto é a média dos cinco níveis de heterogeneidade de máquinas e os cinco níveis de heterogeneidade de tarefas.

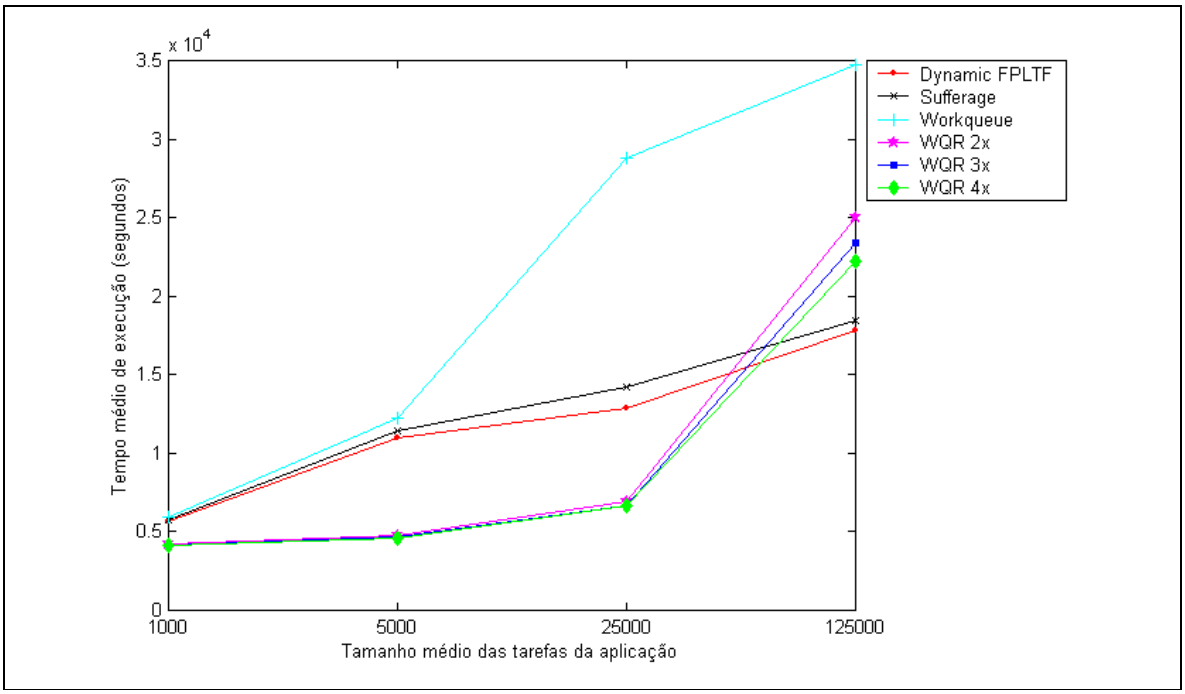

Figura 6. Desempenho por tipo de aplicação.

Através do gráfico, pode-se notar que, em situações onde o tamanho médio das tarefas é pequeno (menor relação máquinas por tarefa), os algoritmos têm um desempenho muito parecido. A granularidade menor da aplicação faz com que o final da execução (momento onde todas as tarefas já foram escalonadas, mas existem tarefas sendo executadas) represente uma pequena fração da execução. Desse modo, a alocação de uma tarefa para uma máquina lenta neste momento não causa grande impacto no tempo final de execução da aplicação. Ao aumentar o tamanho médio das tarefas (aumentar a relação máquinas por tarefa), a tendência é que todos algoritmos tenham uma queda de rendimento, pois a granularidade maior das tarefas dificulta o escalonamento.

O Sufferage e o Dynamic FPLTF, algoritmos que utilizam informações sobre as máquinas e tarefas para tomar decisões de escalonamento, têm um comportamento relativamente constante em relação ao aumento no tamanho médio das tarefas (aumento na relação máquinas por tarefa). Como pode ser visto na Figura 6, à medida que o tamanho médio das tarefas aumenta eles sofrem uma queda quase linear no seu rendimento. Quanto maior o tamanho de uma tarefa, maior é a chance de que a máquina que está executando-a sofra alterações na sua performance durante sua execução. Isso ocorre devido ao fato da máquina estar sendo compartilhada por outros usuários, e porventura pode se tornar extremamente lenta a ponto de prejudicar a execução da aplicação. Esse fato indica que, nem sempre, escalonar uma tarefa grande para uma máquina rápida naquele momento é a melhor escolha.

O Workqueue consegue acompanhar o nível de performance do Sufferage e do Dynamic FPLTF em situações onde a granularidade das tarefas é pequena (1.000 e 5.000). Com as tarefas tendo uma média de tamanho maior (25.000 e 125.000), o Workqueue sofre uma queda considerável no seu rendimento. A granularidade maior da aplicação faz com que o final da execução (momento onde não há mais tarefas para serem escalonadas, mas existem tarefas sendo executadas) represente uma fração maior da execução. Desse modo, a alocação de uma tarefa para uma máquina lenta neste momento tem um impacto maior no tempo final de execução da aplicação.

As três versões do WQR tiveram o melhor rendimento entre todos os algoritmos em situações onde existiam mais tarefas do que máquinas no Grid (relação máquinas por tarefa < 1). Esse melhor desempenho ocorre devido à estratégia de replicação, porque mesmo que uma tarefa seja inicialmente escalonada para uma máquina lenta, essa tarefa pode ser replicada em uma máquina rápida e ser completada antes da “original”. Além disso, mesmo que isso não ocorra, o prejuízo no tempo final de execução da aplicação pode não ser muito grande porque as tarefas não são muito grandes. Para as aplicações onde as tarefas eram maiores e a relação máquinas por tarefas era maior que 1, o WQR já entra em modo de replicação desde o início. Nesse caso o WQR não conseguiu um bom rendimento, in-

clusive sendo pior que o Sufferage e Dynamic FPLTF. Essa queda em seu rendimento se deve ao fato da granularidade das tarefas ser muito grande. Neste caso, pode acontecer que, no fim do escalonamento, uma dessas tarefas grandes tenha sido alocada para uma máquina lenta e sua(s) réplica(s) também, prejudicando bastante a execução da aplicação.

Impacto da Heterogeneidade das Máquinas

A Figura 7 exibe a influência da heterogeneidade das máquinas no Grid sobre a performance dos algoritmos. Cada ponto leva em consideração todos os níveis de heterogeneidade de tarefas e tipos de aplicações.

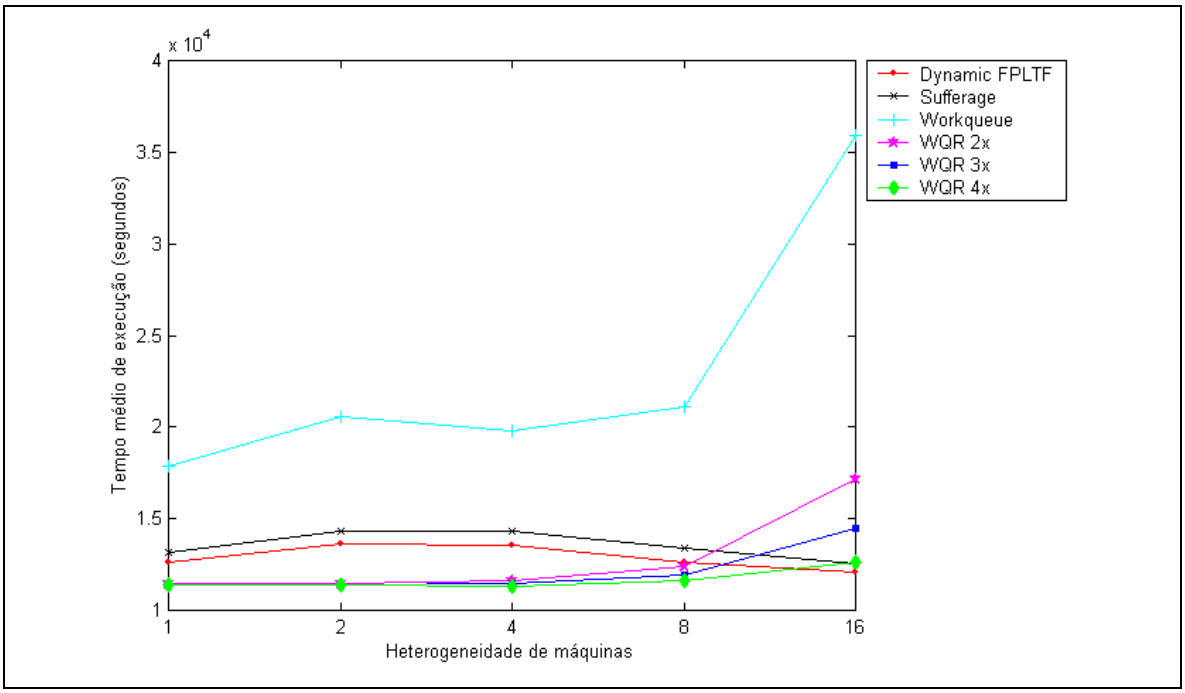

Figura 7. Desempenho por heterogeneidade de máquinas.

O Workqueue não alcançou um bom desempenho, mas demonstrou uma certa constância em sua performance enquanto o nível de heterogeneidade de máquinas não passava de 8. Com a heterogeneidade de máquinas sendo 16, sua performance foi muito ruim comparada aos demais algoritmos. Isso acontece porque quanto maior essa heterogeneidade, maior pode ser a diferença de uma máquina rápida para uma lenta. A escolha de uma máquina muito lenta para executar

uma tarefa grande tem grande impacto no tempo de execução da aplicação e o Workqueue não tem nenhum mecanismo para evitar isso.

O WQR teve um comportamento de certa forma similar ao Workqueue. Até o nível de heterogeneidade de máquinas 8, teve sua performance praticamente inalterada. No nível de heterogeneidade de máquinas 16, o WQR sofreu uma queda de rendimento, mas essa queda foi bem menos acentuada que a do Workqueue. Ainda pode-se notar que quanto mais replicações são utilizadas, menor é a perda de desempenho do WQR. Essa queda no desempenho observada no Workqueue e WQR no nível de heterogeneidade 16 deve-se ao fato de que as máquinas podem ser muito lentas ou muito rápidas. A diferença de velocidade entre as máquinas é muito grande. Enquanto, no nível 8, por exemplo, a máquina mais lenta possui velocidade 6, a máquina mais lenta do nível 16 tem velocidade 2. Se uma tarefa for alocada para uma máquina tão lenta assim no final do escalonamento, o prejuízo para o tempo de execução da aplicação pode ser grande.

O Sufferage e o Dynamic FPLTF tiveram um comportamento quase inverso ao WQR. Ao aumentar o nível de heterogeneidade das máquinas no Grid, a performance desses algoritmos melhora, chegando até a ultrapassar o WQR no nível 16. Essa melhor performance obtida pelo Sufferage e Dynamic FPLTF ocorre devido à capacidade que estes algoritmos têm de escolherem máquinas mais poderosas para executarem tarefas custosas. Dessa forma, as tarefas menores que porventura venham ser escalonadas para máquinas lentas não terão impacto no tempo final de execução da aplicação.

O gráfico apresentado na Figura 8 tem como base os mesmos dados utilizados na Figura 7, mas os resultados referentes às aplicações com média de tamanho de tarefas igual a 125.000 segundos não foram utilizados nos cálculos. Essa exclusão dos dados relativos às aplicações com tamanhos médios de tarefas de 125.000 segundos tem como objetivo destacar a superioridade do WQR em relação aos outros algoritmos em situações onde a relação máquinas por tarefa é menor que 1.

