Cássio Henrique Volpatto Forte

Adaptação do Algoritmo OSEP de escalonamento para sistemas heterogêneos

Brasil

2015

Cássio Henrique Volpatto Forte

Adaptação do Algoritmo OSEP de escalonamento para sistemas heterogêneos

Monografia apresentada ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para aprovação na disciplina Projeto Final.

Universidade Estadual Paulista "Júlio de Mesquita Filho". Instituto de Biociências, Letras e Ciências Exatas.

Departamento de Ciências de Computação e Estatística

Orientador: Prof. Dr. Aleardo Manacero Jr.

Brasil

2015

Forte, Cássio Henrique Volpatto.
Adaptação do algoritmo OSEP de escalonamento para sistemas heterogêneos / Cássio Henrique Volpatto Forte. -- São José do Rio Preto, 2015
46 f. : il., gráfs.

Orientador: Aleardo Manacero Junior
Trabalho de conclusão de curso (bacharelado – Ciência da Computação) – Universidade Estadual Paulista "Júlio de Mesquita Filho", Instituto de Biociências, Letras e Ciências Exatas

1. Computação. 2. Computação em grade (Sistemas de computador) 3. Redes de computadores – Escalabilidade. 4. Algoritmos de computador. I. Manacero Junior, Aleardo. II. Universidade Estadual Paulista "Júlio de Mesquita Filho". Instituto de Biociências, Letras e Ciências Exatas. III. Título.

CDU – 681.3

Ficha catalográfica elaborada pela Biblioteca do IBILCE
UNESP - Câmpus de São José do Rio Preto

Cássio Henrique Volpatto Forte

Adaptação do Algoritmo OSEP de escalonamento para sistemas heterogêneos

Monografia apresentada ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para aprovação na disciplina Projeto Final.

Prof. Dr. Aleardo Manacero Jr. Cássio Henrique Volpatto Forte

Banca Avaliadora:

Prof.ª Dr.ª Renata Spolon Lobato

Prof.ª Dr.ª Rogéria Cristiane Gratão de Souza

Brasil
2015

Aos meus pais Manuel e Gislaine, e à minha irmã Amanda

Agradecimentos

Em primeiro lugar, quero agradecer a Deus, que na sua infinita bondade me deu a existência. Quero agradecer também à minha mãe, Gislaine, ao meu pai, Manuel, e à minha irmã Amanda, eles são as bases da minha vida, sem o amor deles eu não seria ninguém. Quero agradecer também aos meus orientadores Prof. Dr. Aleardo Manacero Jr. e Prof.ª Dr.ª Renata Spolon Lobato, que me guiaram com paciência, dedicação e profissionalismo sem jamais me deixar desamparado na trajetória acadêmica. Quero agradecer também à Reitoria da UNESP, que apoiou meus trabalhos com a bolsa PIBIC/Reitoria. Agradeço também ao meu amigo Denison Menezes, que usou seu tempo para me ajudar neste e em outros projetos com paciência e dedicação. Aos amigos Arthur Jorge, Diogo Tavares, Fernanda Peronaglio, Gabriel Saraiva, Gabriel Covello Furlanetto, João Antonio Magri Rodrigues, Leonardo Santos, Lucas Afonso Cazarote, Mario Camara Neto, Matheus A. S. Guissi, Renan Alboy, Victor Hugo Cândido e aos que passaram pelo laboratório, Danilo Costa, Leandro Moreira, Igor Butarello, Matheus Oliveira, Thiago Okada, Lúcio Rodrigo Carvalho.

"Possuímos em nós mesmos, pelo pensamento e a vontade, um poder de ação que se estende muito além dos limites de nossa esfera corpórea."

-Allan Kardec

Resumo

O uso de grades computacionais formadas com recursos de diferentes proprietários é interessante para obter alto desempenho, sem que cada participante tenha que construir um novo sistema ou expandir os recursos que já possui. Um exemplo disso é a formação de uma grade com os computadores e recursos de interconexão dos departamentos de um campus universitário. Para esse tipo de sistema o critério de justiça no uso da grade é de extrema importância, dado que cada participante não quer abrir mão da propriedade sobre seus recursos particulares e espera poder usá-los sempre que necessário. Sendo assim, um critério de justiça para essas grades consiste em oferecer aos usuários um volume de processamento no mínimo igual ao que foi ofertado ao sistema pelo usuário. Para casos em que essas grades são formadas por recursos computacionais idênticos, ou seja, são grades homogêneas, a política de escalonamento OSEP (*Owner Share Enforcement Policy*) foi criada e avaliada. Seu objetivo consiste em garantir que cada usuário seja atendido por um número de máquinas maior ou igual ao número de máquinas que ofereceu ao sistema. Dessa forma, a política OSEP fica restrita a sistemas homogêneos, uma vez que em sistemas heterogêneos é necessário levar em conta o poder computacional de cada recurso para atender ao critério de justiça, ou seja, cada usuário deve ser atendido de forma que tenha poder computacional maior ou equivalente ao poder computacional total oferecido pelos computadores que colocou na grade. Para eliminar a restrição da política OSEP foi proposta a política OSEP heterogênea. Na OSEP heterogênea foi necessário modificar o critério de justiça, bem como as métricas para avaliar sua satisfação na tomada de decisões para a alocação de recursos aos usuários. Essa nova política foi implementada e avaliada por comparações com a OSEP original usando-se o simulador de grades iSPD, com resultados que mostram um melhor desempenho no atendimento aos proprietários de recursos da grade.

Palavras-chaves: Escalonamento. Grades. Justiça.

Abstract

The use of computer grids that are assembled with resources from different owners is interesting to achieve high performance computing, without the need to build new systems or expand the ones that already exist. An example of this is the creation of a grid with computers and network resources from the departments in an university campus. For this type of system, the criteria of fairness in the use of grid resources is extremely important, as each participant does not want to give up their property rights over the resources and expect to be able to use them whenever necessary. Thus, the criteria of fairness for these grids consists in providing users with a processing volume at least equals to the volume offered by them. For grid systems formed by identical computing resources (the grid is homogeneous), the scheduling policy OSEP (Owner Share Enforcement Policy) was created and evaluated with good results. Its goal is to ensure that each user receives an amount of resources that is greater than, or at least equal to, the number of machines that he offered to the system. Thus, the OSEP policy is restricted to homogeneous systems, since that in heterogeneous systems it's necessary to consider the computational power of each resource to meet the fairness criteria, that is, each user should be served with a computing power that it is greater than, or equivalent to, the computing power offered by the computers he placed in the grid. In order to circumvent this constraint a heterogeneous OSEP policy was proposed, wich is the project presented here. In heterogeneous OSEP it was necessary to modify the fairness criteria, as well as the metrics used to evaluate the satisfaction in the decision making process to allocate resources to the users. This new policy was implemented and evaluated through comparisons with the original OSEP using the grid simulator iSPD, achieving results that show a better performance in attending the grid's resource owners.

Key-words: Scheduling. Grids. Fairness.

Lista de ilustrações

Lista de abreviaturas e siglas

OSEP	*Owner-Share Enforcement Policy*
IDE	*Integrated Development Environment*
FLOPS	*Floating-point Operations Per Second*
VPN	*Virtual Private Network*

Sumário

1 Introdução

Sistemas distribuídos consistem de máquinas ligadas em rede para resolver problemas em conjunto. O grande foco desses sistemas é o compartilhamento de recursos, que podem ser de processamento, armazenamento, etc. Com essa premissa, uma das formas de construir um sistema distribuído é a ligação de recursos de diferentes proprietários em rede. Esse tipo de sistema se encaixa na categoria das grades computacionais e é de grande interesse para indústrias e universidades, mas apresenta o problema de como dividir o uso dos recursos entre seus vários usuários. Com a finalidade de incentivar o compartilhamento de recursos surgiram, ao longo dos anos, diversos algoritmos de escalonamento com foco em critérios de justiça para nortear a alocação e compartilhamento dos recursos. Um desses algoritmos é o OSEP (FALAVINHA et al., 2009), sobre o qual o presente projeto se baseia.

1.1 Motivação

Em grades computacionais estabelecidas de forma colaborativa, com recursos de diferentes proprietários, é imprescindível respeitar a propriedade de cada um, pois tipicamente um provedor de recursos abre mão da posse para participar da grade. Assim, um possível critério de justiça para respeitar propriedade consiste em oferecer a cada usuário um volume de processamento maior ou igual ao por ele oferecido ao sistema.

Um dos algoritmos que buscam oferecer justiça através do conceito de propriedade é o OSEP (*Owner Share Enforcement Policy*) (FALAVINHA et al., 2009). O objetivo do algoritmo OSEP consiste em garantir que cada usuário seja atendido por um número de máquinas maior ou igual ao número de máquinas que esse usuário ofereceu ao sistema.

Isso torna o algoritmo eficaz em sistemas com *hardware* homogêneo, em que cada máquina apresenta o mesmo poder computacional. Entretanto, para grades heterogêneas é necessário levar em conta o poder computacional de cada máquina para atingir o critério de justiça visado. Dessa forma, o algoritmo OSEP perde sua eficiência quando aplicado em sistemas heterogêneos, ao não considerar as diferenças de desempenho de cada máquina da grade.

1.2 Objetivo

Considerando a deficiência observada do OSEP para sistemas heterogêneos, o objetivo deste projeto consiste em alterar o algoritmo OSEP de escalonamento para que ele

passe a atender eficientemente também sistemas heterogêneos e, com isso, promover ainda mais compartilhamento de recursos computacionais. Para atingir esse objetivo são analisados os conceitos de propriedade e de compartilhamento usados, assim como os critérios para definir os desvios de atendimento ao critério de justiça e de como um usuário tem sua satisfação com o sistema melhorada.

No caso de sistemas heterogêneos é preciso mudar as métricas relacionadas ao número de máquinas para métricas associadas a poder computacional. Isso ocorre por se tornar virtualmente impossível garantir que cada usuário conseguirá exatamente o mesmo poder computacional que ofereceu. Essa "impossibilidade" surge pois a menos que se restrinja a alocação de tarefas de um usuário apenas para as máquinas por ele ofertadas, a soma de poder computacional de n máquinas, distintas das compartilhadas, não será igual ao poder computacional ofertado.

1.3 Organização do Texto

O capítulo 2 a seguir, trás um estudo dos sistemas distribuídos, com destaque para grades, e dos algoritmos de escalonamento utilizados em grades computacionais para situar o OSEP em relação a eles e fundamentar o projeto. No capítulo 2 se faz uma análise detalhada do algoritmo OSEP, explicando os conceitos por ele usados. O capítulo 4 contém explicações e justificativas sobre as alterações propostas para o algoritmo original, enquanto no capítulo 5 se apresentam os modelos criados para os testes realizados, incluindo uma descrição do simulador utilizado para sua execução. Também aparecem neste capítulo os resultados obtidos assim como as conclusões tiradas deles. Por fim, o capítulo 6 apresenta as conclusões gerais e aponta para possíveis caminhos futuros deste projeto.

