Política de escalonamento Owner-Share para sistemas de grades heterogêneas

C.H.V. Forte[1], A. Manacero[1], R.S. Lobato[1], R. Spolon[2], J.N. Falavinha-Jr[3]

[1]Depto. de Ciências de Computação e Estatística
Universidade Estadual Paulista - UNESP
Campus de São José do Rio Preto

[2]Depto. de Computação
Universidade Estadual Paulista - UNESP
Campus de Bauru

[3]Flextronics Instituto de Tecnologia - FIT
Sorocaba

{aleardo,renata}@ibilce.unesp.br, cassiohvforte@gmail.com

Abstract. *Grid environments are an efficient infrastructure to share computing resources over the internet, being the foundation of cloud computing. One important open issue with grids is how to allocate resources among users, specially when the grid resources are provided by their own users. For these grids several fair policies have been proposed, based in usage history, resource provisioning, or ownership for example. In this paper we present an extension for an owner-share based policy (OSEP), aiming a more effective control of heterogeneous nodes. This policy is based in the attribution of penalties for unfair use of resources. The definition and calculus of the penalties are presented, as well the results achieved.*

Resumo. *Grades formam uma infraestrutura eficiente para compartilhar recursos computacionais na internet, sendo a base para computação em nuvem. Um problema importante com grades é como alocar recursos entre usuários, em especial quando os recursos são fornecidos pelos próprios usuários. Várias políticas de justiça têm sido propostas, baseadas no histórico de uso, provisão de recursos, ou propriedade, por exemplo. Aqui apresentamos uma extensão de uma política baseada em propriedade (OSEP), buscando um maior controle de nós heterogêneos. Essa política é baseada em penalidades atribuídas por uso injusto de recursos. A definição e forma de cálculo das penalidades são descritas, assim como os resultados obtidos.*

1. Introdução

Grades computacionais foram introduzidas para reduzir custos no processamento de grandes volumes de dados pelo compartilhamento de recursos distribuídos fornecidos por proprietários diversos [Livny and Raman 1998]. Grades podem ser viabilizadas por grupos de usuários que compartilhem entre si o *hardware* que possuam. Nesses sistemas o principal problema é definir quanto dos recursos um dado usuário pode

ocupar a cada momento, de modo a fazer o melhor uso da infraestrutura, ao mesmo tempo em que todos os usuários fiquem satisfeitos com essa alocação. Diversas políticas têm sido propostas para resolver esse problema, incluindo o escalonador Fair-Share [Kay and Lauder 1988], estendido para grades em [Doulamis *et al.* 2007], os quais consideram o histórico de uso como critério de justiça.

Uma abordagem diferente para justiça, chamada OSEP (*Owner Share Enforcement Policy*), foi proposta por Falavinha et al. [Falavinha Jr. *et al.* 2009]. OSEP se aplica num ambiente compartilhado preemptivo [Amar *et al.* 2007] usando preempção de tarefas para atingir a quantidade de recursos devidos a um usuário. A quantidade de recursos em uso por um dado usuário é ajustada dinamicamente de acordo com quantos recursos ele disponibilizou. Isso é realizado a partir de informações sobre a grade (propriedade e tarefas alocadas) em vez de dados históricos como pelo algoritmo Fair-Share. Entretanto, para simplificar sua operação, OSEP considera recursos como sendo nós homogêneos, isto é, cada nó é contado como uma unidade de elemento de computação, qualquer que seja seu poder computacional.

Em sistemas heterogêneos, que são a maioria das grades, a abordagem adotada no OSEP não consegue assegurar uma justiça completa, pois pode alocar aos usuários máquinas menos poderosas do que aquelas que forneceram. Por isso apresentamos aqui uma extensão ao OSEP, em que recursos são definidos por seu poder computacional, medido em MFlops. Essa extensão, denominada HOSEP, introduz penalidades na verificação de justiça para usar o poder computacional como métrica.