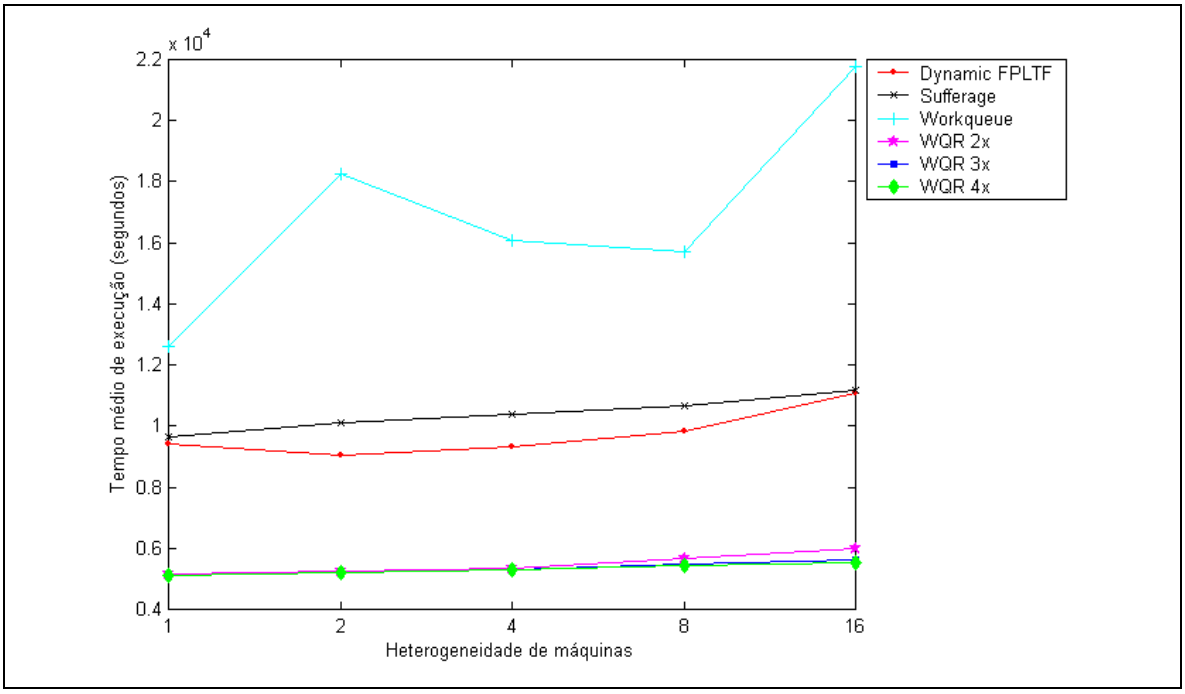

Figura 8. Desempenho por heterogeneidade de máquinas de todos os tipos de aplicações sem levar em conta as aplicações com tamanho médio de tarefas de 125.000 segundos.

Com uma granularidade maior na aplicação (aplicações com média de tamanho de tarefas sendo 125.000 segundos) a situação é diferente. Na Figura 9 apenas os dados referentes a esse tipo de aplicações foram utilizados. Neste caso o fato do algoritmo ter a capacidade de priorizar tarefas grandes para máquinas rápidas é bastante relevante, fazendo com que o Sufferage e o Dynamic FPLTF consigam resultados melhores que o WQR. Como os resultados dessa categoria de aplicações possuem médias mais altas e valores altos dominam as médias, o gráfico geral mostrado na Figura 7 fica com o aspecto baseado nos resultados dessa categoria.

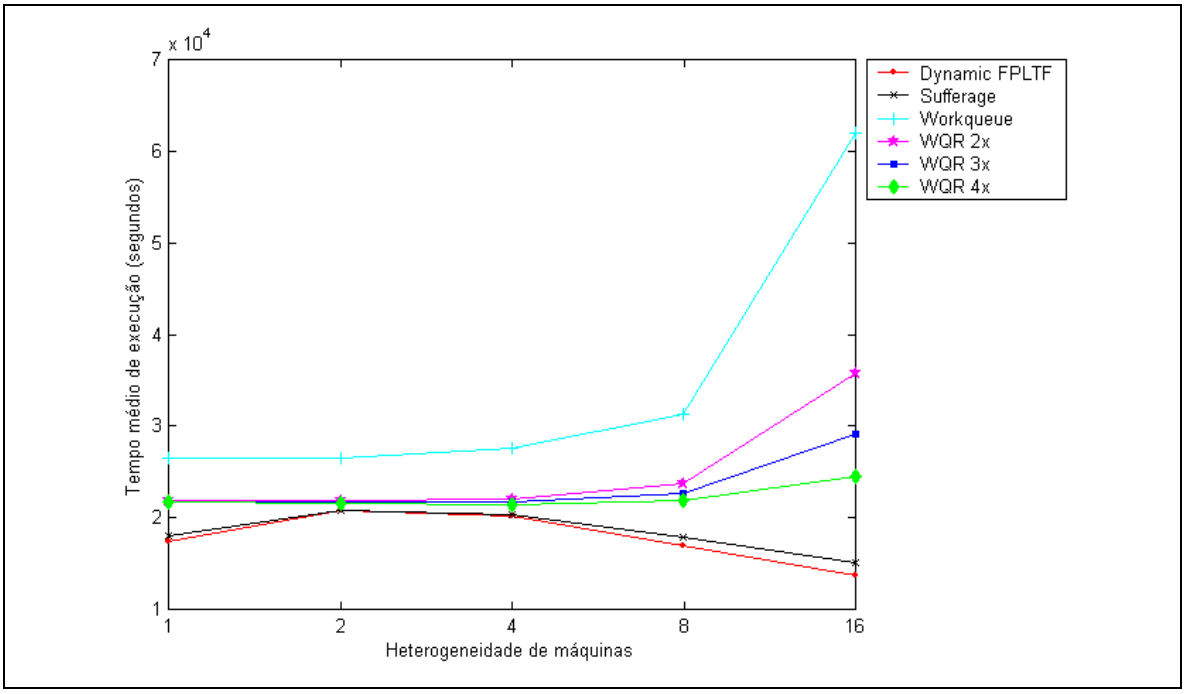

Figura 9. Desempenho por heterogeneidade de máquinas das aplicações com tamanho médio de tarefas de 125.000 segundos.

Impacto da Heterogeneidade das Tarefas

A Figura 10 mostra a influência da heterogeneidade das tarefas sobre o desempenho dos algoritmos. De forma geral, a heterogeneidade das tarefas mostrou pouco impacto sobre os algoritmos estudados (vale lembrar que Sufferage e Dynamic FPLTF têm informações perfeitas). Neste gráfico, o Workqueue confirma mais uma vez seu comportamento bastante irregular e com um rendimento muito inferior aos outros algoritmos estudados. O Sufferage e o Dynamic FPLTF obtiveram resultados parecidos, mas o segundo sofreu menos o impacto da diversidade no tamanho das tarefas da aplicação. O fato de o Dynamic FPLTF ordenar as tarefas a priori por ordem de tamanho (as tarefas maiores são as primeiras a serem alocadas) o leva a atingir melhores resultados que o Sufferage. Ao utilizar-se do fator “tarefa que sofre menos” para realizar o escalonamento, o Sufferage nem sempre prioriza as tarefas maiores. A versão 2x do WQR teve uma performance praticamente igual à do Dynamic FPLTF mostrando ser uma boa alternativa quando não é possível obter informações sobre o ambiente e a aplicação. Ao aumentar o nível de replicação (3x e 4x), o WQR consegue resultados ainda melhores al-

cançando uma performance superior aos algoritmos que lidam com informações para definir o plano de escalonamento.

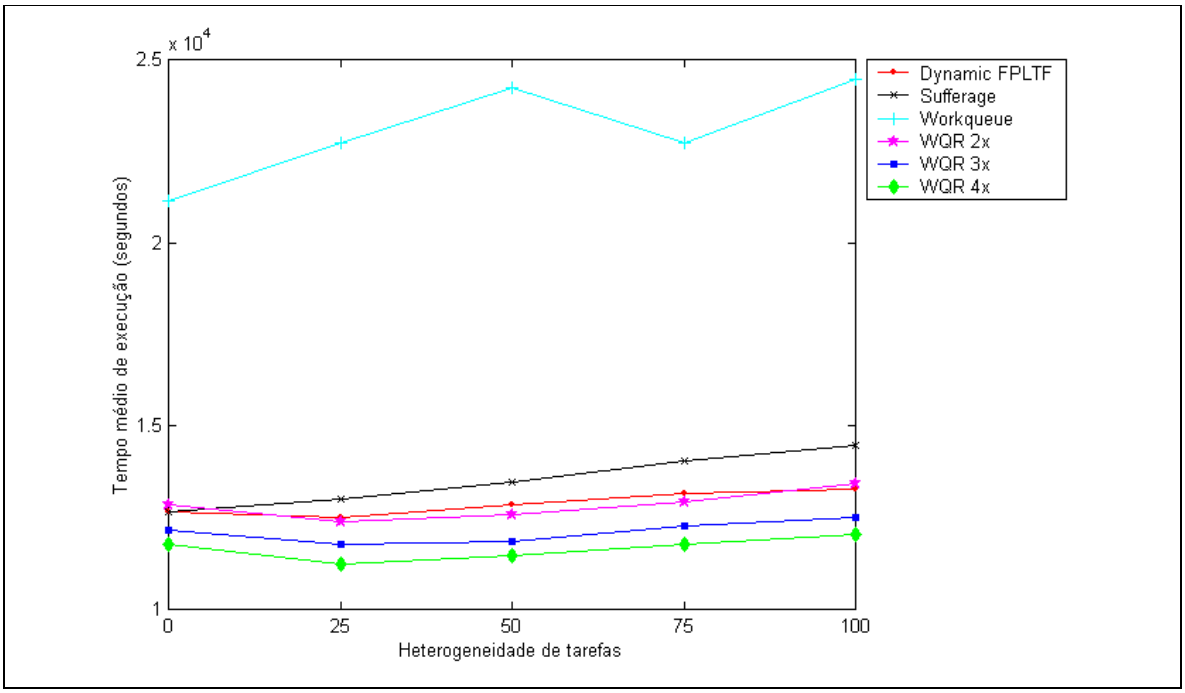

Figura 10. Desempenho por heterogeneidade de tarefas.

Na Figura 11 é apresentado um gráfico com a mesma base de dados do gráfico da Figura 10 excetuando-se os resultados das aplicações com tamanho médio de tarefas igual a 125.000 segundos. Como mencionado anteriormente, essa exclusão visa destacar o excelente rendimento que o WQR atinge em situações onde a relação máquinas por tarefa é menor que 1. O gráfico da Figura 11 mostra que o WQR teve uma performance aproximadamente duas vezes melhor que o Sufferage e o Dynamic FPLTF. Entretanto, o Workqueue não obteve um bom rendimento e apresentou um comportamento irregular em função da variação na heterogeneidade das tarefas. O WQR corrige esse problema de instabilidade do Workqueue através do uso da replicação de tarefas e praticamente não é afetado pela diversidade nos tamanhos das tarefas da aplicação.