2 Revisão Bibliográfica

Neste capítulo são apresentados os conceitos necessários ao projeto. Assim, o capítulo é iniciado por uma descrição de sistemas distribuídos, passando para o exame de grades computacionais e, principalmente, de escalonadores para grades computacionais.

2.1 Sistemas Distribuídos

Um sistema distribuído é formado por elementos de processamento conectados por elementos de interconexão, e que se coordenam por meio de troca de mensagens. Os elementos de processamento variam desde computadores pessoais a *smartphones* e essa variação pode ocorrer dentro de um mesmo sistema distribuído. Da mesma forma, os elementos de comunicação também variam, abrangendo diversas tecnologias e diferentes pilhas de protocolos. Um exemplo de sistema distribuído é o GridUnesp [1] (UNESP, 2014), em que os elementos de processamento do GridUnesp estão espalhados pelas unidades da universidade e ligados por enlaces de alta velocidade.

A motivação para a criação desses sistemas está no compartilhamento de recursos, algo que abrange desde elementos concretos como *hardware* a objetos mais abstratos como software e os dados que são tratados por eles. No presente trabalho, se tratam os sistemas distribuídos em que se compartilha poder de processamento, de forma que seja obtido o paralelismo necessário às aplicações de alto desempenho.

2.1.1 Características dos sistemas distribuídos

De acordo com (COULORIS; DOLLIMORE; KINDBERG, 2001), um sistema distribuído qualquer deve atender a alguns requisitos:

- Heterogeneidade: Em geral, o sistema distribuído deve ser capaz de lidar com diferentes tipos de rede, sistemas operacionais, *hardware*, e linguagens de programação. Apesar disso, existem casos em que os sistemas operacionais e o *hardware* são homogêneos no sistema.

- Tratamento de falhas: Dada a multiplicidade e heterogeneidade dos elementos do sistema, é natural a ocorrência de falhas, que devem ser tratadas pelo sistema sem que haja necessidade de intervenção do usuário ou do programador da aplicação executada.

[1] UNESP. Gridunesp homepage. Disponível em <http://unesp.br/portal!/gridunesp>, acessado em 21 de agosto de 2014. 2014.

- Abertura: O sistema deve ser aberto à adição de novos recursos.
- Escalabilidade: O desempenho do sistema e sua capacidade de atendimento devem crescer de forma diretamente proporcional à adição de novos recursos, ou seja, o sistema deve ser capaz de aproveitar de forma satisfatória os recursos adicionados a ele.
- Segurança: Devem ser oferecidas aos usuários condições mínimas de segurança, principalmente em relação aos dados que eles enviam ou recebem do sistema, uma vez que esses dados podem passar por múltiplos domínios devido à distribuição dos recursos.
- Concorrência: O sistema deve ser capaz de manter a integridade de seus recursos e dados em casos de concorrência entre usuários pelos recursos.
- Transparência: A complexidade do sistema não deve ser tratada pelos usuários nem pelos programadores, o sistema deve lidar com ela sem exigir ação destes. Alguns aspectos da complexidade podem ser importantes para o programador dependendo da aplicação.

O atendimento dos requisitos listados fica a cargo do *middleware*, camada de *software* que faz a abstração da complexidade do sistema e evita que programadores e usuários tenham que lidar com ela. Isso faz do *middleware* um elemento cuja complexidade cresce conforme a abrangência do sistema distribuído e, devido à sua função-chave para o sistema, seu projeto requer cuidados especiais (COULORIS; DOLLIMORE; KINDBERG, 2001). O *middleware* situa-se entre os sistemas operacionais dos recursos de processamento e a aplicação, fazendo a interface entre eles, e, por isso, uma de suas funções é o escalonamento de tarefas, que será discutido mais adiante.

2.2 Grades

Propostas na década de 90, as grades são sistemas distribuídos em que os recursos computacionais e de armazenamento estão dispersos geograficamente, podendo abranger desde as unidades de uma universidade, caso do GridUnesp (UNESP, 2014), ao planeta todo, caso do Seti@Home [2] (SETI@HOME, 2014). Esses sistemas, em geral, são heterogêneos e podem abranger diferentes domínios administrativos, tornando-os sistemas comuns em casos de colaboração entre entidades diferentes (CIRNE; NETO, 2005).

A abrangência das grades, permite a superação dos limites impostos ao paralelismo em sistemas como *clusters* e supercomputadores, que ficam espacialmente restritos a de-

[2] SETI@HOME. Seti@home homepage. Disponível em <http://setiathome.ssl.berkeley.edu>, acessado em 21 de agosto de 2014. 2014.

terminadas áreas, mas a natureza flexível desses sistemas leva a maior complexidade de gerenciamento, uma vez que os recursos são heterogêneos, não há garantia de disponibilidade desses recursos e espera-se uma qualidade de serviço mínima oferecida pelo sistema. Sendo assim, o sucesso desses sistemas exige especialmente de gerenciamento eficiente dos recursos e da tolerância a falhas.

Para que seja uma grade, um sistema distribuído deve atender a três requisitos (FOSTER; KESSELMAN, 2003):

- Coordenação de recursos que estão em domínios administrativos diferentes. Isso leva a questões delicadas de política e segurança tanto do sistema como dos usuários.
- Padronização de interfaces. Deve-se decidir entre usar interfaces padronizadas ou abertas para ligar os elementos da grade. O uso de interfaces abertas leva a maior complexidade de gerenciamento.
- Qualidade de serviço para funcionalidades não-triviais, ou seja, oferecer tempo de resposta adequado, disponibilidade e segurança em um sistema cujos recursos são dinamicamente disponíveis, os tempos de transmissão de dados podem ser altos e a passagem desses dados por diferentes redes e domínios administrativos impõe desafios adicionais à segurança.

2.2.1 Tipos de Grades

Segundo Abbas (ABBAS, 2003), é possível classificar as grades em:

- Grades Departamentais: Grade definida para uso de um grupo ou departamento dentro de uma organização maior ou empresa. Esse tipo de grade é de uso exclusivo ao departamento a que pertence e não colabora com outros setores ou grupos.
- Grades Empresariais: Abrange toda a empresa e atende a todos os setores dela, combinando recursos disponíveis em toda a entidade e permitindo a cooperação entre os setores da empresa.
- Grades Extra-Empresariais: Agregam recursos de empresas e seus parceiros. Pode abranger clientes dessas empresas. Normalmente utiliza redes virtuais privadas (VPN) para oferecer seus serviços.
- Grades Globais: Utilizam a internet para prover serviços e emprega excesso de recursos computacionais de seus participantes. Esses recursos podem ser emprestados ou vendidos pelos participantes para a grade.
- Grades de Dados: O foco dessas grades está em tratar dados, e, por isso dependem de grande volume de recursos de armazenamento.

- Grades de Utilidades: São grades voltadas para o oferecimento de serviços a clientes. Por essa razão, uma de suas características principais é a disponibilidade.

- Grades de alto desempenho: O objetivo dessas grades é oferecer poder computacional para aplicações de alto desempenho. Para tanto, esse tipo de grade agrega grande volume de elementos de processamento, que podem variar de computadores pessoais a supercomputadores e *clusters*.

O presente trabalho tem seu foco nas grades de alto desempenho, mais especificamente no escalonamento de tarefas nessas grades.

2.2.2 Escalonamento em Grades

Uma das funções do *middleware* é escalonar tarefas ou processos no sistema, ou seja, decidir em que máquina do sistema cada tarefa ou processo deve executar. A tomada dessa decisão deve levar em conta os seguintes elementos básicos:

- Usuários.

- Recursos.

- Política de escalonamento.

A atribuição de recursos para as tarefas dos usuários deve seguir as diretrizes definidas na política de escalonamento. Essas diretrizes refletem o objetivo do sistema. Além disso, os objetivos podem ser conflitantes. Por exemplo, para sistemas em que se deseja grande vazão, ou seja, grande quantidade de tarefas atendidas por unidade de tempo, a política de escalonamento dá preferência às tarefas pequenas. Por outro lado, para sistemas com foco em utilização dos recursos, as tarefas maiores recebem prioridade (PAULA, 2009).

Em termos de organização do sistema distribuído, o escalonamento pode ser feito de três formas (FALAVINHA et al., 2009):

- Escalonador Centralizado: O escalonador controla todo o sistema e aloca todas as tarefas no sistema. Essa centralização pode gerar problemas de desempenho, já que o escalonador pode tornar-se um gargalo para a grade, e impede a definição de políticas particulares para os diferentes domínios da grade. Por esses motivos, escalonadores centralizados são mais indicados para Grades Departamentais e Empresariais.

- Escalonador Descentralizado: Para superar as limitações do escalonador centralizado, o escalonador descentralizado é dividido em escalonadores de aplicações e

escalonadores de recursos. Os primeiros decidem para que domínio devem enviar a tarefa a para que seja obtido melhor desempenho. Os segundos gerenciam domínios e, ao receberem tarefas, decidem o melhor recurso interno ao domínio para atender às tarefas.

- Escalonador Hierárquico: Neste caso os escalonadores são organizados em níveis e os níveis superiores controlam os inferiores. Essa organização favorece a escalabilidade e tolerância a falhas.

2.2.3 Escalonadores para Grades e o Critério de Justiça

Em sistemas operacionais, é comum considerar-se como justiça a divisão igualitária do tempo de CPU entre os processos, sem considerar os usuários que iniciaram cada processo (KAY; LAUDER, 1988). A justiça para usuários, em grades em que os proprietários delas não submetem tarefas, é o foco de diversos algoritmos e sistemas de escalonamento, e alguns deles são:

- *"A Fair Decentralized Scheduler For Bag-of-Tasks Applications on Desktop Grids"* (CELAYA; MARCHAL, 2010): O escalonador trabalha de forma híbrida, ou seja, é dividido em escalonador local e escalonador global. O critério de justiça se aplica aos usuários que submetem tarefas ao sistema, o objetivo é minimizar o tempo de execução de todas as tarefas por meio da distribuição justa dos recursos. Nenhuma tarefa deve ser privilegiada em relação às outras.

- *"Decentralized Priorization-Based Management Systems for Distributed Computing"* (KARTHIKUMAR; PREETHI; CHITRA, 2013): Este escalonador é totalmente descentralizado, também tem seu foco na distribuição justa de recursos entre as tarefas. Para isso, utiliza o histórico de atendimento dos usuários, e faz uso da estrutura de árvore binária para fazer os cálculos. É baseado no *Fair-Share* (KAY; LAUDER, 1988) e não procura justiça para os proprietários dos recursos computacionais.

- *"iGrid"* (MEONI, 2011): Da mesma forma que os anteriores, oferece justiça aos proprietários das aplicações e para isso faz uso do histórico de uso destes. Além disso permite aos usuários a requisição de diferentes formas de prioridade para suas aplicações.

- *"Addressing QoS in Grids through a Fairshare Meta-Scheduling In-Advance Architecture"* (TOMAS et al., 2012): Também leva em conta o passado para definir a ordem de execução dos processos em um esquema de árvore binária semelhante ao anterior. Seu foco também está nos usuários finais do sistema.