No restante do texto são apresentados detalhes do HOSEP, incluindo como as penalidades são introduzidas na política original e como isso não invalida o princípio de justiça por proriedade. Também se apresentam os resultados obtidos com a aplicação desta abordagem para ambientes heterogêneos, comparando seu desempenho ao do OSEP em diferentes cenários.

2. Trabalhos relacionados

O projeto de escalonadores para grades tem sido uma área de pesquisa relevante desde a introdução de grades. Como se sabe, escalonamento é um problema NP-completo, dificultando a obtenção de soluções eficientes, mesmo para sistemas simples. Isso é ainda mais difícil para sistemas distribuídos, com recursos compartilhados. Vários escalonadores foram propostos, como Workqueue-with-replication, Round-Robin, Dynamic-FPLTF e XSufferage [Fujimoto and Hagihara 2004], usando abordagens diversas como algoritmos genéticos, algoritmos estáticos, dinâmicos, etc. [Qureshi *et al.* 2014], porém sem oferecer solução definitiva.

Apesar de buscarem reduzir o *makespan* das tarefas em execução, esses algoritmos não atacam o problema de provisão de recursos. Para tratar o provisionamento existem propostas como o escalonador Fair-Share [Doulamis *et al.* 2007], e algumas de suas extensões ([Amar *et al.* 2007], [Arpaci-Dusseau and Culler 1997], [Tomas *et al.* 2012]), que tratam diferentes objetivos no provisionamento, como QoS ou alocação de *clusters*.

Outra abordagem foi proposta por Andrade et al. [Andrade *et al.* 2004], na qual tentam organizar a alocação de tarefas como uma rede de favores entre

provedores de recursos, com usuários conseguindo maior prioridade se forneceram mais recursos em momentos anteriores. Essa é uma abordagem interessante para ambientes P2P.

Numa linha diferente, Hadoop habilita dois mecanismos diferentes de justiça adicionados ao seu mecanismo *map-reduce*. Um é denominado **Fair Scheduler** [Hadoop Project 2016b], desenvolvido pela Facebook para oferecer resposta rápida para tarefas pequenas. O segundo mecanismo, denominado **Capacity Scheduler** [Hadoop Project 2016a], foi desenvolvido pela Yahoo buscando o compartilhamento de *clusters* por múltiplos clientes. Em ambos casos não existe preocupação sobre quem fornece os recursos, apenas com respostas rápidas para usuários individuais.

OSEP [Falavinha Jr. *et al.* 2009] introduz o conceito de propriedade distribuída para garantir justiça entre provedores de recursos. Esta abordagem é diferente das anteriores pois não necessita do histórico das tarefas executadas na grade. OSEP necessita apenas dos dados de propriedade e do estado atual das tarefas em execução, como será descrito na próxima seção. Seu fundamento, assim como em [Doulamis *et al.* 2007] e [Andrade *et al.* 2004], é tornar o compartilhamento interessante para quem fornece os recursos, uma vez que os fornecedores podem utilizar, quando disponíveis, os recursos de outros provedores, tendo a garantia de que usarão algo equivalente a seus recursos particulares sempre que necessário.

3. Extensão do OSEP para sistemas heterogêneos

A aplicação do conceito de propriedade introduzida pelo OSEP foi restrita a grades homogêneas, isto é, grades em que os nós tenham (ou assuma-se que tenham) o mesmo poder computacional e recursos. Isso é claramente pouco realístico, embora possa oferecer uma aproximação básica de sistemas compartilhados. Neste trabalho se apresenta uma versão heterogênea do OSEP, baseada na quantidade de MFlops fornecidos por cada proprietário e pela quantidade de MFlops demandada por cada tarefa submetida. Para melhor compreender esta proposta é preciso descrever brevemente o algoritmo OSEP original e por quais motivos ele não pode tratar eficientement sistemas heterogêneos.