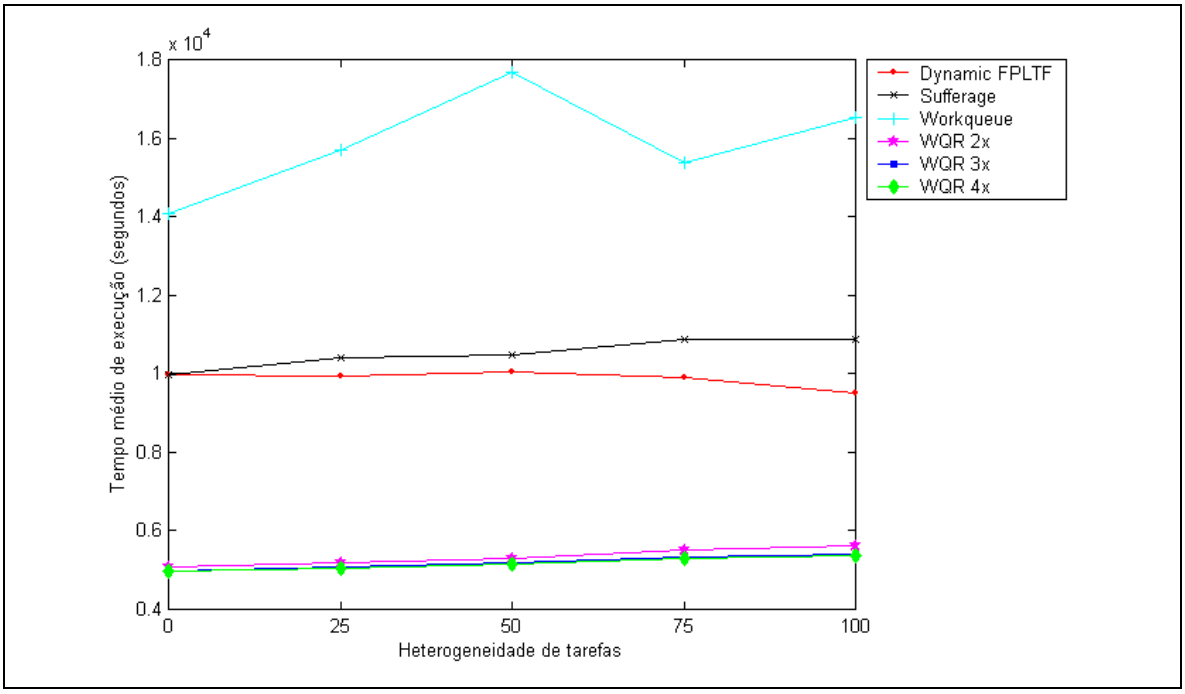

Figura 11. Desempenho por heterogeneidade de tarefas de todos os tipos de aplicações sem levar em conta as aplicações com tamanho médio de tarefas de 125.000 segundos.

A Figura 12 apresenta um gráfico apenas com os dados das aplicações com média de tamanho de tarefas igual 125.000 segundos. Por este gráfico fica evidente a grande vantagem de, em situações onde há poucas tarefas em relação ao número de máquinas, ter informações sobre o ambiente e a aplicação para realizar o escalonamento.

Os algoritmos Sufferage e Dynamic FPLTF atingem um rendimento melhor que o WQR justamente pelo fato de trabalharem com informações sobre a performance das máquinas e tamanho das tarefas da aplicação. O Workqueue tem uma performance muito ruim porque como o grão é muito grande, a parte final da execução (quando não há mais tarefas para serem escalonadas) representa toda a execução. Assim, ao alocar uma tarefa grande para uma máquina lenta nesse momento causa um prejuízo maior do que em aplicações com menor granularidade. Apesar do WQR também estar sujeito a esse problema, existe também a possibilidade dessa tarefa grande ser replicada em uma máquina rápida. Dessa maneira, o WQR consegue ter uma performance muito melhor que o Workqueue,

mas sua performance, mesmo da sua versão 4x, fica um pouco distante das performances do Sufferage e do Dynamic FPLTF.

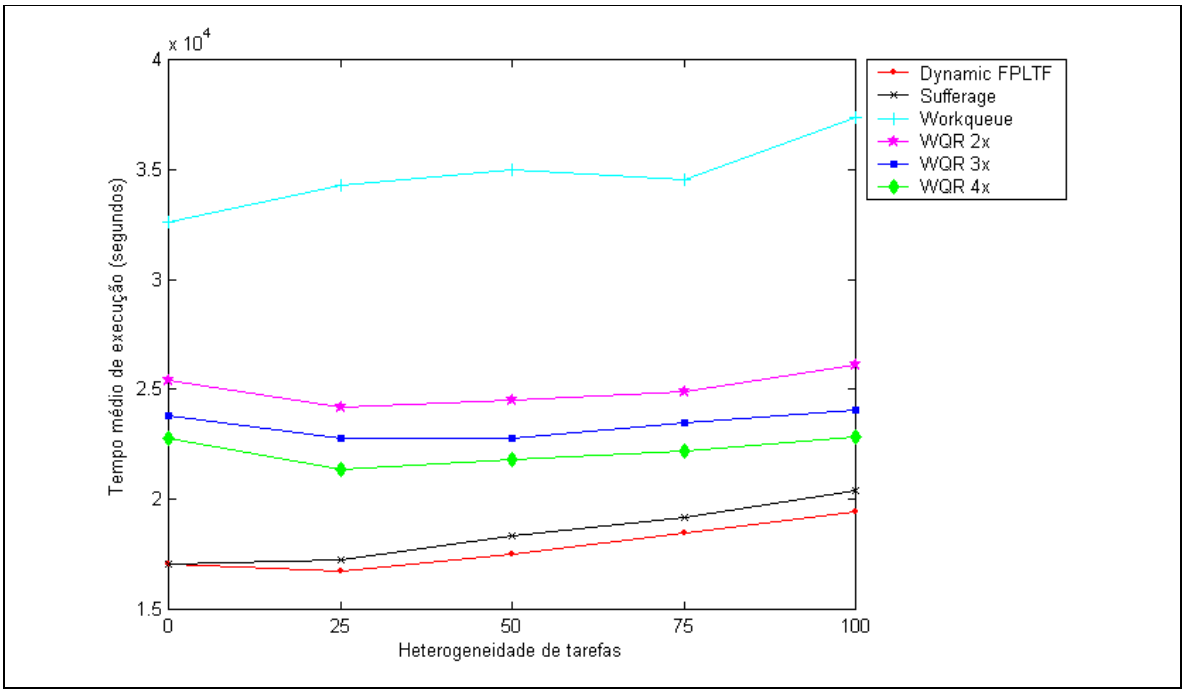

Figura 12. Desempenho por heterogeneidade de tarefas das aplicações com tamanho médio de tarefas de 125.000 segundos.

Entendendo Melhor o Efeito da Granularidade das Aplicações

Um maior detalhamento do efeito da granularidade das aplicações sobre os algoritmos é feito nessa seção pelo fato dessa variável ter sido a que mais afetou os algoritmos. As Figuras 11 a 16 mostram o comportamento de cada algoritmo em relação à variação no tamanho médio das tarefas das aplicações, destacando sua performance de acordo com a heterogeneidade das máquinas no Grid.

O primeiro ponto a ser destacado em relação a esses resultados é que para as aplicações com tarefas grandes (média de 125.000 segundos), os algoritmos que lidam com informações do ambiente e da aplicação (Sufferage e Dynamic FPLTF) obtiveram melhores performances. De fato essa situação fica mais evidente quando a heterogeneidade de máquinas no Grid é muito alta (nível 16). Com uma maior granularidade da aplicação, o final da execução (momento em que todas as tarefas já foram escalonadas) chega de forma mais rápida e representa uma fra-

ção maior do tempo de execução. Neste caso, a aplicação já está em “final de execução” desde o início. O Workqueue (Figura 15) e o WQR (Figuras 14 a 16) não têm mecanismos para impedir que uma tarefa grande seja alocada para uma máquina lenta nesse momento e eleve consideravelmente seus tempos de execução. Entretanto, o WQR tem uma chance de evitar esse problema, pois uma réplica de uma tarefa alocada para uma máquina lenta pode ser alocada para uma máquina mais rápida, diminuindo o impacto no tempo de execução. O Dynamic FPLTF (Figura 13) e o Sufferage (Figura 14) conseguem amenizar o impacto causado por este problema pelo fato de terem informações sobre o comportamento das máquinas no Grid. Assim, estes algoritmos podem evitar que uma tarefa seja alocada para uma máquina muito lenta.

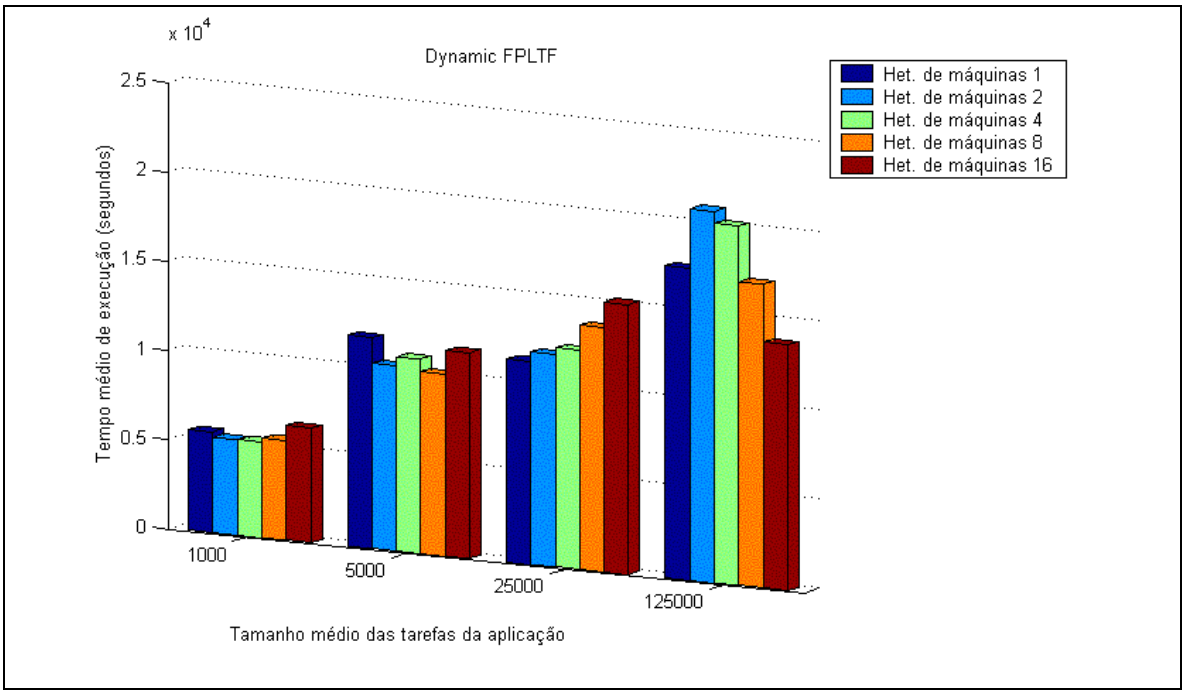

Figura 13. Performance alcançada pelo Dynamic FPLTF em cada tipo de aplicação.

Um fato interessante que deve ser notado no comportamento do Dynamic FPLTF e Sufferage é a melhor performance ocorrida no nível de heterogeneidade de máquinas 16 de aplicações com tamanho médio de tarefas de 125.000 segundos em relação às de 25.000 segundos. Ao rodar aplicações com granularidade alta (125.000 de média nos tamanhos das tarefas), passam a existir mais máqui-

nas no Grid do que tarefas a serem executadas. Assim, esses algoritmos conseguem evitar que as tarefas sejam executadas em máquinas lentas, atingindo uma excelente performance.