- *"Fair Scheduling Approach For Load Balancing and Fault Tolerant in Grid Environment"* (KARTHIKUMAR; PREETHI; CHITRA, 2013): Focado no balanceamento

de carga, esse algoritmo procura distribuir o poder computacional de forma justa entre as tarefas, ao mesmo tempo em que tenta maximizar a utilização da grade computacional.

Embora interessantes, os algoritmos e sistemas vistos não atendem a grades formadas de forma colaborativa, em que os recursos pertencem aos usuários. Neste caso, para que seja interessante a participação na grade uma garantia básica deve ser oferecida: os usuários devem ter à sua disposição um volume de processamento maior ou igual ao volume que ofereceram ao sistema. Este critério é o foco do presente trabalho e há poucos algoritmos de escalonamento que procuram satisfazer esse critério:

- *"Distributed Clustering"* (OZDEN; GOLDBERG; SILBERSCHATZ, 1993): Algoritmo híbrido, em termos de centralização, esse algoritmo permite aos proprietários dos recursos a definição da forma como seus recursos podem ser usados por processos de outros usuários. Essa definição pode ser feita tanto para recursos de processamento como para recursos de comunicaçao.
- *"Non-Monetary Fair Scheduling"* (SKOWRON; RZADCA, 2013): Este escalonador faz uso da Teoria dos Jogos para atender ao critério de justiça. A ideia é considerar o escalonamento como um jogo cooperativo entre os participantes da grade. Seu funcionamento se baseia em avaliar como a participação de cada proprietário de recursos afeta o desempenho dos outros participantes.

2.3 Considerações Finais

No cenário apresentado neste capítulo tem-se que há grande preocupação em oferecer justiça a usuários em sistemas cujos proprietários não são usuários. A maioria dos algoritmos apresentados é descentralizada e depende do histórico de uso da grade para tomar suas decisões. Isso torna algoritmo OSEP (FALAVINHA et al., 2009), foco do presente trabalho, um algoritmo relevante, dado que trata-se de um algoritmo centralizado, independente de histórico de uso e o critério de justiça que visa atender é adequado a grades formadas de forma colaborativa. Nessas grades os clientes também são proprietários do sistema e esperam ter à disposição um volume de processamento maior ou igual a aquele que seus recursos particulares apresentam.

3 O Algoritmo OSEP

Neste capítulo é apresentado o algoritmo OSEP (FALAVINHA et al., 2009) e são explicados os conceitos introduzidos sobre os quais ele está fundamentado.

3.1 Conceitos fundamentais

Os recursos que compõem uma grade computacional estão, normalmente, dispersos setores administrativos diferentes, e pertencem a proprietários diferentes. O sistema pode atender tanto a usuários proprietários como a usuários que não contribuem em nada para a formação do sistema. Com isso tem-se a definição dos conceitos a seguir.

- Porção: define-se como porção ou *Owner-Share* de um usuário como o conjunto de recursos de uma grade que pertencem a esse usuário. Sendo assim, em um sistema distribuído formado a partir da interação de recursos de diferentes donos tem-se que a propriedade do sistema é distribuída, e o uso desse sistema final deve seguir regras definidas por esses proprietários no acordo de formação da infraestrutura agregada.

- Propriedade Distribuída: para referência a sistemas distribuídos cujos componentes estão em domínios administrativos distintos, cujos proprietários decidem colaborar para atingir um objetivo comum, usa-se o termo propriedade distribuída. A existência de sistemas como esses está associada à solução de problemas que não podem ser resolvidos apenas com os recursos particulares de alguém. Empresas e entidades que procuram resolver tais problemas podem colaborar entre si para a solucionar tais problemas sem a necessidade de expandir seus recursos isolados.

3.2 Política *Owner-Share*

Nos sistemas tratados aqui os usuários, apesar do compartilhamento que fazem de seus recursos, ainda são seus proprietários e têm o direito de exigir o uso de seus recursos particulares a qualquer momento. Como trata-se de sistemas distribuídos, é possível atender a tal exigência com máquinas de terceiros, desde que o usuário tenha a mesma qualidade de serviço que receberia de sua porção de recursos.

Para atender à garantia de uso da porção de cada usuário em sistemas de *hardware* homogêneo foi proposta a política *Owner Share Enforcement Policy* (OSEP) ou Política de Garantia da Porção do Proprietário. A política define que um usuário do sistema deve ser atendido por um número de recursos em quantidade que seja equivalente ao menor

Algoritmo 1: Algoritmo Dinâmico de Atualização

```
1 A cada t segundos faça
2 |  Para toda Tarefa na fila do escalonador faça
3 |  |  Carrega as informações monitoradas sobre o recurso alocado e a tarefa;
4 |  |  Analisa as informações de status da tarefa e de seu proprietário;
5 |  |  Atualiza as informações do usuário na estrutura de armazenamento;
6 |  fim
7 |  Chame Algoritmo de Decisão(maxt);
8 fim
```

Figura 1: Algoritmo Dinâmico de Atualização

valor entre a demanda desse usuário, em número de tarefas, e o número de máquinas que pertencem ao usuário. Caso haja máquinas ociosas de outros usuários elas podem ser usadas.

3.3 Algoritmo OSEP

A atendimento da política OSEP necessita de dados de propriedade de todo o sistema, assim como de dados de demanda e máquinas alocadas para cada usuário. Sendo assim, é natural que o algoritmo seja centralizado para que possa recolher dados de todo o sistema. Outra característica do algoritmo é a preempção, uma vez que ele deve ser capaz de recuperar recursos em uso por usuários que possuem excesso em relação às suas porções, para atender a usuários com falta de recursos.

O algoritmo é dividido em dois, um algoritmo de atualização e um algoritmo de decisão. O primeiro trata dos dados dinâmicos do sistema, ou seja, recolhe dados sobre a demanda dos usuários e sobre a quantidade de máquinas que atende cada usuário. Esses dados são coletados periodicamente e servem de entrada para o algoritmo de decisão, que utiliza essas informações para definir a distribuição do sistema entre os usuários, e disparar as ações que levarão a grade ao estado desejado.

3.3.1 Algortimo de Atualização

O pseudocódigo para o algoritmo de atualização é apresentado na figura 1. Esse algoritmo usa duas variáveis, t e $maxt$ para controle de fluxo. A primeira consiste no intervalo de tempo entre iterações do **Algoritmo Dinâmico de Atualização**. Já $maxt$ é o número máximo de tarefas cuja execução pode ser suspensa pelo **Algoritmo de Decisão**.

A cada iteração do algoritmo são recolhidas informações sobre tarefas, recursos e usuários, que são analisados para determinar a existência de usuários com atendimento

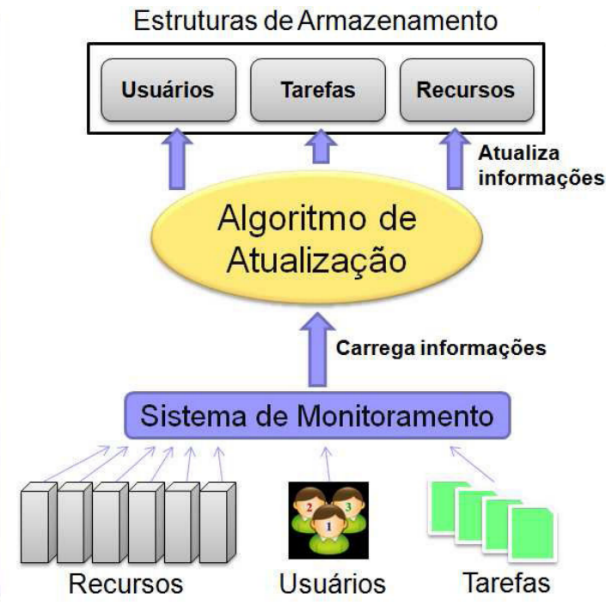

Figura 2: Diagrama do **Algoritmo Dinâmico de Atualização**

injusto. Com base no resultado dessa análise obtém-se *maxt*. Na figura 2 é apresentado um diagrama dos componentes envolvidos neste procedimento. O sistema de monitoramento da grade fornece os dados necessários, o algoritmo recebe as informações como entrada e atualiza os dados obsoletos que estavam nas estruturas de armazenamento. A partir

3.3.2 Algoritmo de Decisão

O algoritmo de decisão redistribui as tarefas no sistema com base nos dados recolhidos pelo algoritmo de atualização. Para o algoritmo de decisão, as tarefas podem assumir apenas três estados:

- ***Ativa***: A tarefa está sendo executada em uma máquina da grade.
- ***Pronta***: A tarefa está pronta para execução, aguardando para ser alocada a um recurso.
- ***Em Espera***: A tarefa foi bloqueada pelo usuário ou pelo escalonador.

O algoritmo de decisão procura fazer a preempção de tarefas em estado ***Ativa*** para alocar tarefas do estado ***Pronta***, seguindo a política OSEP. A tarefa a ser suspensa deve atender a duas condições:

1. A tarefa deve pertencer ao usuário que tem o maior excesso, em relação à sua porção, de máquinas executando suas tarefas.
2. A tarefa interrompida deve ser aquela que executou por menos tempo, a fim de que o desperdício de processamento seja minimizado.

A execução do algoritmo de decisão utiliza os seguintes parâmetros:

- Demanda não atendida do usuário, representada por $i_{usuario}$.
- Defasagem ou excesso no atendimento ($f_{usuario}$), dado pela diferença entre o número de tarefas em execução ($nt\_ativa_{usuario}$), e a porção do usuário ($p_{usuario}$): $f_{usuario} = p_{usuario} - nt\_ativa_{usuario}$.
- Tempo em que cada tarefa executou, para cada tarefa do usuário, representado por w_{job}.
- Estado em que se encontra cada tarefa, representado por s_{job}.
- Quantidade de usuários, representada por m.
- Parâmetro $maxt$, recebido do Algoritmo Dinâmico de Atualização.

Na figura 3 é apresentado o pseudocódigo do algoritmo de decisão. Na primeira linha procura-se pelo usuário que apresenta maior defasagem de uso dentre aqueles que possuem demanda não atendida. Em seguida, na segunda linha, procura-se pelo usuário que apresenta o maior excesso de uso. Na linha 3 procura-se entre as tarefas de estado ***Ativa*** do usuário com maior excesso, aquela que executou por menos tempo, representado por w_j. Na figura 4 é apresentado o diagrama para o algoritmo de decisão. O diagrama ilustra o uso das informações, a atualização delas após a decisão e a desalocação de recursos.