3.1. O algoritmo OSEP

OSEP (*Owner-Share Enforcement Policy*) [Falavinha Jr. *et al.* 2009] é um algoritmo de escalonamento para grades colaborativas, em que a propriedade dos recursos é distribuída entre seus usuários, com critério de justiça baseado no conceito de propriedade. Este tipo de justiça parte do princípio de que usuários estariam propensos a compartilhar mais recursos para a grade caso tenham a garantia de obter pelo menos a quantidade de recursos oferecidos sempre que necessitarem. No restante do texto se define **porção** como a quantidade de recursos fornecidos por um usuário.

O algoritmo OSEP se baseia em um controle em dois níveis, em que o componente do nível inferior (***Dynamic Algorithm***) coleta dados sobre o estado da grade, tais como tarefas ainda não atendidas, desempenho dos recursos alocados e dos não-alocados. Essas informações são utilizadas para atualizar o estado das filas e dos usuários. Já o componente superior (***Decision Algorithm***) fica encarregado de garantir a justiça.

O *Dynamic Algorithm* não será descrito aqui pois é apenas uma coleção de procedimentos de medição oferecidos pelo sistema. O protótipo implementado fez uso de funções do ambiente Condor, como o mecanismo **ClassAds** para coletar dados dos nós da grade [Thain *et al.* 2005]. Há diversos tipos de ClassAds, como **Job**, com dados sobre usuário que submeteu uma tarefa, máquinas que a executam, arquitetura e S.O. dessas máquinas, tempo de execução, tempo de espera, etc. Outros tipos de ClassAds, como **Machine**, **DaemonMaster** ou **Submitter**, oferecem informações adicionais sobre número de CPUs, desempenho (KFlops/MIPS), servidor de **Checkpointing** das tarefas ou número de tarefas não atendidas.

O *Decision Algorithm*, visto na Figura 1, usa dados do ClassAds para decidir se uma tarefa será interrompida ou não. Na linha 1 é determinado se existe usuário (u_{max}) com demanda não atendida ($i_{max} > 0$), usando menos recursos do que forneceu ($f_x = ofertados - possuídos > 0$). Caso exista se determina, na linha 2, qual usuário (u_{min}) está usando mais recursos acima de sua porção ($f_{min} < 0$). Para u_{min} se identifica qual tarefa deve ser interrompida, que é aquela que causará o menor impacto (ω_j), ou seja, menor volume de computação desperdiçado entre os ativos (linha 3). No OSEP este impacto é dado pela quantidade de computação que será refeita ou atrasada (se o uso de *checkpointing* for possível). O algoritmo é finito, apesar de recursivo, pois é demonstrável que é impossível ter usuário com mais recursos do que forneceu e que não tenha tarefa que possa ser interrompida.

Decision Algorithm (int p) ## p é o número de tarefas na grade
1. if ($p > 0 \ \wedge \ (\exists \ u_{max} \ | \ f_{max} = max(f_1, ..., f_m) \ \wedge \ f_{max} > 0 \ \wedge \ i_{max} > 0)$)
2. { if ($\exists \ u_{min} \ | \ f_{min} = min(f_1, ..., f_m) \wedge f_{min} < 0$)
3. { if ($\exists \ job \ j \ \in u_{min} \ | \ \omega_j = min(\omega_1, ..., \omega_l) \ \wedge \ (s_j = Active)$
4. { Vacate $job(j)$;
5. Allocate resource to the the idle job k submitted by u_{max};
6. Decrease p;
7. Update the ClassAds of users u_{max} and u_{min};
8. }
9. Call ***Decision Algorithm(p)***;
10. }
11. }

Figura 1. Decision Algorithm do OSEP [Falavinha Jr. et al. 2009]

Uma vez que o objetivo do OSEP é oferecer justiça baseado no conceito de propriedade distribuída, também é necessário introduzir uma métrica de desempenho para medir a satisfação de um usuário com a grade em que o OSEP opera. As equações 1 e 2 determinam a satisfação do usuário (**User Satisfaction, US**), em que US_{i_j} é a satisfação do usuário i pela execução da tarefa j, enquanto US_i é a satisfação média de um usuário considerando todas as tarefas por ele submetidas durante um dado período. CT_j e AT_j são respectivamente os tempos de término e de chegada da tarefa j, e n é o número de tarefas submetidas pelo usuário i. Observe-se que os tempos de término reais ($CT_{j_{real}}$) dão a medida da satisfação real, quando comparados com os tempos de término ideais ($CT_{j_{ideal}}$), determinados considerando que o usuário obteria instantaneamente todos os recursos para execução.