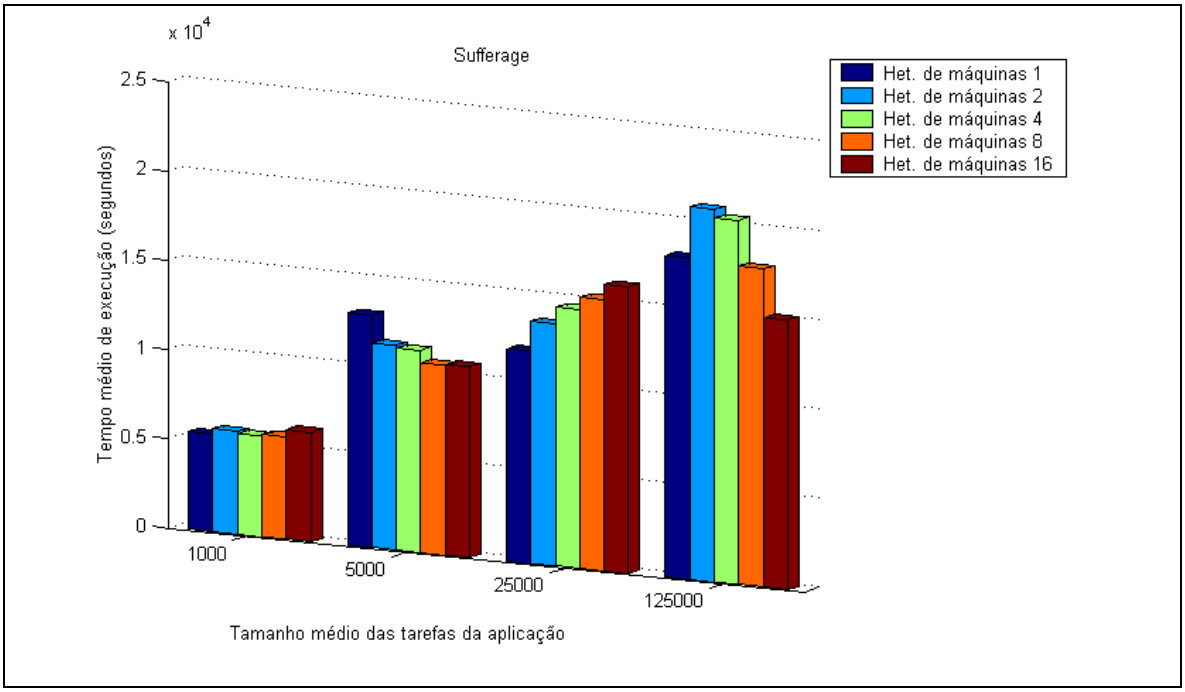

Figura 14. Performance alcançada pelo Sufferage em cada tipo de aplicação.

O segundo ponto que merece destaque a respeito desses gráficos é a boa performance do Workqueue para aplicações com baixa granularidade, mesmo tendo demonstrado um comportamento bastante irregular nos experimentos, de maneira geral. O Workqueue (Figura 15) atingiu uma performance praticamente igual à do Dynamic FPLTF (Figura 13) e do Sufferage (Figura 14) nas aplicações com tarefas de tamanho médio de 1.000 e 5.000 segundos. Com tarefas maiores, a performance do Workqueue é bastante degradada e ele não consegue acompanhar a performance dos outros algoritmos. O Workqueue não possui nenhum mecanismo que evite o escalonamento de uma tarefa grande para uma máquina lenta no fim do plano de escalonamento ou que amenize os efeitos dessa situação, por isso ocorre essa queda em seu rendimento.

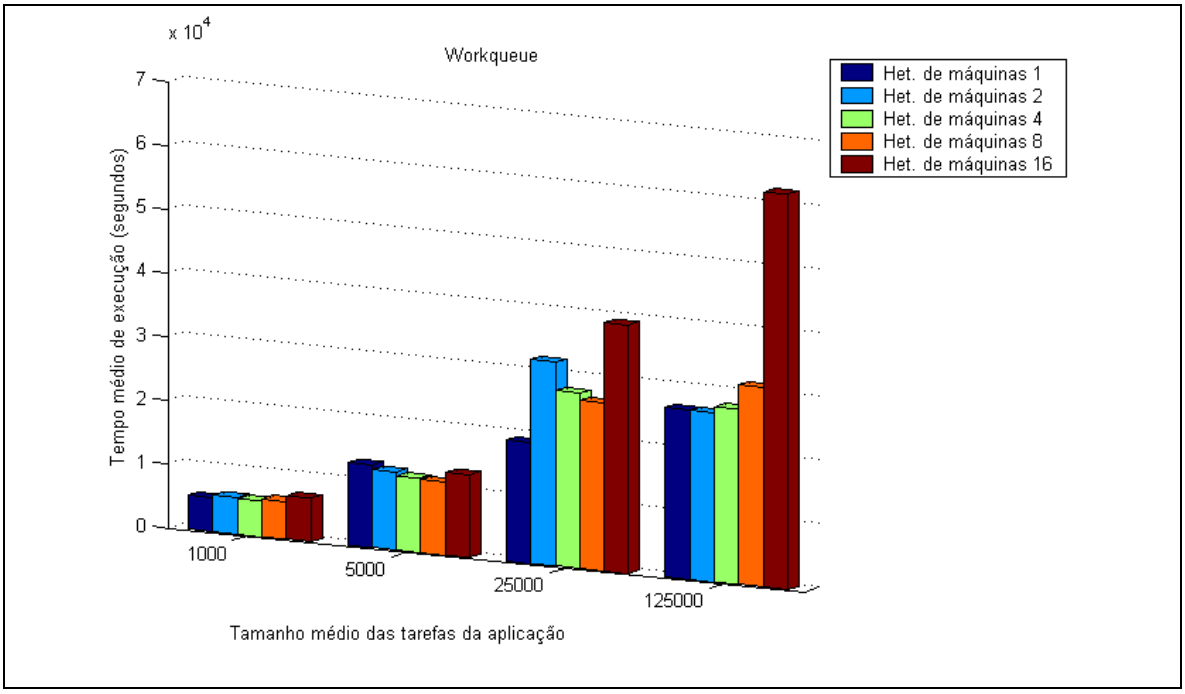

Figura 15. Performance alcançada pelo Workqueue em cada tipo de aplicação.

A respeito das três versões do WQR, o que vale ser ressaltado é a regularidade em seus resultados. O ponto bastante positivo dessa solução é que a heterogeneidade do Grid praticamente não tem influência no tempo final de execução das aplicações como pode ser notado nos gráficos seguintes (Figuras 14 a 16). A exceção a esse comportamento é quando a heterogeneidade de máquinas é 16 nas aplicações com tamanho médio de tarefas igual a 125.000 segundos. Nesse caso, quando uma tarefa grande e sua(s) réplica(s) são alocadas para máquinas excessivamente lentas/carregadas, o tempo final de execução da aplicação pode ser bastante elevado. O que pode ser percebido é que, ao aumentar o nível de replicação, essa “anormalidade” causada pela grande heterogeneidade de máquinas no Grid é consideravelmente reduzida.

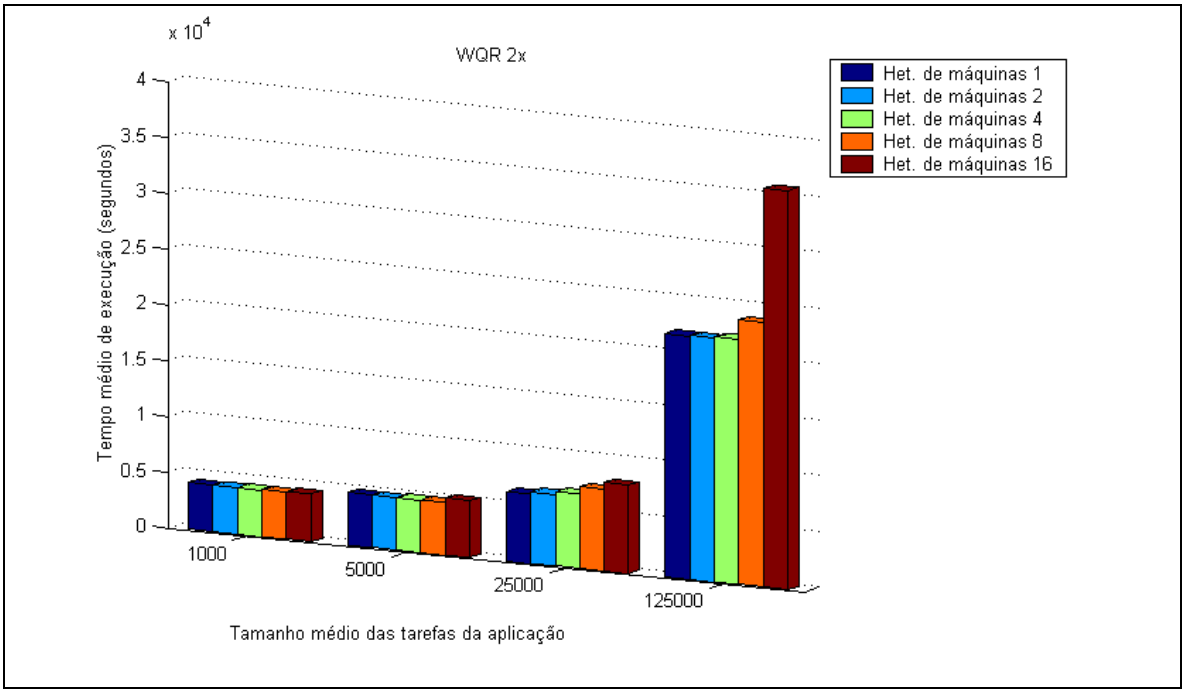

Figura 16. Performance alcançada pelo WQR 2x em cada tipo de aplicação.

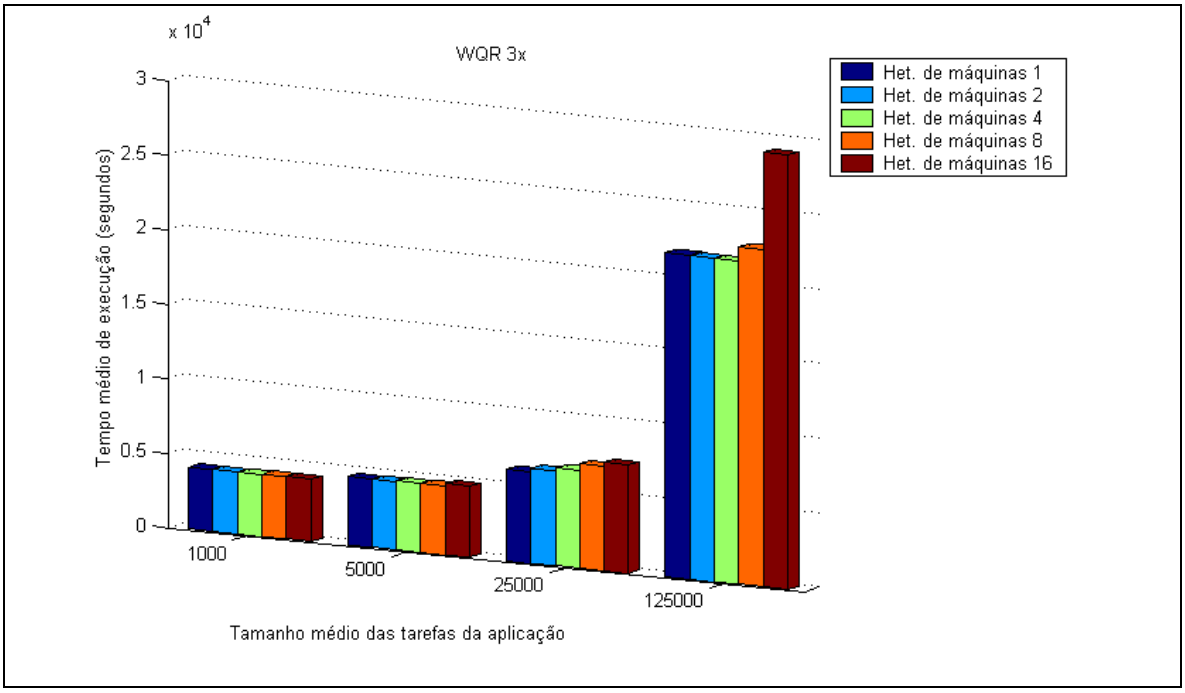

Figura 17. Performance alcançada pelo WQR 3x em cada tipo de aplicação.