Algoritmo 2: Algoritmo de Decisão (int $maxt$)

```
1  Se maxt > 0 ∧ (∃u_max | f_max = max(f_1, ..., f_m) ∧ f_max > 0 ∧ i_max > 0) então
2      Se ∃u_min | f_min = min(f_1, ..., f_m) ∧ f_min < 0 então
3          Se ∃Tarefaj ∈ u_min | w_j = min(w_1, ..., w_l) ∧ (s_j = Ativa) então
4              Suspende_tarefa(j);
5              Reserva recurso para a tarefa pronta k do usuário u_max;
6              Decrementa maxt;
7              Atualiza informações dos usuários u_max, u_min e das tarefas j e k;
8          fim
9          Chame Algoritmo de Decisão(maxt);
10     fim
11 fim
```

Figura 3: Algoritmo de Decisão

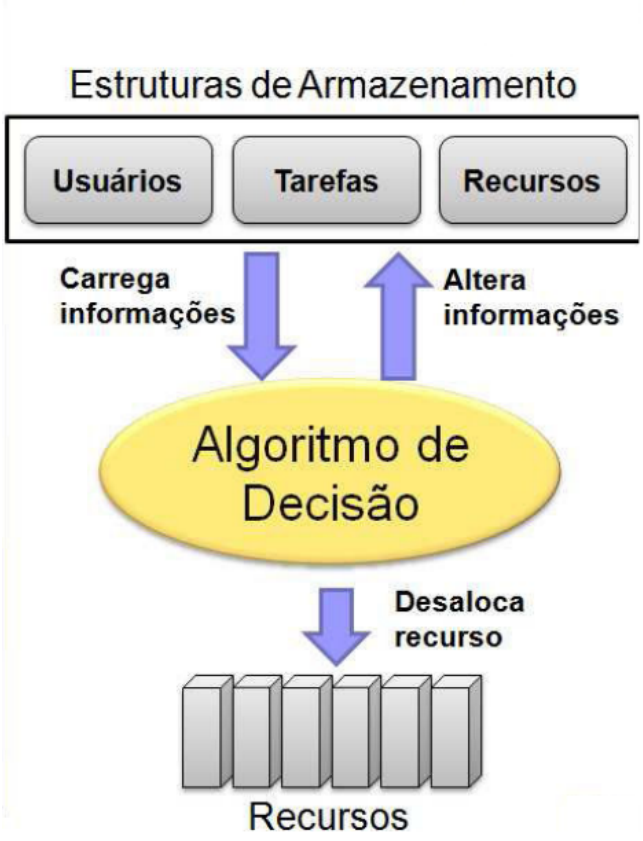

Figura 4: Diagrama do **Algoritmo de Decisão**

3.4 Eficiência e aplicabilidade do algoritmo OSEP

Em (FALAVINHA, 2009) a eficiência e a aplicabilidade do OSEP são asseguradas. Os resultados mostram que o algoritmo leva um período de tempo aceitável para deixar a grade na condição desejada, ou seja, a partir do momento em que é detectado o não cumprimento do critério de justiça, o algoritmo leva pouco tempo para ajustar a grade de forma que atenda ao critério. O algoritmo também desperdiça pouco processamento. Esse desperdício ocorre entre os instantes de suspensão da execução de uma tarefa e o início da execução de outra. Durante esse período, o recurso que executava a tarefa suspensa fica ocioso. Por fim, também ficou provado que o *overhead* imposto pelo algoritmo à grade também é pequeno.

Deve ser observado que naquele trabalho se apresenta também o conceito de *Checkpointing* de tarefas, que se refere ao armazenamento do progresso das tarefas em blocos. O uso de *checkpointing* permite que, em caso de preempção, esses blocos sejam salvos e recuperados na realocação da tarefa. Assim, é perdido apenas o processamento que não formou um bloco completo.

3.5 Considerações Finais

Neste capítulo foi detalhado o algoritmo OSEP. Mostrou-se que, para cumprir a política OSEP, o algoritmo realoca recursos de usuários que estão com mais máquinas do que suas porções para usuários que estão sendo atendidos por menos do que suas porções. Além disso, o algoritmo está circunscrito a sistemas de *hardware* homogêneo, uma vez que trabalha com número de máquinas, de forma que não considera o poder computacional particular de cada uma. No próximo capítulo a metodologia do presente projeto será apresentada, assim como serão explicados os procedimentos de teste.

4 Algoritmo OSEP heterogêneo

Neste capítulo se apresenta a especificação das alterações necessárias ao OSEP, de forma a tornar sua aplicação eficiente em sistemas heterogêneos. Isso contempla a definição de um novo critério de justiça e de métricas para alocação de recursos e satisfação do usuário, descritas na sequência.

4.1 Conceitos propostos

No algoritmo OSEP justiça consiste em garantir que cada usuário tenha à sua disposição um número de computadores maior ou igual à quantidade de recursos que esse usuário ofertou ao sistema. Como o algoritmo trabalha em sistemas de *hardware* homogêneo, tal critério de justiça garante que cada usuário usufrua de um poder computacional total maior ou igual ao seu, como demonstrado por Falavinha (FALAVINHA et al., 2009).

Para sistemas heterogêneos não basta considerar o número de máquinas que atende a cada usuário, já que cada cada máquina pode apresentar capacidades de processamento distintas. Com isso, é proposto um novo conceito de justiça, baseado nos aspectos descritos a seguir.

4.1.1 Diferencial de Poder

Para calcular o excesso ou a defasagem no atendimento de um usuário em sistemas homogêneos, basta subtrair o número de máquinas que formam sua porção, do número de máquinas que estão executando tarefas desse usuário. Se o usuário está em falta de máquinas o resultado será negativo mas, se o valor for maior que zero, o usuário está em situação de excesso. Para um sistema heterogêneo esse cálculo não pode ser aplicado, como já foi explicado, e, portanto, propõe-se outro a seguir.

Para um usuário i, dono de uma porção no sistema com capacidade de processamento de $Porcao_i$ MFLOPS, e que é atendido por $Alocado_i$ MFLOPS, calcula-se o Diferencial de Poder $DifPoder$:

$$DifPoder_i = \frac{Alocado_i - Porcao_i}{Porcao_i} \tag{4.1}$$

O resulto desse cálculo pode ser negativo ou positivo. Para valores negativos, tem-se a defasagem relativa que afeta o usuário. Se o resultado for 1, o usuário está sendo atendido exatamente pela sua porção de poder computacional. Para valores positivos, tem-se o excesso relativo que atende ao usuário.

4.1.2 Poder Remanescente

A preempção de tarefas em sistemas homogêneos não requer cálculos para verificar como vai impactar no atendimento dos usuários envolvidos, uma vez que o poder computacional de cada máquina é idêntico. Para sistemas heterogêneos, entretanto, podem ocorrer situações em que a preempção de um recurso pode afetar de forma diferente os usuários. Isso ficará mais claro com o cálculo a seguir, proposto para obter o Poder Remanescente (PR_Preemp) que atenderá a um usuário z, após a preempção da sua tarefa que está executando na máquina y. Para o cálculo considera-se o poder alocado antes da preempção $Alocado_z$, o poder da máquina y $Poder_y$, e a porção do usuário z $Porcao_z$.

$$PR\_Preemp_{z,y} = \frac{Alocado_z - Porcao_z - Poder_y}{Porcao_z} \tag{4.2}$$

De forma semelhante calcula-se o poder remanescente para o usuário w cuja tarefa passará a ser executada na máquina y.

$$PR\_Aloc_{w,y} = \frac{Alocado_w - Porcao_w + Poder_y}{Porcao_z} \tag{4.3}$$

4.2 Novo Critério de Justiça

Para sistemas heterogêneos considera-se que um usuário x está sendo atendido de forma justa se uma das seguintes condições for atendida:

1. $DifPoder_x \geq 0$.

2. $DifPoder_x < 0 \rightarrow \ \forall$ usuário $i \wedge$ máquina $j \ \mid DifPoder_i \geq 0 \ \wedge \ j$ está alocado para i $\mid PR\_Preemp_{i,j} \leq DifPoder_x$

A primeira condição indica a melhor situação, ou seja, o usuário x está usando sua porção ou mais do que ela. Na segunda condição tem-se uma situação em que não é interessante fazer uma preempção para acomodar o usuário x pois faria com que qualquer usuário cuja tarefa fosse interrompida, se tornasse mais prejudicado do que o usuário x está no momento. Embora possa ocorrer um prejuízo momentâneo, a dinâmica do sistema fará com que o x seja recompensado em outros momentos, quando estiver na situação dos outros usuários. Isso faz com que, em média, todos os usuários tenham acesso às suas porções, ou mais do que elas, ao longo do uso da grade.

4.3 Algoritmo OSEP adaptado – Algoritmo de Decisão

Para o algoritmo heterogêneo não há alteração no seu componente de atualização e define-se a seguir as alterações feitas no algoritmo de decisão.

4.3.1 Condições para a ocorrência de preempção

Para que ocorra a preempção de uma tarefa de um usuário x, que executa na máquina y, a fim de atender a um usuário k três condições devem ser atendidas:

1. $\exists\, Usuario_k \mid DifPoder_k < 0$
2. $DifPoder_k = min(DifPoder_z)$, para todo usuário z
3. $PR\_Preemp_{x,y} = max(PR\_Preemp_{z,w})$
4. $PR\_Aloc_{k,y} > PR\_Preemp_{z,y}$, para todo usuário z e toda máquina w

A primeira condição garante que apenas usuários que estão em falta de poder computacional sejam beneficiados por uma preempção. A segunda condição garante que a preempção beneficie sempre o usuário com maior defasagem de uso da grade. A terceira condição garante que dentre os usuários com excesso de volume de processamento, apenas o usuário que possui o maior excesso tenha tarefas interrompidas e, além disso, faz com que a máquina escolhida seja a menos poderosa dentre aquelas que estão atendendo o usuário $x$usuários. Por fim, a quarta condição garante que, o menor prejuízo permaneça vigente entre os dois usuários envolvidos.

4.3.2 Critério de seleção de tarefas

Seja para executar em uma máquina livre ou para ocupar uma máquina que passou por preempção, a tarefa escolhida deve ser a tarefa de menor tamanho entre as tarefas do usuário escolhido, e que estão no estado *Pronta*. Combinando esse critério com os critérios de preempção, tem-se que o sistema trabalha com os usuários que estão nas condições extremas de uso.

4.3.3 Algoritmo de Decisão

Com os conceitos e critérios definidos neste capítulo é possível apresentar o algoritmo adaptado. Da mesma forma que o algoritmo original, é usada a variável *maxt*, que é recebida do algoritmo de atualização e representa o número máximo de tarefas que podem sofrer preempção. Na figura 5 é apresentado o pseudocódigo para o algoritmo adaptado. Na primeira linha, procura-se pelo usuário com maior defasagem de uso da grade. Encontrado esse usuário, procura-se, na linha 2, pelo usuário com maior excesso de uso do

sistema. Encontrado o usuário com maior excesso, procura-se, na linha 3, por uma tarefa desse usuário que execute em uma máquina cuja preempção não leve a uma violação de justiça ainda maior que a que já existe. Atendidas a todas as condições anteriores, os procedimentos de suspensão de tarefa e alocação de nova tarefa são feitos, a variável *maxt* é decrementada e os dados são atualizados.