$$US_{i_j} = \left(\frac{CT_{j_{ideal}} - AT_j}{CT_{j_{real}} - AT_j} \right) x100 \quad (1)$$

$$US_i = \frac{\sum_{j=1}^{n} US_{i_j}}{n} \quad (2)$$

3.2. Especificação do OSEP heterogêneo

Como já descrito, OSEP não considera diferenças no poder computacional dos nós compartilhados. Ele considera que um usuário oferece uma quantidade de nós (N_i) para a grade e requisita a execução de um número de tarefas (M_i) num dado momento. Como cada tarefa é alocada individualmente a um único nó, a política de alocação trata apenas operações unitárias, alocando os recursos compartilhados às tarefas requisitadas. Usando essa abordagem pode ser verificado que se $M_i > N_i$, o usuário não receberá menos que N_i nós.

Infelizmente, receber N_i nós em sistema heterogêneo não garante que esses nós fornecerão o mesmo poder computacional (em MFlops) que o usuário ofereceu com seus N_i nós. Isso acontece porque o OSEP tenta fornecer a mesma "quantidade de" recursos possuídos pelo usuário e não os "mesmos" recursos. Uma solução poderia ser forçar o sistema a alocar sempre os "mesmos" recursos sempre que requisitados, mas isso é ineficiente pois aumentaria o número de preempções sempre que um usuário requisitar recursos, levando a grande desperdício de processamento.

Sabendo que existem diferenças no poder computacional entre os nós oferecidos e os nós recebidos, pode ser assumido que alguns usuários não receberão sua porção completa. Para minimizar essa diferença se propõe uma extensão ao OSEP chamada **Heterogeneous OSEP (HOSEP)**. Nessa extensão as decisões feitas no algoritmo da Figura 1 são modificadas para incluir uma função de diferença de poder computacional.

Definição 1. A diferença de poder computacional (***PowerDiff***) é a diferença relativa entre o poder computacional fornecido e o alocado, como definido pela equação 3 para um dado usuário x.

$$PowerDiff_x = \frac{Allocated_x - Provided_x}{Provided_x} \quad (3)$$

Se existir um usuário com $PowerDiff_x < 0$ o procedimento de alocação deve tentar alocar um nó adicional para esse usuário. Isso é obtido com um nó disponível, se existir, ou pela preempção de um nó previamente atribuído a outro usuário. Quando ocorre o último caso é necessário determinar qual usuário terá a menor penalização por ter uma tarefa atrasada.

Definição 2. O poder computacional restante resultante em se atrasar uma tarefa é calculado quando essa tarefa é excluída do conjunto alocado para aquele usuário. Essa medida, definida como poder computacional restante (***RP***) do usuário z, é determinada pela equação 4, para uma máquina y de poder computacional $Power_y$,

sendo que a diferença entre o valor de RP e $PowerDiff$ representa a penalidade dessa operação.

$$RP_{z,y} = \frac{Allocated_z - Provided_z - Power_y}{Provided_z} \tag{4}$$

Definição 3. Para um dado usuário z, se define ω_y como seu poder restante máximo, isto é, o poder computacional que restará em seu uso se o nó menos potente y for retirado de seu conjunto. Isso significa que $\omega_y = RP_{z,y}$.

A definição anterior permite a especificação de uma nova condição para justiça de propriedade, isto é, o estado em que um usuário é considerado ter recebido a sua porção justa, sabendo que sob as restrições de sistemas heterogêneos é difícil ter $PowerDiff = 0$ para todos os usuários.