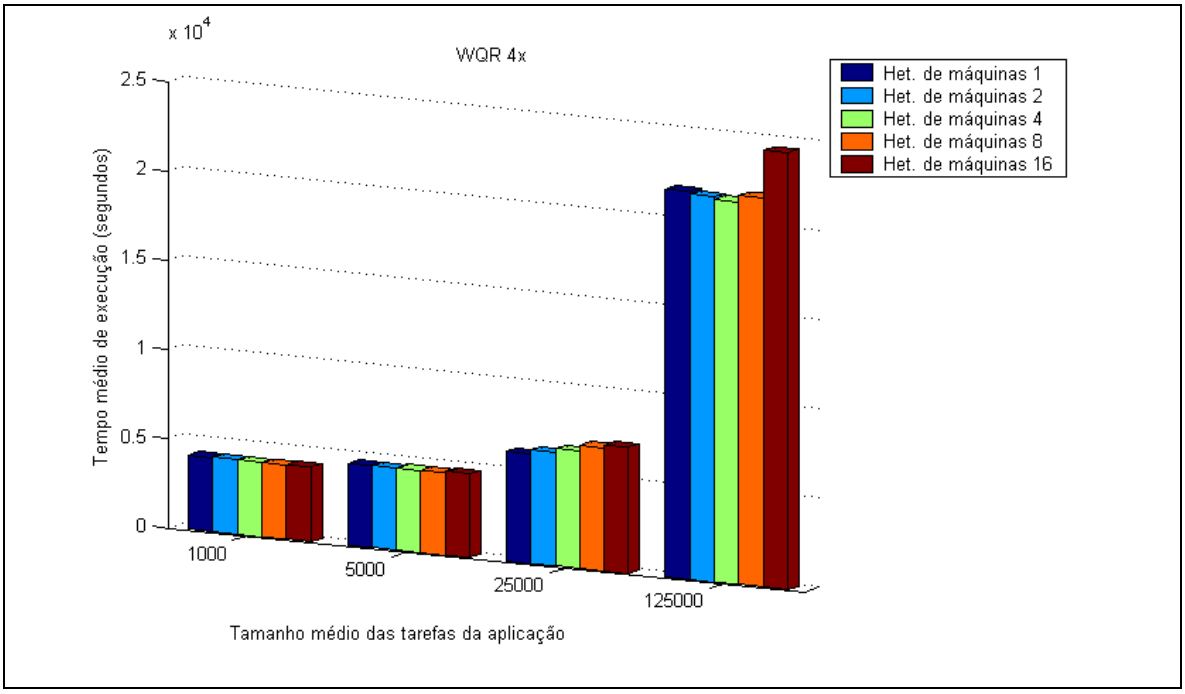

Figura 18. Performance alcançada pelo WQR 4x em cada tipo de aplicação.

Desvio Padrão

A Figura 19 mostra o desvio padrão dos algoritmos agrupados pela granularidade das aplicações. Ela evidencia que o WQR é uma excelente solução para o escalonamento de tarefas em ambientes heterogêneos. O gráfico mostra o desvio padrão dos tempos de execução de cada algoritmo agrupados por tipo de aplicação. Os dados referentes ao Workqueue foram excluídos dos gráficos dessa Subseção, pois seus valores eram muito altos (atingindo até 60.000), dificultando a visualização dos mesmos. O fato de o Workqueue ter valores muito altos de desvio padrão caracteriza a sua irregularidade. Em certos momentos consegue bons resultados e em outros momentos resultados catastróficos.

Com um desvio padrão consideravelmente baixo, principalmente com as aplicações com tarefas menores (1.000, 5.000 e 25.000 segundos), o WQR mostra ser um algoritmo bastante confiável. Não há uma variação muito grande nos tempos de execução das aplicações. Com o Dynamic FPLTF e Sufferage isso não acontece, há uma maior variação nos tempos de execução das aplicações o que

pode trazer uma certa insegurança aos usuários. Em certos momentos esses algoritmos podem ter uma boa performance e em outros se comportarem mal.

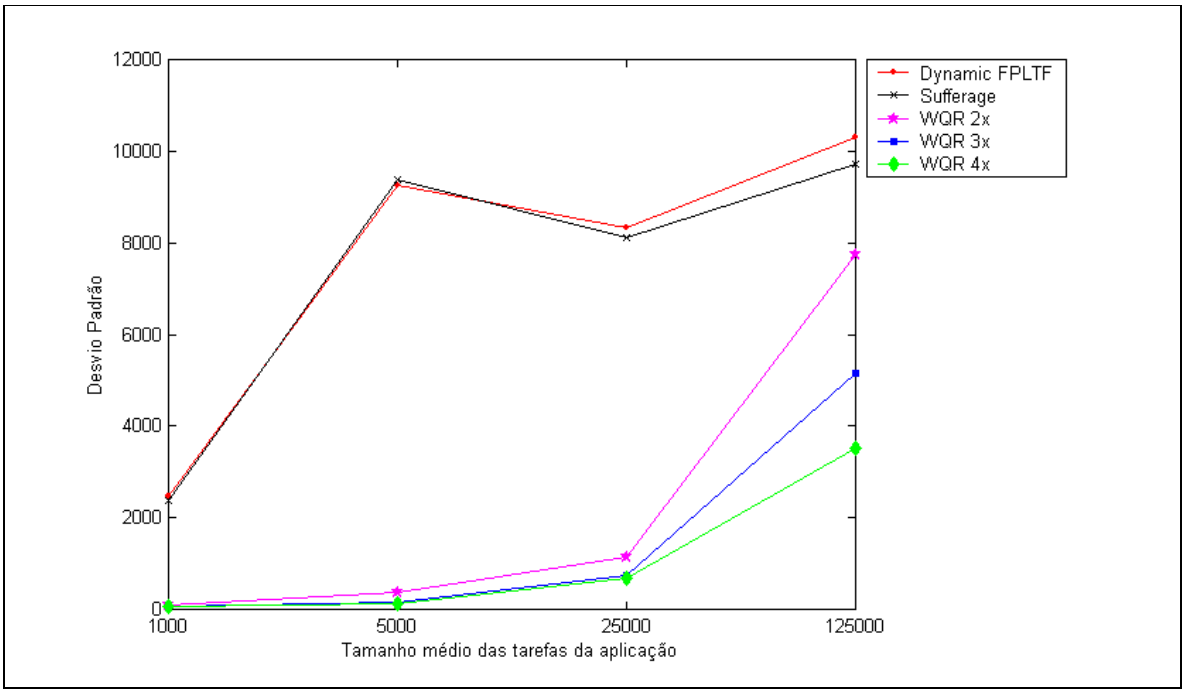

Figura 19. Desvio Padrão dos tempos de execução agrupados pela granularidade das aplicações.

A Figura 20 mostra o desvio padrão dos algoritmos agrupados pelo nível de heterogeneidade das máquinas do Grid. Quando o Grid é homogêneo (nível de heterogeneidade 1), os algoritmos se comportam de forma muito parecida em relação ao desvio padrão. Neste cenário não existe diferença entre as velocidades das máquinas, os tempos de execução dos algoritmos tendem a variar menos. Quando o Grid não é muito heterogêneo (níveis 2, 4 e 8), o WQR consegue obter melhores valores de desvio padrão que o Dynamic FPLTF e o Sufferage. Note que o fato do Dynamic FPLTF e do Sufferage piorarem seus desvios padrões ao passar de Grids homogêneos para heterogêneos é um comportamento absolutamente normal. Com máquinas diferentes a tendência é ter uma variação maior nos tempos de execução.

Um fato interessante ocorrido é que o WQR consegue manter seu desvio padrão praticamente constante até o nível 8 devido a sua estratégia de replicação. Até esse nível, a estratégia de replicação funciona bem na tentativa de replicar

uma tarefa para uma máquina rápida quando acontece dela ser escalonada para uma máquina lenta. Mesmo quando a tarefa é replicada para uma máquina lenta também, o prejuízo no tempo de execução é pequeno porque a diferença entre a máquina mais lenta e mais rápida não é muito grande.

Entretanto, no nível de heterogeneidade 16 a situação é diferente. A diferença entre a máquina mais poderosa do Grid e a mais fraca pode ser consideravelmente grande. A pior máquina desse nível (velocidade relativa 2) pode ser três vezes mais lenta que a pior do nível 8 (velocidade 6). A alocação de tarefas para máquinas lentas no nível 16 e também de suas réplicas podem causar maiores variações nos tempos de execução como visto na Figura 20. Ao contrário do que acontece com o WQR, o Dynamic FPLTF e o Sufferage melhoram seus desvios padrões no nível 16. Isso ocorre por causa da maior prioridade que esses escalonadores dão às máquinas mais rápidas. Anteriormente nessa seção, pôde ser visto que esses algoritmos alcançam melhores resultados que o WQR nesse nível de heterogeneidade e a tendência é que, com resultados melhores, a variação entre eles também seja menor.

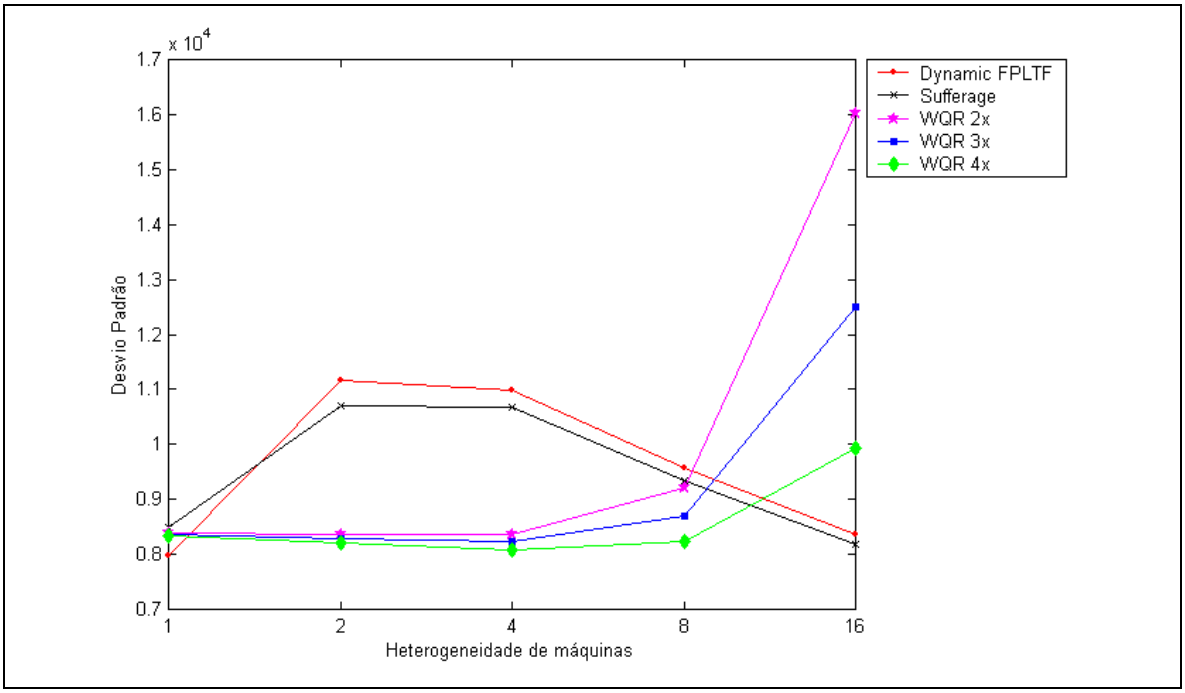

Figura 20. Desvio Padrão dos tempos de execução agrupados pelo nível de heterogeneidade de máquinas.