Algoritmo 3: Algoritmo de Decisão Adaptados (int *maxt*)

1 **Se** $maxt > 0 \wedge (\exists$ *usuário* $su_i \mid DifPoder_i = min(DifPoder_1, ..., DifPoder_m) \wedge DifPoder_i < 0$ **então**
2 **Se** $\exists$ *usuário* $u_j \mid DifPoder_j = max(DifPoder_1, ..., DifPoder_m) \wedge DifPoder_j > 0$ **então**
3 **Se** $\exists$ *Tarefa* $k \in u_j \mid s_k = Ativa \wedge k$ *executa na máquina* $y \wedge PR\_Aloc_{i,y} > PR\_Preemp_{j,y}$ **então**
4 Realiza preempção da tarefa k do usuário u_j;
5 Reserva recurso para a menor tarefa do usuário u_i;
6 Decrementa *maxt*;
7 Atualiza dados dos usuários e das tarefas;
8 **fim**
9 **fim**
10 Chame ***Algoritmo de Decisão(maxt)***;
11 **fim**

Figura 5: Algoritmo de Decisão

4.4 Satisfação do usuário

Para medir a eficácia do algoritmo OSEP modificado foi utilizada a métrica satisfação do usuário, que também foi usada nos testes do algoritmo original. Entretanto, para o presente trabalho, seu cálculo teve que ser adaptado. A equação 4.4 apresenta o cálculo para a satisfação de um usuário i. Nesta equação, $CT_{j_{ideal}}$ representa o instante de tempo em que o usuário espera que a execução da tarefa j termine, e, $CT_{j_{real}}$ representa o instante de tempo que a execução terminou de fato. AT_j representa o instante de tempo em que a tarefa j foi submetida ao escalonador. O número total de tarefas submetidas pelo usuário i é dado por n .

$$SU_i = \frac{\sum_{j=1}^{n} \left(\frac{(CT_{j_{ideal}} - AT_j)}{(CT_{j_{real}} - AT_j)} \right) \times 100}{n} \tag{4.4}$$

O cálculo da satisfação consiste na média das satisfações calculadas para cada tarefa. Para cada tarefa calcula-se a satisfação em porcentagem, dividindo o tempo esperado pelo tempo real. Se a expectativa é atingida, a satisfação será de 100 %.

4.5 Considerações Finais

O novo algoritmo de decisão, adaptado para sistemas heterogêneos, permite tratar o poder computacional de cada máquina de forma individual. As restrições para preempção impostas evitam que o algoritmo entre em um círculo vicioso de preempções/alocações entre dois usuários. A implementação desse conjunto de modificações permite o tratamento do conceito de justiça baseada em propriedade para grades heterogêneas, como será avaliado no próximo capítulo.

5 Avaliação do OSEP heterogêneo

Neste capítulo é apresentada a avaliação do da OSEP heterogênea, incluindo os modelos de grade desenvolvidos para teste e os resultados obtidos tanto com a aplicação da OSEP original quanto das modificações propostas. Os testes foram executados usando-se um ambiente de simulação de grades, evitando a necessidade de alterar os *middlewares* de ambientes reais.

5.1 Implementação dos algoritmos

Tanto o algoritmo OSEP original como o algoritmo modificado foram implementados no simulador de grades iSPD. Trata-se de um simulador robusto, capaz de simular diversos tipos de grades e seus resultados se mostraram bastante precisos (MANACERO et al., 2012). A linguagem usada foi Java, que é nativa do simulador.

5.2 Grade modelada

Para a realização dos testes usou-se o modelo de uma grade de 12 máquinas, 1 escalonador e 4 usuários. O modelo está representado na figura 6 e, nele, as máquinas apresentam diferentes capacidades de processamento, que representam valores equivalentes a processadores Intel da série Sandy Bridge, sendo assim distribuídas:

- 4 máquinas desenvolvem 132250,0 MFLOPS cada (i7).
- Outras 4 máquinas desenvolvem 54760,0 MFLOPS cada (i5).
- As últimas quatro desenvolvem 29750,0 MFLOPS cada (i3).

Cada usuário é proprietário de 3 das 12 máquinas, divididas da seguinte maneira:

- user1: 45,76%
  - 3 máquinas de 132250,0 MFLOPS
- user2: 27,88%
  - 1 máquina de 132250,0 MFLOPS
  - 2 máquinas de 54760,0 MFLOPS
- user3: 16,06%

  - 1 máquina de 29750,0 MFLOPS
  - 2 máquinas de 54760,0 MFLOPS

- user4: 10,29%
  - 3 máquinas de 132250,0 MFLOPS

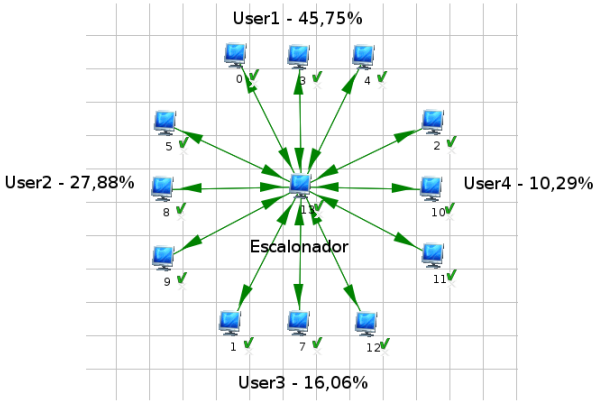

Figura 6: Modelo usado

5.3 Demanda dos usuários

A carga de trabalho dos usuários consiste em 10 tarefas submetidas por cada usuário. O número de tarefas foi definido de forma a manter a grade ocupada por tempo suficiente para afetar a satisfação dos usuários, uma vez que um número muito baixo de tarefas seria atendido rapidamente, deixado todos os usuários satisfeitos independente das porções. O tamanho das tarefas foi determinado considerando-se os dados apresentados em (DI; KONDO; CAPPELLO, 2013), que mostram que no ambiente real dos sistemas Google, 1,5% das tarefas executadas é responsável por 98,5% do tempo de execução do sistema. Dessa forma, o tamanho das tarefas foi definido de forma a permitir uma aproximação da realidade nos casos de carga de trabalho definidos mais adiante. Os tamanhos das tarefas foram definidos em:

- Tarefa Grande: Entre 634800000 MFLOP e 3332700000 MFLOP, o que corresponde a executa de 80 a 420 minutos na melhor máquina do sistema.
- Tarefa Média: Entre 238050000 MFLOP e 634800000 MFLOP, o que corresponde a executa de 30 a 80 minutos na melhor máquina do sistema.

- Tarefa Pequena: Entre 39675000 MFLOP e 238050000 MFLOP, o que corresponde a executa de 5 a 30 minutos na melhor máquina do sistema.

Considerando-se as escalas de tamanho, foram definidos três diferentes tipos de carga de trabalho para um usuário. As demandas pequena e média se aproximam mais da realidade do ambiente real dos serviços Google. A demanda grande foi usada para verificar como seria o comportamento do algoritmo em ambientes usados em pesquisas de larga escala, com grandes volumes de dados, sobre os quais são aplicadas numerosas e complexas operações. Dessa forma, a carga de trabalho é aplicada em três modelos:

- Demanda Pequena: cada usuário submete
  - 6 tarefas pequenas
  - 3 tarefas médias
  - 1 tarefa grande
- Demanda Média: cada usuário submete
  - 3 tarefas pequenas
  - 6 tarefas médias
  - 1 tarefa grande
- Demanda Grande: cada usuário submete
  - 1 tarefa pequena
  - 3 tarefas médias
  - 6 tarefas grandes

5.4 Casos de Teste

Para a avaliação foram definidos 12 casos de teste, a saber:

- Atraso de submissão das tarefas do **user1**
  - Sem *Checkpointing* de Tarefas
    - Demanda Pequena;
    - Demanda Média;
    - Demanda Grande;
  - Com *Checkpointing* de Tarefas

        * Demanda Pequena;
        * Demanda Média;
        * Demanda Grande;

- Atraso de submissão das tarefas do **user4**

    - Sem *Checkpointing* de Tarefas
        * Demanda Pequena;
        * Demanda Média;
        * Demanda Grande;
    - Com *Checkpointing* de Tarefas
        * Demanda Pequena;
        * Demanda Média;
        * Demanda Grande;

O atraso das tarefas do **user1** e do **user4**, usuários que possuem, respectivamente, a maior e a menor porção, é feito para verificar se o algoritmo de fato toma as providências para retirar recursos dos outros usuários para acomodar o **user1** ou tirar apenas o necessário para acomodar o **user4**. Os 12 casos de teste são aplicados tanto para o algoritmo original como para o heterogêneo. O atraso definido é de 6 minutos.

Nos testes feitos com *Checkpointing* definiu-se blocos de 10 minutos de execução.

5.5 Resultados

Os vários ambientes de teste descritos foram avaliados com o uso do iSPD. Para os gráficos apresentados na sequência se utiliza da seguinte legenda:

- Modificado-SC: Resultado do teste do algoritmo modificado sem o uso de *checkpointing*;
- Modificado-CC: Resultado do teste do algoritmo modificado com o uso de *checkpointing*;
- Original-SC: Resultado do teste do algoritmo original sem o uso de *checkpointing*;
- Original-CC: Resultado do teste do algoritmo original com o uso de *checkpointing*;

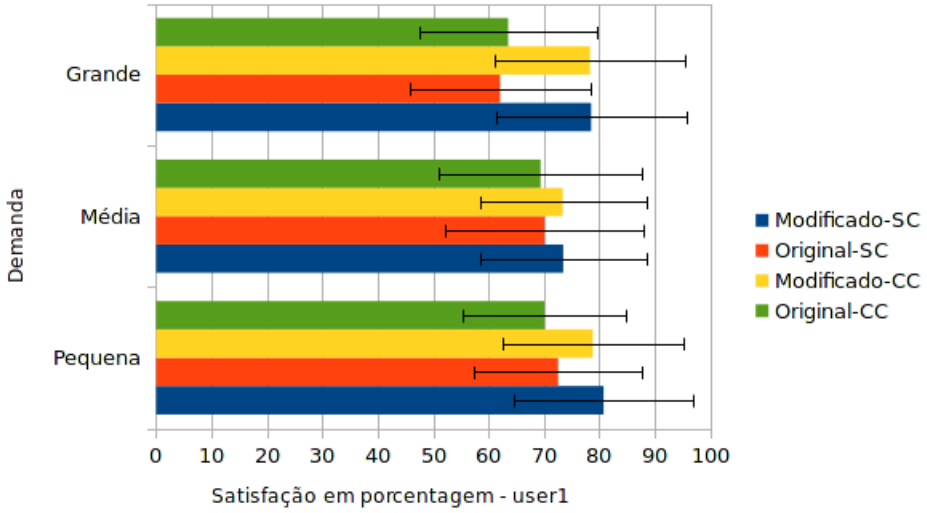

Figura 7: Resultados do **user1** para testes de atraso do **user1**

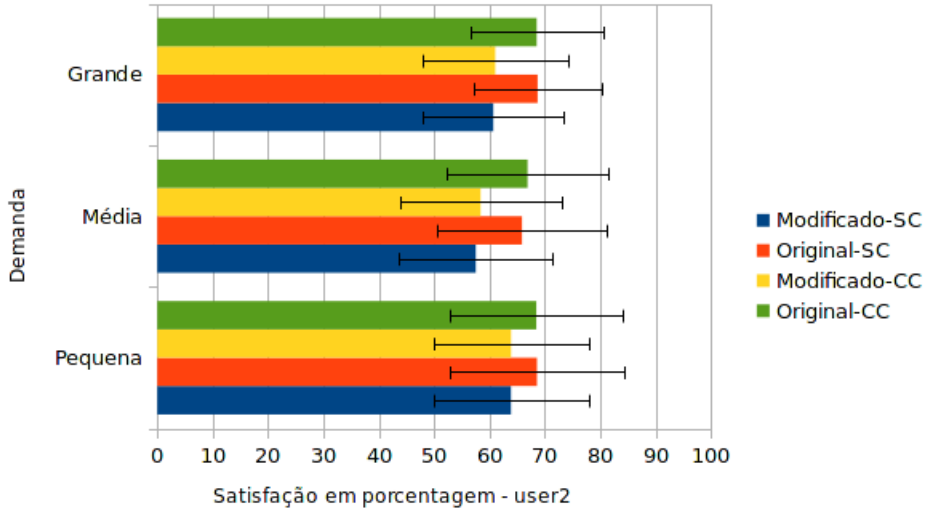

Figura 8: Resultados do **user2** para testes de atraso do **user1**

5.5.1 Resultados dos testes de atraso do **user1**

Para cada caso de teste definido foram feitos 1000 testes, e foi calculada a média dos resultados. Nas figuras 7, 8, 9 e 10 são apresentados os gráficos dessas médias.