Definição 4. Um usuário x tem sua porção de recursos justa se uma das condições é observada:

i) $PowerDiff_x \geq 0$, ou

ii) $PowerDiff_x < 0 \implies \forall i, j \mid PowerDiff_i \geq 0 \wedge (j \text{ alocado para } i \mid RP_{i,j} \leq PowerDiff_x)$

A condição i) apenas indica que a alocação é justa para o usuário, pois ele recebe pelo menos tantos recursos quanto ofereceu. Já a condição ii), quando o usuário recebe menos recursos do que forneceu, diz que a alocação é justa se qualquer nó a ser retirado de outro usuário levará a uma situação mais injusta do que a atual, ou seja, se as restrições dadas pelas equações a seguir não forem verificadas:

$$\exists\ User_k \mid PowerDiff_k < 0 \tag{5}$$

$$\exists\ User_x \mid RP_{x,j} > \|PowerDiff_k\| \tag{6}$$

A restrição 5 obviamente diz que um usuário pode tomar um recurso de outro usuário somente se o poder computacional em sua posse for menor do que o que foi oferecido. Já a restrição 6 diz que a tomada do nó é possível apenas se o poder computacional a ser retirado não fará com que o usuário que o perder fique com menos poder computacional do que o poder atual do usuário que receberia o nó. Se (5) e (6) forem atendidos, então o usuário x que terá um nó removido é determinado pela equação 7.

$$User_{to\_preempt} = \ User_x \mid \ RP_{x,j} = max(RP_{z,y}) \tag{7}$$

Usando essas equações é possível modificar a seleção feita na linha 2 do Decision Algorithm (figura 1) para escolher o usuário com o maior poder computacional restante (Equação 7). Ao adotar esse critério garante-se que o usuário que terá um nó tomado será aquele que sofrerá o menor atraso, isto é, aquele que pagará a menor penalidade por usar a grade em excesso.

Definição 5. A métrica de satisfação do usuário (US) é redefinida a partir de um novo valor de tempo de término ideal ($CT_{j_{ideal}}$), uma vez que esse tempo agora depende da máquina utilizada. Considerando que a satisfação é dependente dos recursos reais do usuário, o novo valor de $CT_{j_{ideal}}$ é determinado pelo instante em que a tarefa terminaria se alocada, instantaneamente, a uma máquina ideal com desempenho equivalente à média dos desempenhos dos nós que compõem a porção do usuário.

3.3. Alocando novas tarefas

O processo faz inicialmente a alocação de tarefas a nós desocupados. Se não houverem mais nós disponíveis o algoritmo passa a aplicar o modelo de justiça descrito, possivelmente tomando nós de usuários com excesso de poder computacional. Para minimizar o impacto de preempções, e maximizar a satisfação dos usuários, o procedimento para alocar tarefas de usuários com deficit de poder computacional segue as regras a seguir:

i) Alocar primeiro a tarefa com menor demanda.
ii) Alocar essa tarefa ao nó com o menor poder computacional entre aqueles em posse do usuário a ter uma tarefa interrompida.

Esse procedimento é repetido enquanto as equações 5 e 6 forem satisfeitas ou todas as tarefas forem alocadas. A partir dessas premissas o Decision Algorithm heterogêneo pode ser delineado na Figura 2.

Heterogeneous Decision Algorithm (int p)
1. *if* $(p > 0 \ \wedge \ (\exists \ u_\alpha \mid PowerDiff_\alpha = min(PowerDiff_1, ..., PowerDiff_m) \ \wedge PowerDiff_\alpha < 0))$
2. { *if* $(\exists \ u_\beta \mid PowerDiff_\beta = max(PowerDiff_1, ..., PowerDiff_m) \wedge PowerDiff_\beta > 0)$
3. { if $(\exists \ job \ j \ \in u_\beta \mid \omega_j = max(\omega_1, ..., \omega_l) \ \wedge \ (s_j = Active) \ \wedge \ (\omega_j > PowerDiff_\alpha))$
4. { Preempt *job(j)* from user u_β;
5. Allocate resource to the job k submitted by u_α;
6. Decrease p;
7. Update the ClassAds of users u_α and u_β;
8. }
9. Call ***Heterogeneous Decision Algorithm(p)***;
10. }
11. }