Ao agrupar o desvio padrão dos algoritmos pela heterogeneidade de tarefas como mostrado na Figura 21, nota-se que a influência da heterogeneidade na variabilidade dos resultados é muito pequena. Talvez isso fosse diferente para Sufferage e Dynamic FPLTF se a informação fornecida a eles não fosse perfeita. A tendência geral é que os resultados variem mais com uma maior heterogeneidade, mas essa variação é praticamente insignificante. Em relação às tarefas, o que realmente importa para os escalonadores é o tamanho do grão como foi visto na Figura 19.

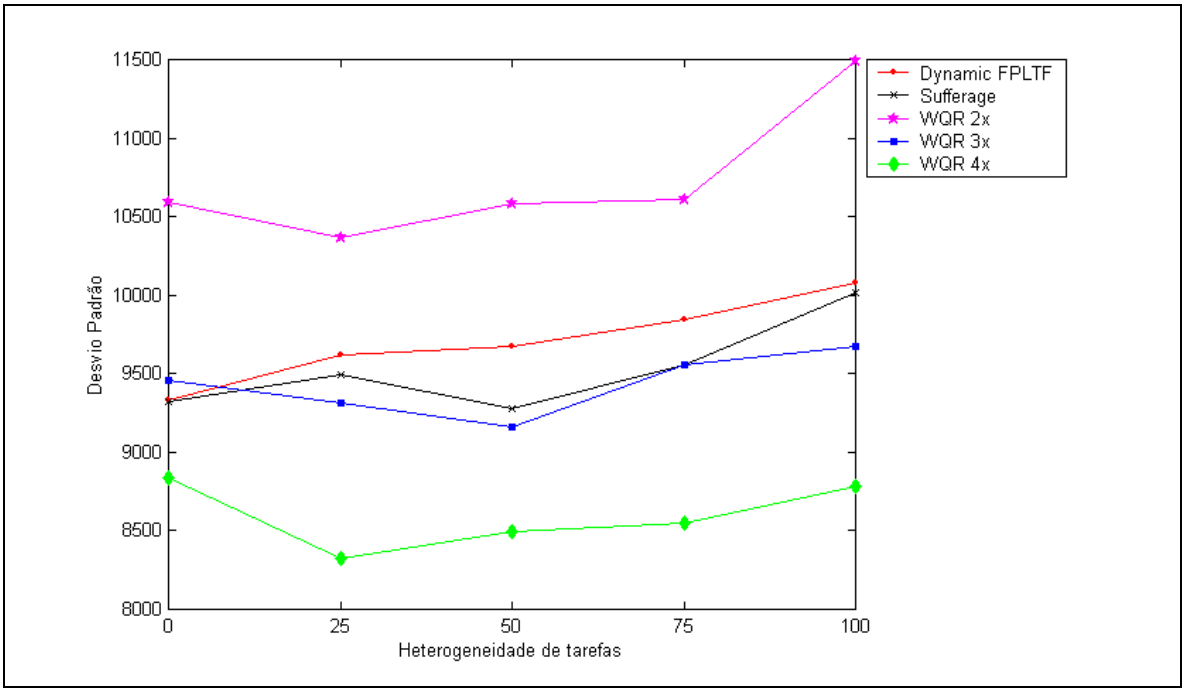

Figura 21. Desvio Padrão dos tempos de execução agrupados pelo nível de heterogeneidade de tarefas.

Desperdício de Ciclos de CPU

Para alcançar o ótimo desempenho demonstrado ao longo desse trabalho, o WQR sacrifica alguns ciclos de CPU como é mostrado na Figura 22. Neste gráfico os dados estão agrupados pela granularidade da aplicação. O valor percentual de ciclos de CPU gastos é obtido através da divisão da quantidade de ciclos gastos com réplicas que foram canceladas pelo total de ciclos.

O desperdício de ciclos de CPU é consideravelmente pequeno nas aplicações cujos tamanhos médios das tarefas são 1.000 e 5.000 segundos, com valores abaixo de 7%. Nas aplicações com tarefas de 25.000 segundos, o percentual de ciclos desperdiçados não chega a 35% com o WQR 4x. Pelo fato do WQR 2x atingir resultados muito próximos do WQR 4x nessa categoria de aplicação (Figura 7), é possível limitar a replicação em 2x e, ainda assim, obter excelentes resultados. Dessa forma, o percentual de ciclos desperdiçados passa a ser de aproximadamente 20%. Para as aplicações onde as tarefas possuem tamanho médio igual a 125.000 segundos, a situação é um pouco diferente. A diferença entre o WQR 2x e o WQR 4x é um pouco maior, inclusive o WQR 2x e 3x são piores que o Dynamic FPLTF e o Sufferage, nesse caso (Figura 7). Para obter resultados melhores que o Dynamic FPLTF, utilizando o WQR 4x, é necessário gastar um pouco mais que o dobro da CPU que seria utilizada para executar a aplicação.

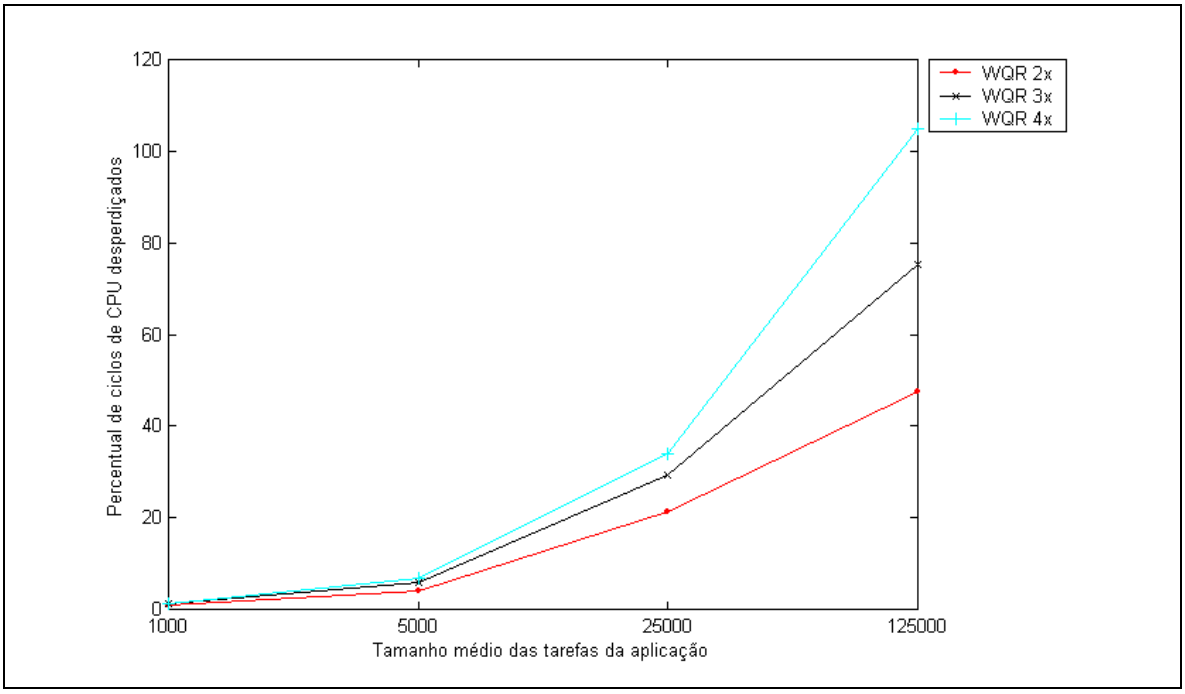

Figura 22. Percentual de ciclos desperdiçados com a replicação por tipo de aplicação.

A granularidade maior da aplicação faz com que o final da execução (momento onde não há mais tarefas para serem escalonadas e início da replicação) represente uma fração maior no tempo de execução. No caso 125.000 segundos

representa todo o tempo da execução. Desse modo, réplicas realizadas em máquinas lentas têm um impacto maior no desperdício total.

A Figura 23 e a Figura 24 mostram a influência da heterogeneidade das máquinas e das tarefas, respectivamente, sobre o desperdício de ciclos de CPU dos algoritmos. O que pode ser notado em ambas as figuras é que o aumento da heterogeneidade eleva o desperdício dos algoritmos. Esse fato ocorre porque maior heterogeneidade no Grid e na aplicação torna o escalonamento mais difícil como já foi observado anteriormente.

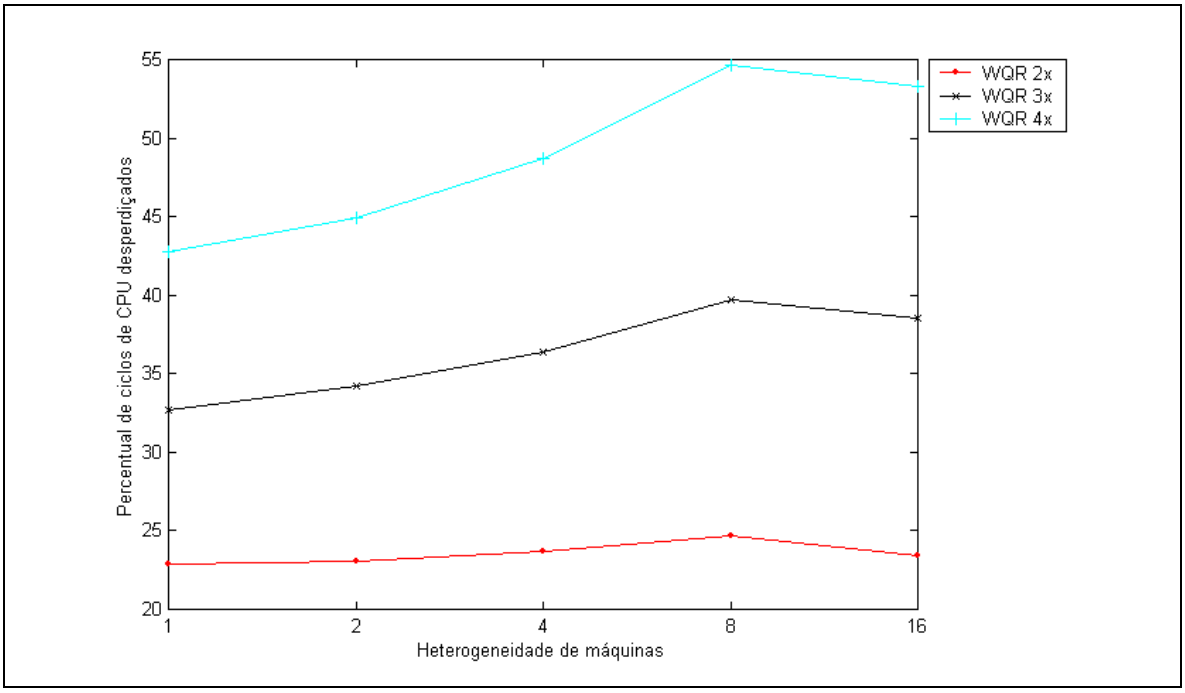

Figura 23. Percentual de ciclos desperdiçados com a replicação pelo nível de heterogeneidade de máquinas.