Pelos gráficos, nota-se que o algoritmo original falha em atender ao critério de justiça. Ele entrega menos poder computacional ao **user1** do que deveria, e entrega mais do que deveria aos outros usuários. Por isso, o usuário **user4** chega a ficar 10 % mais satisfeito do que deveria, enquanto o **user1** chega a ficar quase 20% menos satisfeito do que deveria para demanda grande. Isso prova que, em uma grade de *hardware* heterogêneo, o algoritmo OSEP original não atende ao objetivo. Além disso, o algoritmo original não respeita o tamanho das porções, isso é visível ao fazer a comparação entre as satisfações dos pares (**user1,user2**) e (**user3,user4**). Em cada par, os usuários possuem porções

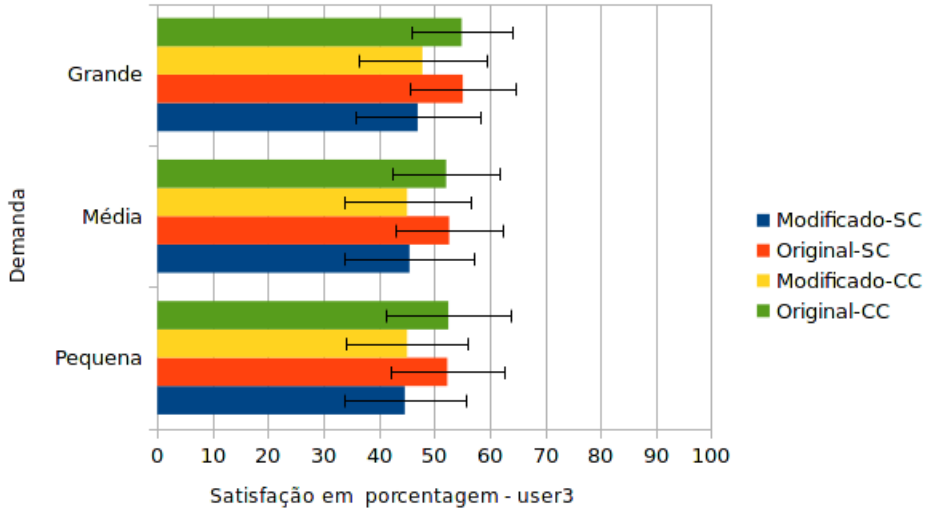

Figura 9: Resultados do **user3** para testes de atraso do **user1**

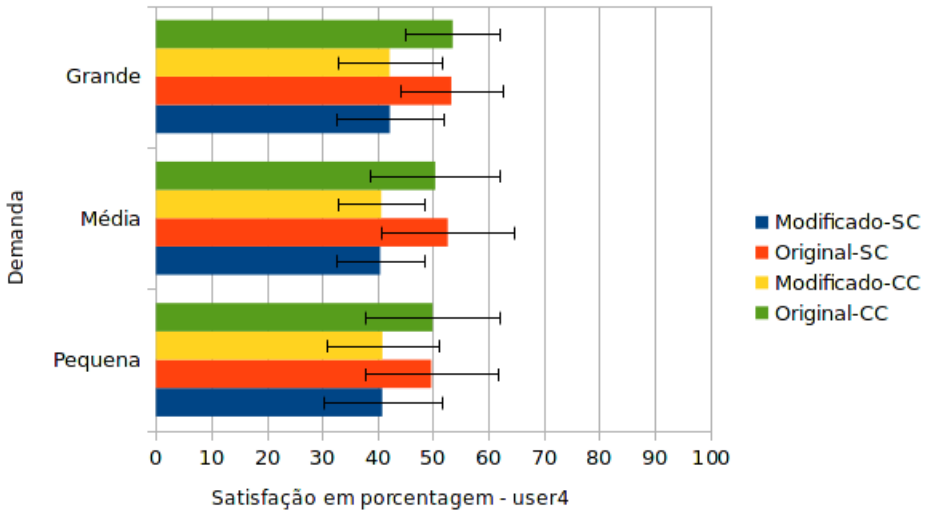

Figura 10: Resultados do **user4** para testes de atraso do **user1**

bastante diferentes do sistema, mas ficam com satisfações muito parecidas.

5.5.2 Resultados dos testes de atraso do **user4**

Da mesma forma que os testes anteriores, estes testes também foram repetidos 1000 vezes e a média dos resultados é apresentada nos gráficos das figuras 11, 12, 13 e 14.

Os gráficos mostram de maneira bastante clara o sucesso do algoritmo adaptado. Diferente do original, ele não penaliza os outros usuários de forma errada para acomodar o usuário **user4** atrasado. Nesses testes as diferenças nos resultados para **user4** e **user2** entre os algoritmos são maiores que no teste com atraso do **user1**.

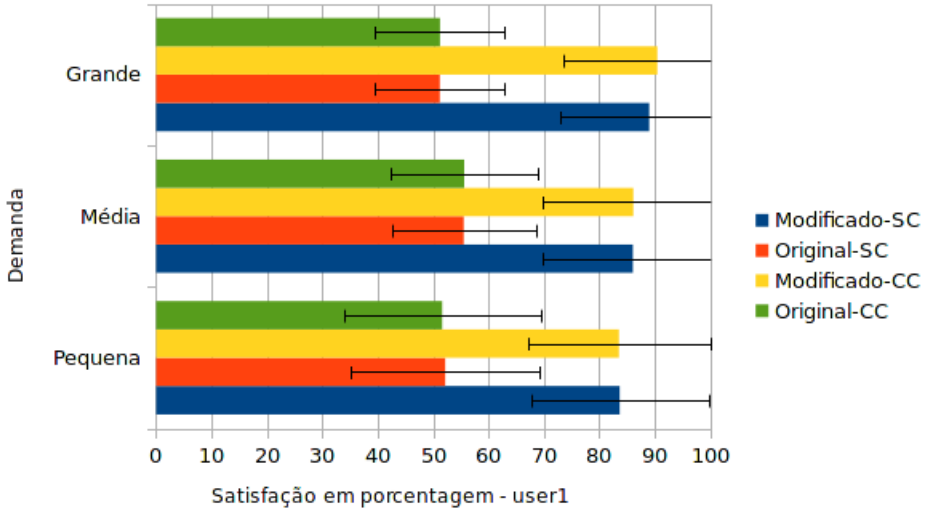

Figura 11: Resultados do **user1** para testes de atraso do **user4**

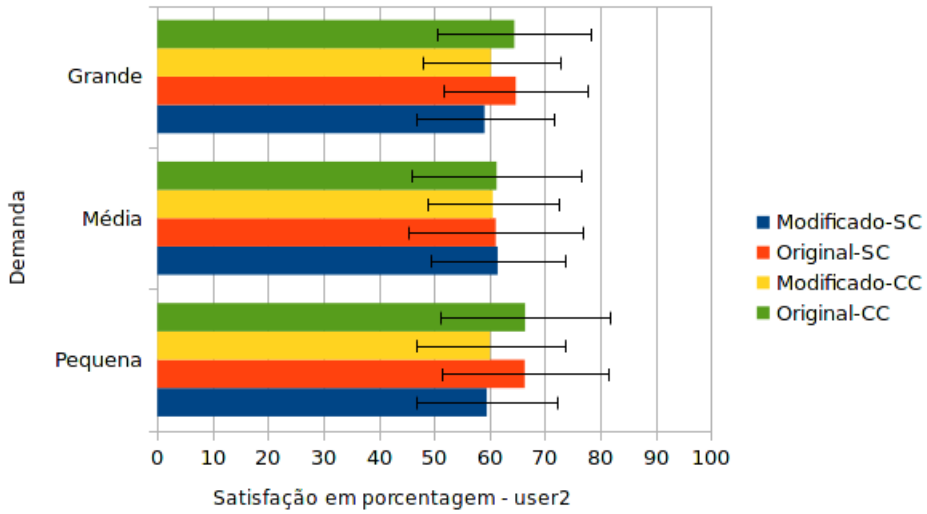

Figura 12: Resultados do **user2** para testes de atraso do **user4**

5.5.3 Gráficos de uso

Além dos gráficos de satisfação, foram calculados gráficos de uso do sistema, que mostram como o poder computacional da grade se dividiu entre os usuários. Como foram feitos 1000 testes para cada caso de teste, é inviável colocar todos os gráficos nesta monografia, sendo apresentado aqui um o resultado de uma execução para cada caso de atraso, sem uso de *checkpointing*. Para os testes com atraso do usuário **user1**, tem-se a figura 15, para o OSEP original, e a figura 16, para o OSEP heterogêneo. Já para os testes com atraso do usuário **user4**, tem-se as figuras 17 e 18, respectivamente para o algoritmo original e o heterogêneo. Pelos gráficos apresentados é possível confirmar a eficácia do algoritmo adaptado, diferente do algoritmo original, ele distribui o poder computacional de forma a respeitar o tamanho das porções de cada usuário.