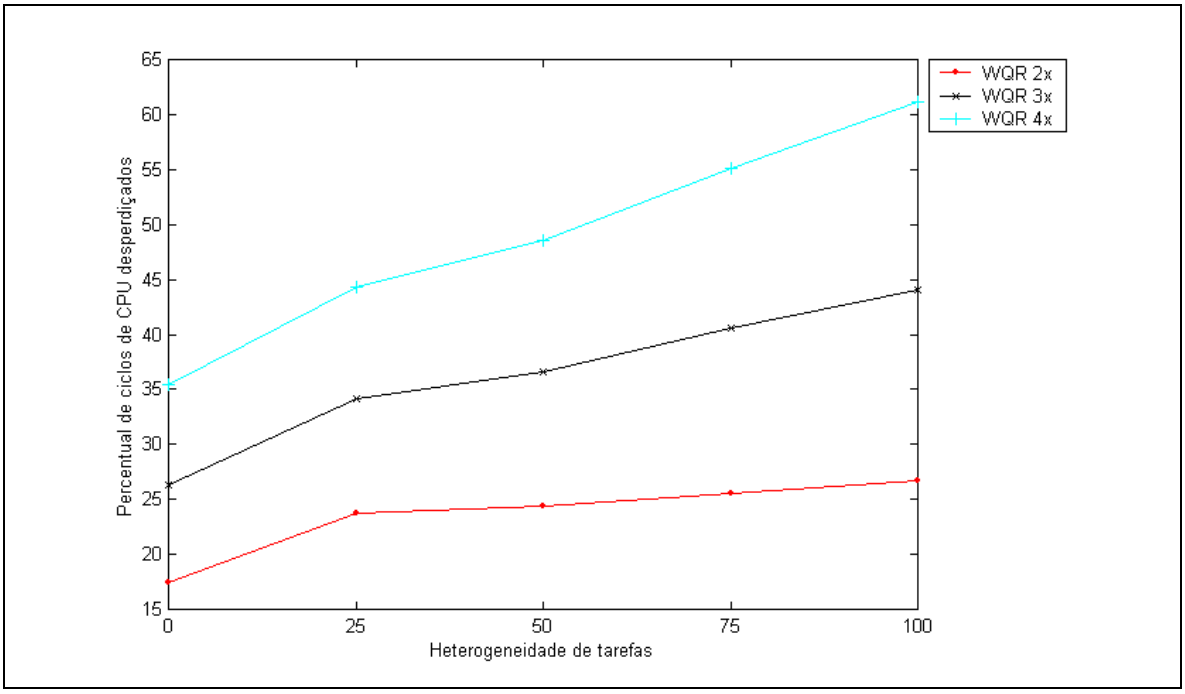

Figura 24. Percentual de ciclos desperdiçados com a replicação pelo nível de heterogeneidade de tarefas.

Sumário dos Resultados

A Tabela 6 apresenta os resultados gerais dos algoritmos sumariados pela granularidade das aplicações. O que vale ser ressaltado aqui é o ótimo desempenho do WQR nas aplicações com granularidade 1000, 5000 e 25000 segundos, gastando menos de 35% ciclos de CPU extras nessa última. Quando a granularidade é maior pode se perceber que o rendimento do WQR não é tão bom porque o final da execução representa toda a aplicação e a alocação de uma tarefa para uma máquina muito lenta pode prejudicar bastante a execução da aplicação. Os algoritmos Dynamic FPLTF e Sufferage têm uma melhor performance nesses casos (assumindo que eles têm informação perfeita).

	Granularidade 1000	Granularidade 5000	Granularidade 25000	Granularidade 125000
Média Dynamic FPLTF	5674.94	10954.28	12883.95	17835.73
Média Sufferage	5747.73	11428.85	14257.36	18425.57
Média Workqueue	5898.56	12271.49	28792.39	34753.25
Média WQR 2x	4202.75	4796.48	6925.78	25019.80
Média WQR 3x	4176.21	4644.60	6681.39	23381.91
Média WQR 4x	4171.09	4622.23	6614.17	22198.30
Desperdício WQR 2x	0.81%	3.97%	21.20%	47.35%
Desperdício WQR 3x	1.16%	5.74%	29.18%	75.26%
Desperdício WQR 4x	1.37%	6.82%	33.86%	104.91%

Tabela 6. Sumário dos resultados obtidos pelos escalonadores.

5.3 Validação dos Resultados

Para validar os resultados obtidos com os experimentos, o algoritmo WQR foi implementado no MyGrid [12][35][39] com o intuito de avaliá-lo em um ambiente real. Para efeito de comparação foi utilizado o Workqueue que era o algoritmo utilizado pelo MyGrid anteriormente.

A quantidade média de máquinas utilizadas nas simulações era 100. Não foi possível conseguir esse número de máquinas para executar os experimentos em um ambiente real. Apenas 35 máquinas ficavam disponíveis, em média, foram utilizadas em cada experimento. Assim sendo, o número de tarefas foi reduzido proporcionalmente em uma tentativa de aproximar o experimento simulado do prático. O ambiente de máquinas era composto por 3 *clusters* (2 *clusters* Linux e 1 *cluster* Solaris) descritos na Tabela 7. As máquinas utilizadas eram compartilhadas, enquanto algumas máquinas tornavam-se disponíveis, outras passavam a ficar inacessíveis. Usuários frequentemente reiniciam suas máquinas, tornando-as inacessíveis por um certo tempo e matando os processos que nela estavam sendo executados. Existe também a possibilidade de alguns usuários aplicarem certas restri-

ções em suas máquinas sobre os processos que podem ser executados por outros usuários, possivelmente tornando essas máquinas inutilizáveis ao Grid. Esse fato de máquinas poderem “aparecer” e “desaparecer” do Grid não acontecia nas simulações. Certamente, essa característica presente no ambiente real tem um impacto na performance dos algoritmos, já que uma tarefa que estava sendo executada em uma máquina que saiu do Grid deverá ser re-executada em outra máquina.

Máquinas	Processador	Velocidade (MHz)	Memória (MB)
Cluster Linux Local na Universidade Federal de Campina Grande			
14	Intel(R) Pentium(R) 4	1800	640
1	Intel(R) Pentium(R) 4	1500	512
1	Intel(R) Pentium(R) 4	1500	256
1	AMD Athlon(TM) XP 1700+	1466	512
2	Pentium III (Coppermine)	800	192
Cluster Linux Remoto na University of California San Diego			
7	AMD Athlon(tm) XP 2100+	1733	512
1	AMD Athlon(tm) Processor	700	256
1	Pentium II (Deschutes)	400	512
Cluster Solaris Remoto na University of California San Diego			
1	Sun UltraSPARC-IIi	440	256
9	Sun UltraSPARC-IIi	333	128
3	Sun UltraSPARC-IIi	300	128
1	Model 61 SuperSPARC	60	96

Tabela 7. Máquinas utilizadas no experimento em um ambiente real.

A aplicação utilizada nos experimentos era composta por 250 tarefas para ser proporcional às 720 tarefas utilizadas nas simulações, já que o número de máquinas médio no ambiente real foi de 35 e no ambiente de simulação eram 100 máquinas. A tarefa é simplesmente um *loop* escrito em C (*CPU-bound*), onde o seu tamanho, definido através de um parâmetro, segue o mesmo método utilizado para simular a heterogeneidade de tarefas nas simulações descrito na Seção 5.1.

A aplicação utilizada para realizar essa análise comparativa tinha tamanho médio das tarefas é igual a 5000 segundos. Note que, em um ambiente real, é difícil definir o tempo de execução de uma tarefa com a mesma precisão que um modelo de simulação permite. Outro ponto difícil também de ser reproduzido como nos experimentos simulados é a característica das máquinas. O ambiente de simulação permite que a definição do poder e comportamento de cada máquina sejam

bem controlados. No mundo real, isso é praticamente impossível de ser atingido, já que a intenção é executar a aplicação em um ambiente de produção.

Com tanta diferença do ambiente simulado para o real, os resultados obtidos ao executar os algoritmos foram um pouco diferentes dos obtidos nas simulações, mas ainda assim comprovam a superioridade do WQR. A Figura 25 mostra o resultado de 25 execuções de cada algoritmo, feitas *back-to-back* (uma vez o WQR, outra o Workqueue) para minimizar as diferenças do ambiente real que é difícil de ser controlado. É possível que houvesse uma diferença considerável no ambiente se os experimentos de cada algoritmo fossem realizados em sequência.

Como nos resultados das simulações apresentado na Seção 5.2, o WQR obteve uma performance bastante superior ao Workqueue, mostrando também mostrou uma menor variação nos seus resultados. Note que ocorreu uma diferença de performance dos algoritmos. A diferença entre o WQR e o Workqueue, aproximadamente 100% nas simulações, caiu para pouco menos que 50% nos experimentos em ambiente real. Essa diferença de performance pode ser creditada à grande dificuldade de reproduzir o cenário das simulações no ambiente real e também ao fato das máquinas poderem “aparecer” e “desaparecer” do Grid.

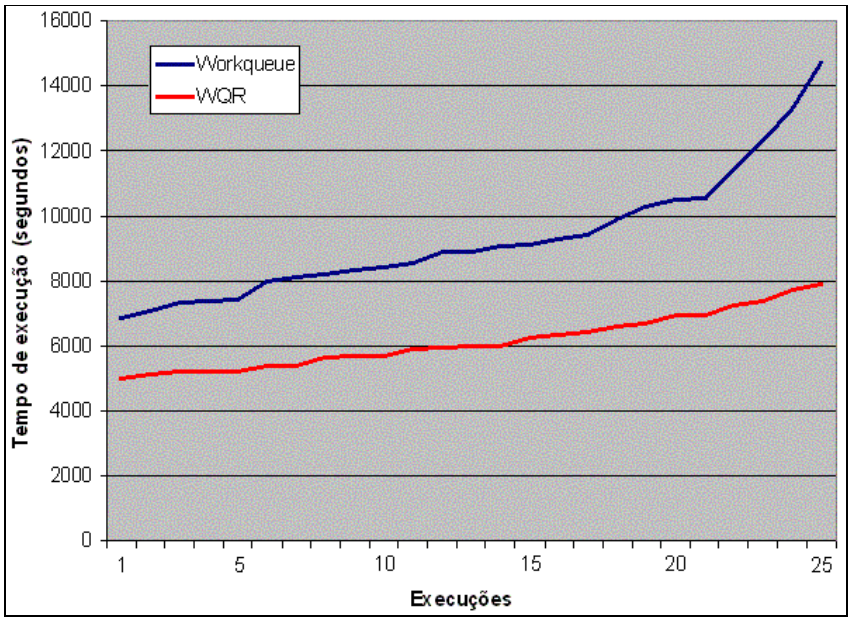

Figura 25. Comparação do WQR com o Workqueue em um ambiente real.

6 Conclusão

Neste trabalho foi proposto um algoritmo de escalonamento de tarefas, *Workqueue com Replicação* (WQR), para Grids Computacionais. O WQR visa as aplicações *Bag-of-Tasks* que são compostas por tarefas independentes e não necessitam de qualquer tipo de comunicação entre as tarefas. O funcionamento do WQR é parecido com o Workqueue clássico, mas no momento em que as máquinas tornar-se-iam ociosas, essas máquinas passam a serem utilizadas para computar réplicas de tarefas que ainda estão sendo executadas. O objetivo dessa estratégia é que, em um conjunto de réplicas de uma tarefa, a primeira réplica completada é considerada como a execução normal da tarefa em questão. As réplicas restantes são canceladas para tornar as máquinas disponíveis para a execução de outras tarefas. Obviamente, WQR assume que as tarefas são idempotentes, uma hipótese bem razoável em Grids.

O WQR não se baseia em nenhum tipo de informação sobre a performance das máquinas ou sobre as características da aplicação para realizar o escalonamento. Entretanto, ao contrário de outras abordagens que também não utilizam essas informações, consegue atingir uma boa performance. Essa boa performance é alcançada com um aumento no consumo de recursos para compensar o fato de que o WQR não baseia seu plano de escalonamento em informações sobre as máquinas e sobre as tarefas. Vale ser ressaltado que esse consumo extra de recursos pode ser controlado através da limitação do número de réplicas, uma solução que tem pouco impacto na performance do WQR.

O WQR atingiu ótimo desempenho, na maioria das situações, melhores do que escalonadores que usam informações sobre o ambiente e sobre a aplicação (Dynamic FPLTF e Sufferage) para elaborar o plano de escalonamento. O WQR só não teve uma boa performance em aplicações com alta granularidade (relação

máquinas por tarefa maior que 1). Com uma granularidade tão alta, o final da execução (momento onde não há mais tarefas para serem escalonadas, mas existem tarefas sendo executadas) representa todo o tempo de execução da aplicação. Desse modo, a alocação de uma tarefa para uma máquina lenta neste momento tem um impacto maior no tempo final de execução da aplicação. Nesses casos, o WQR já inicia em modo de replicação e a possível alocação de uma tarefa e sua(s) réplica(s) para máquinas lentas pode gerar resultados ruins.

O WQR foi implementado como parte do projeto MyGrid [12][35][39] desenvolvido na Universidade Federal de Campina Grande. Espera-se que esse esforço permita que usuários executem suas aplicações de forma mais eficiente em Grids.

Trabalhos Futuros

Essa Seção apresenta propostas de possíveis trabalhos futuros para dar continuidade aos estudos realizados nesse trabalho. As propostas são baseadas em melhorias no modelo Grid utilizado e características das aplicações.

- Modelagem de transferências de dados presente em aplicações com entrada e saída intensivas que não “arrumam” seus dados no Grid previamente.
- Implementação de um modelo de falhas necessário para simular problemas nas máquinas e provocar a necessidade de executar novamente tarefas que foram alocadas para máquinas que falharam.
- Realização de um estudo do efeito causado por múltiplas instâncias do WQR no mesmo Grid. Obviamente, o consumo extra de recursos implica em uma maior carga imposta ao Grid. Por outro lado, aplicações podem ser completadas de forma mais rápida deixando o Grid menos carregado para a execução de aplicações que apareçam no futuro.
- Implementação de um modelo onde se tem acesso a informações parciais do Grid (e.g. máquinas do domínio administrativo do usuário). Esse tipo de informação deve melhorar ainda mais o desempenho do WQR.

- Investigação do WQR quando a replicação é necessária para se defender de usuários mal intencionados (como em SETI@Home [3]).

Referências

[1] I. Ahmad. *Editorial: Resource Management in Parallel and Distributed Systems with Dynamic Scheduling: Dynamic Scheduling.* Concurrency: Practice and Experience, Vol. 7: 587-590, Outubro, 1995.

[2] A. Alhusaini, V. Prasanna, e C. Raghavendra. *A Unified Resource Scheduling Framework for Heterogeneous Computing Environments*. Proc. 8th IEEE Heterogeneous Computing Workshop. Abril, 1999.

[3] D. Anderson, J. Cobb e E. Korpela. *SETI@home: An Experiment in Public-Resource Computing*. Em Communications of the ACM, Vol. 45 No. 11, páginas 56-61. Novembro, 2002.

[4] F. Berman, R. Wolski e S. Figueira *et al*. *Application Level Scheduling on Distributed Heterogeneous Networks*. Em Proc. Of Supercomputing'96, Pittsburgh, 1996.

[5] F. Berman, R. Wolski e G. Shao. *Performance Effects of Scheduling Strategies of Master/Slave Distributed Applications*. Relatório Técnico TR-CS98-598, U. C., San Diego, 1998.

[6] F. Berman e R. Wolski. *The AppLeS Project: A Status Report*. Em Proc. of the 8th NEC Research Sysmposium, Berlin, Germany, Maio, 1997.

[7] *Biology Workbench Portal*. Disponível em http://workbench.sdsc.edu/. Acessado em Fevereiro de 2003.

[8] *Cactus, software de código aberto para a criação de portais Grid*. Disponível em http://www.cactuscode.org/. Acessado em Fevereiro de 2003.

[9] H. Casanova, A. Legrand and D. Zagorodnov *et al*. *Heuristics for Scheduling Parameter Sweep Applications in Grid Environments*. Heterogeneous Computing Workshop: 349-363. 2000.

[10] H. Casanova, A. Legrand and D. Zagorodnov *et al*. *Using Simulation to Evaluate Scheduling Heuristics for a Class of Applications in Grid Environments*. Ecole Normale Superieure de Lyon, RR1999-46. 1999.

[11] *Chemical Engineering Workbench Portal*. Disponível em http://archive.ncsa.uiuc.edu/alliance/partners/ApplicationTechnologies/ChemicalEngineering.html. Acessado em Fevereiro de 2003.

[12] W. Cirne, D. Paranhos and F. Brasileiro *et al*. *Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach*. No Proceedings of the International Conference on Parallel Processing (ICPP-03). Outubro, 2003.

[13] S. Davidson, H. Garcia-Molina e D. Skeen. *Consistency in Partitioned Networks*. ACM Computing Surveys, páginas 341-370, Setembro, 1985.

[14] R. Dragan. *The Meaning of Moore's Law*. Disponível em http://www.pcmag.com/article2/0,4149,4092,00.asp. Acessado em Fevereiro de 2003.

[15] W. Elwasif, J. Plank and R. Wolski. *Data Staging Effects in Wide Area Task Farming Applications*. IEEE International Symposium on Cluster Computing and the Grid, Brisbane, Austrália, Maio, 2001, pp. 122-129.

[16] *Entropia: PC Grid Computing*. Disponível em http://www.entropia.com/. Acessado em Fevereiro de 2003.

[17] *European Union Data Grid*. Disponível em http://www.eu-datagrid.org/. Acessado em Fevereiro de 2003.

[18] G. Fishman. *Monte Carlo: Concepts, Algorithms, and Applications.* Springer-Verlag New York, Incorporated. Novembro, 1995.

[19] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann. 1998.

[20] P. Francis, S. Jamin and V. Paxson *et al*. *An Architecture for a Global Internet Host Distance Estimation Service*. Proceedings of IEEE INFOCOM, 1999.

[21] R. Freund, T. Kidd e D. Hensgen *et al*. *SmartNet: A Scheduling Framework for Heterogeneous Computing*. No Proceedings of the International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN-96), páginas 514-521, 1996.

[22] R. Freund, M. Gherrity e S. Ambrosius *et al*. *Scheduling Resources in Multi-User, Heterogeneous, Computing Environments with SmartNet*. 7th IEEE Heterogeneous Computing Workshop (HCW '98), páginas 184-199. Março, 1998.

[23] *Globus*. Disponível em http://www.globus.org. Acessado em Fevereiro de 2003.

[24] *Grid Portal Development Kit*. Disponível em http://doesciencegrid.org/projects/GPDK/. Acessado em Fevereiro de 2003.

[25] *GriPhyN, the Grid Physics Network*. Disponível em http://www.griphyn.org/. Acessado em Fevereiro de 2003.

[26] B. Hamidzadeh, D. Lilja e Y. Atif. *Dynamic Scheduling Techniques for Heterogeneous Computing Systems.* Concurrency: Practice and Experience, 7, páginas 633-652. 1995.

[27] *International Virtual Data Grid Laboratory*. Disponível em http://www.ivdgl.org/. Acessado em Fevereiro de 2003.

[28] M. Iverson e F. Zgner. *Dynamic*, *Competitive Scheduling of Multiple DAGs in a Distributed Heterogeneous Environment*. No Seventh Heterogeneous Computing Workshop. IEEE Computer Society Press, 1998.

[29] H. James, K. Hawick and P. Coddington. *Scheduling Independent Tasks on Metacomputing Systems*. The University of Adelaide. DHPC-066, 1999.

[30] M. Litzkow, M. Livny, e M. Mutka. *Condor: A Hunter of Idle Workstations*. In Proceedings of the 8th International Conference of Distributed Computing Systems, páginas 104-111. Junho, 1988.

[31] B. Lowekamp, N. Miller and D. Sutherland *et al*. *A Resource Query Interface for Network-Aware Applications*. Seventh IEEE Symposium on High-Performance Distributed Computing, July 1998.

[32] M. Maheswaran, S. Ali and H. Siegel *et al*. *Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems*. Heterogeneous Computing Workshop, 1999.

[33] D. Menascé, D. Saha and S. Porto *et al*. *Static and Dynamic Processor Scheduling Disciplines in Heterogeneous Parallel Architectures*. Journal of Parallel and Distributed Computing, pp. 1-18. 1995.

[34] J. Moreira e W. Cirne. *Using Relative Performance to Gauge Parallel Super-computer Performance*. Proceedings of the Third Workshop on High Perfomance Computing Systems. Outubro, 2002.

[35] *MyGrid*. Disponível em http://www.dsc.ufpb.br/mygrid/. Acessado em Fevereiro de 2003.

[36] *Network for Earthquake Engineering Simulation*. Disponível em http://www.neesgrid.org/. Acessado em Fevereiro de 2003.

[37] *NPACI GridPort Toolkit*. Disponível em https://gridport.npaci.edu/. Acessado em Fevereiro de 2003.

[38] *Nug30 solution*. Disponível em http://www-unix.mcs.anl.gov/metaneos/nug30/solution.html. Acessado em Fevereiro de 2003.

[39] D. Paranhos, W. Cirne e F. Brasileiro. *Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids*. Proceedings of the Euro-Par 2003: International Conference on Parallel and Distributed Computing. Agosto 2003.

[40] *Particle Physics Data Grid*. Disponível em http://www.ppdg.net/. Acessado em Fevereiro de 2003.

[41] G. Perkins, C. Renken e S. Young *et al*. *Electron Tomography of Large Multi-component Biological Structures*. J, Struct. Biol., 120:219 227, 1997.

[42] *Perl.com: The Source for Perl*. Disponível em http://www.perl.com/. Acessado em Fevereiro de 2003.

[43] J. Plank, M. Beck and W. Elwasif *et al*. *The Internet Backplane Protocol: Storage in the Network*. Em NetStore '99: Network Storage Symposium. Outubro, 1999.

[44] *SETI@Home*. Disponível em http://setiathome.ssl.berkeley.edu/. Acessado em Fevereiro de 2003.

[45] *SimGrid*. Disponível em http://grail.sdsc.edu/projects/simgrid/. Acessado em Outubro, 2002.

[46] S. Smallen, W. Cirne and J. Frey *et al*. *Combining Workstations and Supercomputers to Support grid Applications: The Parallel Tomography Experience*. Em Proceedings of the HCW'2000 - Heterogeneous Computing Workshop. 2000.

[47] J. Smith, S. K. Shrivastava. *A System for Fault-Tolerant Execution of Data and Compute Intensive Programs over a Network of Workstations*. EuroPar'96, *Lecture Notes* em Computer Science vol. 1123, 1996.

[48] T. Tannenbaum, D. Wright, K. Miller et al. *Condor – a distributed job Scheduler*. Beowulf Cluster Computing with Linux. Cambridge, MA. 2001.

[49] T. Tannenbaum, D. Wright, K. Miller et al. *Condor – a distributed job scheduler*. Beowulf Cluster Computing with Windows. Cambridge, MA. 2001.

[50] *TeraGrid Project*. Disponível em http://www.teragrid.org/. Acessado em Fevereiro de 2003.

[51] *The CGI Resource Index*. Disponível em http://cgi.resourceindex.com/. Acessado em Fevereiro de 2003.

[52] *The History of nug30*. Disponível em http://www-unix.mcs.anl.gov/metaneos/nug30/history.html. Acessado em Fevereiro de 2003.

[53] R. Wolski. *Dynamically Forecasting Network Performance Using the Network Weather Service*. Cluster Computing, 1(1): 119-132, 1998.7