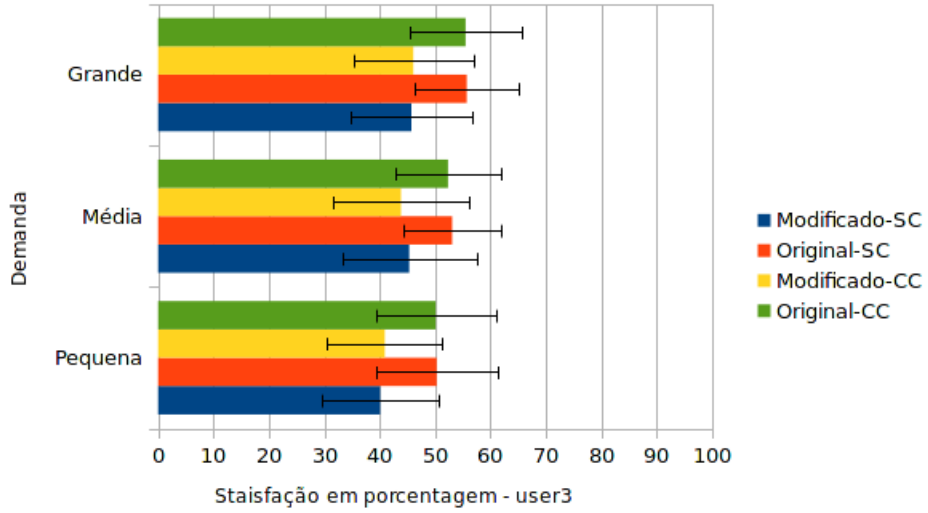

Figura 13: Resultados do **user3** para testes de atraso do **user4**

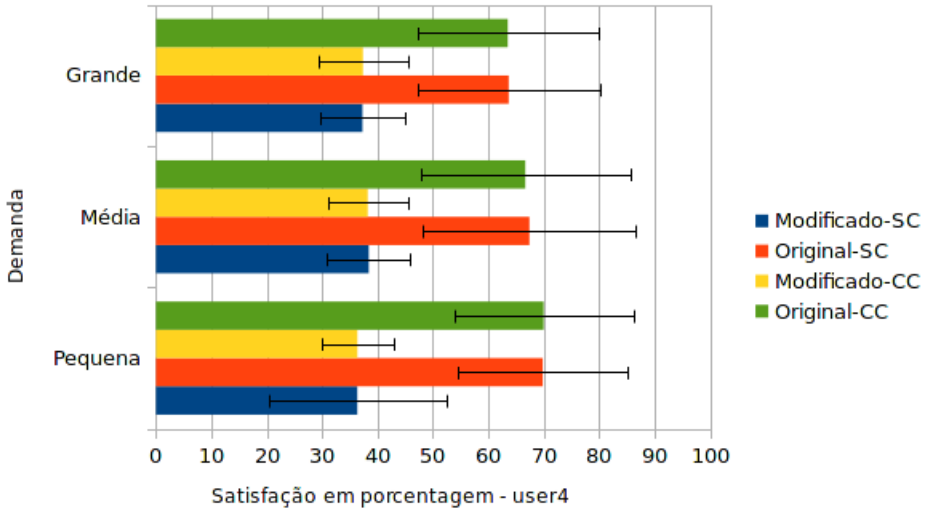

Figura 14: Resultados do **user4** para testes de atraso do **user4**

5.6 Considerações finais

Neste capítulo foram especificados os testes que foram feitos, o ambiente de teste e o modelo que foram usados, assim como foram apresentados os resultados. Percebe-se por eles que a principal falha do algoritmo OSEP original consiste em penalizar muito os usuários que tem mais recursos, quando usuários com poucos recursos chegam atrasados, e penalizar muito pouco os outros usuários, quando o usuário majoritário chega atrasado. O algoritmo heterogêneo atende à expectativa, respeitando a propriedade dos participantes sobre seus recursos. No próximo capítulo são apresentadas as observações finais para o projeto como um todo.

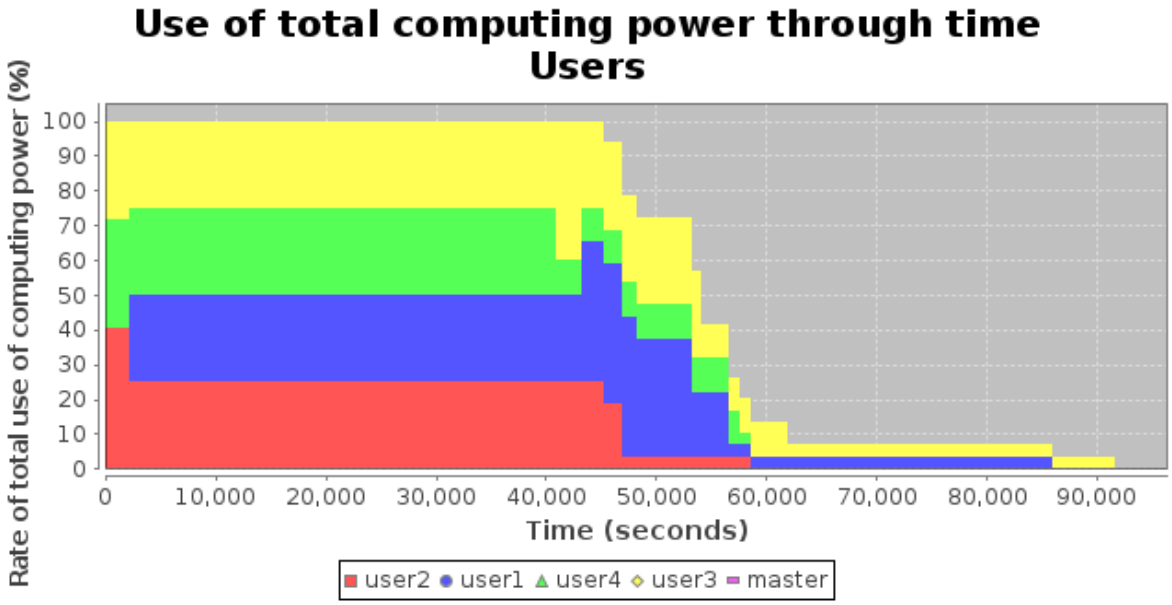

Figura 15: Taxa de uso para OSEP com atraso do **user1** (gerado pelo iSPD)

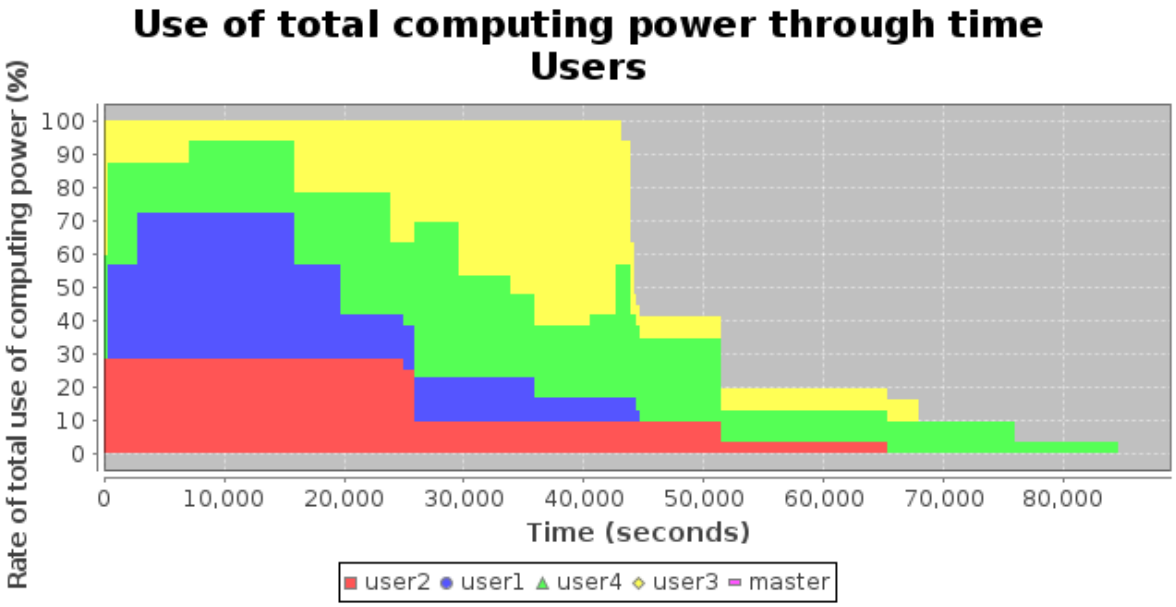

Figura 16: Taxa de uso para OSEP heterogêneo com atraso do **user1** (gerado pelo iSPD)

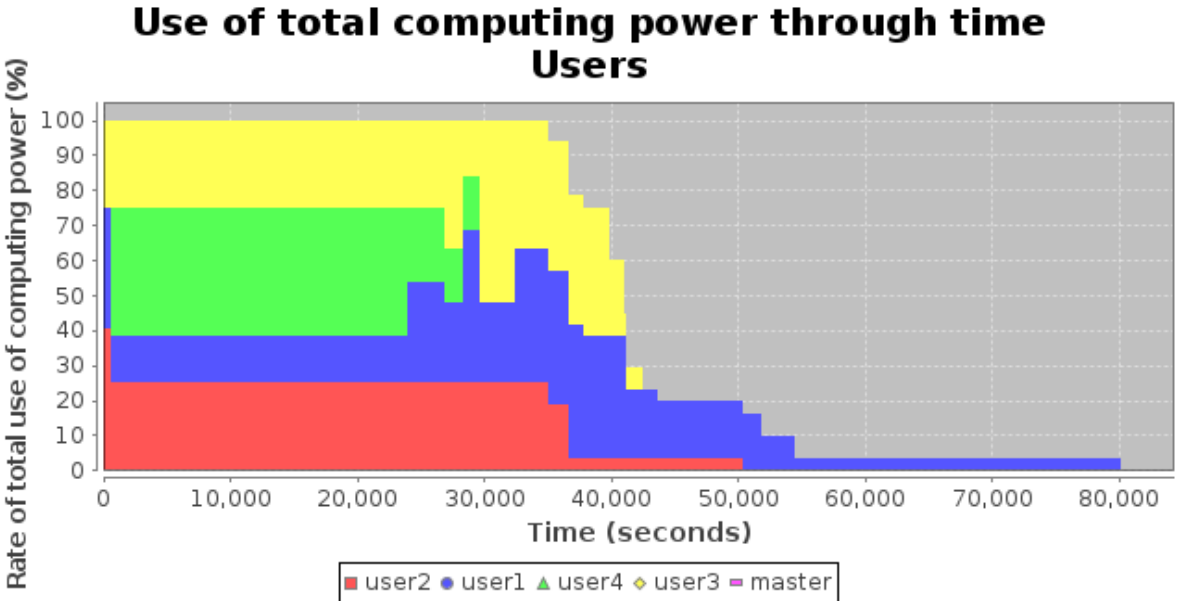

Figura 17: Taxa de uso para OSEP com atraso do **user4** (gerado pelo iSPD)

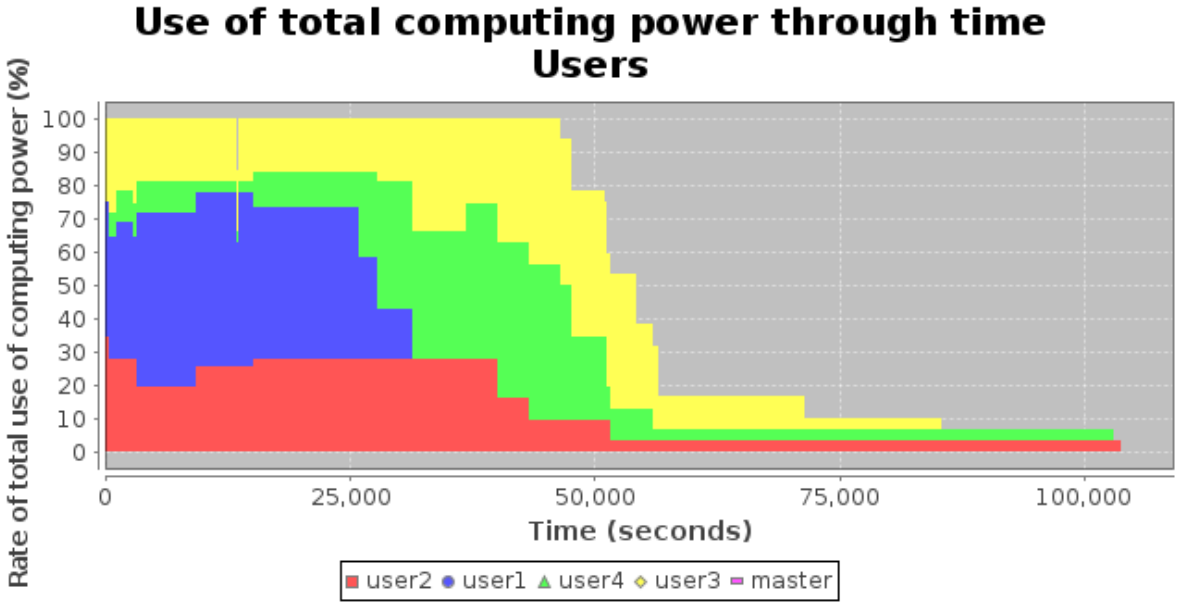

Figura 18: Taxa de uso para OSEP heterogêneo com atraso do **user4** (gerado pelo iSPD)

6 Observações Finais

Durante a leitura deste texto foram vistos os conceitos básicos de sistemas distribuídos, mais especificamente das grades, e de seus escalonadores. Foram apresentados também os conceitos e funcionamento do algoritmo OSEP e da sua adaptação para sistemas heterogêneos. Os resultados obtidos com esta versão heterogênea também foram apresentados no capítulo anterior, os quais permitem algumas conclusões sobre o trabalho executado e seus desdobramentos.

6.1 Conclusões

A necessidade de aplicação do conceito de justiça de propriedade em grades computacionais, construídas a partir de recursos compartilhados, surge para garantir que nenhum dos provedores de recursos ficará insatisfeito por não ter acesso ao poder computacional que ofertou. O algoritmo OSEP propunha uma forma de garantir essa justiça através do conceito de porção de propriedade, porém considera recursos como sendo unidades de processamento, sem diferenças entre si. Neste trabalho o conceito de justiça foi revisto, uma vez que em uma grade heterogênea é muito difícil entregar a um usuário, de forma exata, o poder computacional que ele ofereceu.

A opção feita por manter em vigor, caso necessário, sempre a menor defasagem possível se mostrou mais eficiente do que os critérios exatos da OSEP original. Os novos conceitos de Diferencial de Poder e Poder Remanescente foram propostos e utilizados para que o algoritmo passasse a levar em conta o poder computacional para tomar as decisões. Com base nessas métricas, e no algoritmo que sucessivamente busca minimizar o Diferencial de Poder, foi possível atender ao critério de justiça definido e, portanto, respeitar a propriedade distribuída da grade.

Os resultados positivos se devem ao comportamento do algoritmo adaptado, que procura sempre tirar do usuário com maior excesso para atender ao usuário com maior defasagem. Esse procedimento também é feito no algoritmo original e promove o equilíbrio do sistema de forma rápida. É importante salientar que esse procedimento tem que limitar o número de preempções de tarefas de usuários com excesso de poder computacional, evitando que esses passem a ter uma defasagem maior do que a do usuário que receberia a máquina em que ocorreria preempção. Nas implementações iniciais do OSEP heterogêneo esse problema foi percebido, e por isso, tomou-se o rumo apresentado neste projeto.

Os procedimentos adotados, os testes, e seus resultados mostram que o objetivo do projeto foi atingido. O OSEP heterogêneo supera a limitação do algoritmo original com

grades heterogêneas. Isso traz uma opção viável para grades departamentais e estimula o compartilhamento de recursos entre entidades. A maior eficiência da versão heterogênea ocorre porque o algoritmo respeita a propriedade dessas entidades sobre seus recursos particulares, garantindo que cada participante tenha à sua disposição poder computacional semelhante ao poder de seus recursos particulares. Com essa garantia e a possibilidade de usar tempo ocioso dos recursos de outros participantes, tem-se grande estímulo à colaboração, porque permite usufruir de benefícios como redução de custos e facilidade de execução de projetos colaborativos sem abrir mão dos recursos e, portanto, dos projetos particulares.

6.2 Direções Futuras

Para o futuro do projeto tem-se como possibilidade principal a adição de usuários não proprietários, ou seja, tomar medidas para tratar usuários que não contribuem com a grade, de maneira que eles possam usar a grade em momentos de ociosidade dos usuários proprietários. Isso é importante porque, mesmo em um cenário de colaboração ainda pode haver tempo ocioso na grade. Isso abre espaço para usuários externos, e, o que é ainda mais interessante, leva à possibilidade de um modelo de negócio, em que os usuários externos pagariam para usar esse poder computacional. O dinheiro obtido poderia ser usado para expandir a grade, tornando possível a execução de projetos de tamanho e complexidade crescentes.

6.3 Publicação e Financiamento

Este projeto foi financiado com bolsa PIBIC/Reitoria e uma versão preliminar dele foi publicada na Escola Regional de Alto Desempenho de São Paulo:

FORTE, C. H. V.; Denison Menezes; Aleardo Manacero; Jose N. Falavinha Jr; Renata S. Lobato. Adaptação do Algoritmo OSEP de escalonamento para sistemas heterogêneos. In: IV Escola Regional de Alto Desempenho - ERAD 2013, 2013, São Carlos - SP. Anais da VI Escola Regional de Alto Desempenho de São Paulo, 2013. p. 114-117.

Cabe citar que este trabalho foi um dos três trabalhos de Iniciação Científica selecionados para apresentação oral, dentre os 21 trabalhos selecionados para o ERAD/SP 2013.

Referências

ABBAS, A. *GRID COMPUTING: A Practical Guide to Technology and Applications*. Rockland, MA, USA: Charles River Media, Inc., 2003. ISBN 1584502762. Citado na página 5.

CELAYA, J.; MARCHAL, L. A fair decentralized scheduler for bag-of-tasks applications on desktop grids. In: *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*. [S.l.: s.n.], 2010. p. 538–541. Citado na página 7.

CIRNE, W.; NETO, E. S. *Grids Computacionais: Da Computação de Alto Desempenho a Serviços sob Demanda*. 2005. (23 Simpósio Brasileiro de Redes de Computadores). Citado na página 4.

COULORIS, G.; DOLLIMORE, J.; KINDBERG, T. *Distributed Systems concepts and design*. 3rd. ed. [S.l.]: Addison Wesley, 2001. Citado nas páginas 3 e 4.

DI, S.; KONDO, D.; CAPPELLO, F. Characterizing cloud applications on a google data center. In: *Parallel Processing (ICPP), 2013 42nd International Conference on*. [S.l.: s.n.], 2013. p. 468–473. ISSN 0190-3918. Citado na página 22.

FALAVINHA, J. *Escalonamento de tarefas em sistemas distribuídos baseado no conceito de propriedade distribuída*. Tese (Doutorado) — Universidade Estadual Paulista (UNESP), Ilha Solteira, 2009. Citado na página 13.

FALAVINHA, J. et al. The owner share scheduler for a distributed system. In: *Parallel Processing Workshops, 2009. ICPPW '09. International Conference on*. [S.l.: s.n.], 2009. p. 298–305. ISSN 1530-2016. Citado nas páginas 1, 6, 8, 9 e 15.

FOSTER, I.; KESSELMAN, C. *The Grid 2: blueprint for a new computing infrastructure*. [S.l.]: Morgan Kaufmann, 2003. Citado na página 5.

KARTHIKUMAR, S.; PREETHI, M.; CHITRA, P. Fair scheduling approach for load balancing and fault tolerant in grid environment. In: *Emerging Trends in Computing, Communication and Nanotechnology (ICE-CCN), 2013 International Conference on*. [S.l.: s.n.], 2013. p. 446–451. Citado na página 7.

KAY, J.; LAUDER, P. A fair share scheduler. *Commun. ACM*, ACM, New York, NY, USA, v. 31, n. 1, p. 44–55, jan. 1988. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/35043.35047>. Citado na página 7.

MANACERO, A. et al. ispd: an iconic-based modeling simulator for distributed grids. In: *Annals of 45th Annual Simulation Symposium*. Orlando, USA: [s.n.], 2012. (ANSS12, CDROM), p. 1–8. Citado na página 21.

MEONI, M. Interactivity on the grid: Limitations and opportunities. In: *Proceedings of the 20th International Symposium on High Performance Distributed Computing*. New York, NY, USA: ACM, 2011. (HPDC '11), p. 273–274. ISBN 978-1-4503-0552-5. Disponível em: <http://doi.acm.org/10.1145/1996130.1996171>. Citado na página 7.

OZDEN, B.; GOLDBERG, A. J.; SILBERSCHATZ, A. Scalable and non-intrusive load sharing in owner-based distributed systems. In: *Parallel and Distributed Processing, 1993. Proceedings of the Fifth IEEE Symposium on*. [S.l.: s.n.], 1993. p. 690–699. Citado na página 8.

PAULA, N. C. *Um ambiente de monitoramento de recursos e escalonamento cooperativo de aplicações paralelas em Grades Computacionais*. Tese (Doutorado) — Universidade de São Paulo (USP), São Paulo, 2009. Citado na página 6.

SETI@HOME. Seti@home homepage. Disponível em <http://setiathome.ssl.berkeley.edu>, acessado em 21 de agosto de 2014. 2014. Citado na página 4.

SKOWRON, P.; RZADCA, K. Non-monetary fair scheduling: A cooperative game theory approach. In: *Proceedings of the Twenty-fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures*. New York, NY, USA: ACM, 2013. (SPAA '13), p. 288–297. ISBN 978-1-4503-1572-2. Disponível em: <http://doi.acm.org/10.1145-/2486159.2486169>. Citado na página 8.

TOMAS, L. et al. Addressing qos in grids through a fairshare meta-scheduling in-advance architecture. In: *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2012 Seventh International Conference on*. [S.l.: s.n.], 2012. p. 226–233. Citado na página 7.

UNESP. Gridunesp homepage. Disponível em <http://unesp.br/portal!/gridunesp>, acessado em 21 de agosto de 2014. 2014. Citado nas páginas 3 e 4.