Figura 2. Decision Algorithm para o HOSEP

A viabilidade e eficácia desse algoritmo pode ser verificada a partir das seguintes características:

- Como $\sum Alloc\_Resources = \sum Shared\_Resources$ se todos os recursos estiverem alocados, é verificável que é impossível ter $PowerDiff < 0$ para todos os usuários se algum usuário tiver $PowerDiff < 0$; ou, em outras palavras, é impossível ter todos os provedores de recursos com menos recursos do que forneceram se a soma dos recursos oferecidos for igual ao total de recursos em uso.

- Se um usuário U_i, submetendo um conjunto de tarefas para a grade, ocupar menos poder computacional do que forneceu, então receberá um novo nó, reduzindo sua penalidade, a menos que todos os demais usuários U_j tenham diferença de poder restante $RP_j \leq \|PowerDiff_i\|$.
- O usuário U_β que sofrerá a preempção é aquele com o maior ω_k entre todos, evitando a preempção de um usuário com menos recursos do que sua porção.
- O nó a ser interrompido é aquele com menor poder computacional, minimizando o efeito da preempção no valor de $PowerDiff_\beta$.
- Depois da preempção o poder restante do usuário U_β ou continuará positivo (tendo mais poder do que o que forneceu) ou passará a zero ou negativo (deixando de ser elegível para preempções).

3.4. Prova de justiça de propriedade

Como HOSEP busca garantir que os provedores de recursos na grade terão aproximadamente o mesmo poder computacional que oferecerão, é preciso verificar se a justiça de propriedade é obtida na forma apresentada pela Definição 4. Essa verificação é delineada a seguir.

Primeiro, considere-se que um dado usuário requisita recursos num volume de D MFlops de poder computacional, numa grade que tenha A MFlops de poder disponível, dos quais P MFlops foram fornecidos por esse usuário. Com isso três cenários devem ser considerados:

1. Se $D \leq A$, o usuário receberá todos os nós necessários e a justiça está garantida pois nenhuma tarefa será postergada.
2. Se $D > A$ e $A \geq P$, o usuário receberá todos os recursos disponíveis pelo processo de alocação e a justiça está garantida pois o usuário receberá pelo menos o mesmo poder que forneceu.
3. Se $D > P$ e $P > A$, o usuário receberá todos os recursos disponíveis, mas ainda menos do que forneceu ($PowerDiff < 0$), violando a justiça de propriedade e demandando a aplicação da política HOSEP.

   O terceiro cenário implica em alocar recursos até que a justiça seja atingida.

Assuma-se então que a justiça não seja atingida apesar de existirem condições para isso. Isso significa que as duas condições da Definição 4 são falsas e que, ao mesmo tempo, existam usuários com uma quantidade suficiente de recursos além de suas porções. Dado que a condição *i)* é obviamente falsa, é preciso apenas verificar que a condição *ii)* é também falsa. As asserções a seguir buscam verificar isso.

(i) Assuma que exista um usuário U_{less}, com menos recursos que forneceu ($PowerDiff_{less} < 0$), e que ao menos um usuário u_{more}, que tenha ($PowerDiff_{more} \gg 0$)[†];

(ii) U_{less} não receberá mais recursos se:
   (a) regra na linha 2 do Decision Algorithm não puder ser aplicada, isto é, não existir usuário com $PowerDiff > 0$, que é falso para pelo menos u_{more}; ou

[†]Assuma-se que "$\gg 0$" significa que *PowerDiff* continuará maior que 0 se o nó menos potente for tirado de u_{more}

(b) regra na linha 3 não puder ser aplicada, isto é, não existir usuário com recursos suficientes para sofrer preempção;

(iii) Se isso for verdade, então $\omega_i < PowerDiff_{less}$ para todas as tarefas executando na grade;

(iv) Entretanto, como u_{more} tem $PowerDiff_{more} \gg 0$, é esperado que ele possua pelo menos um nó k tal que $\omega_k > 0$, contradizendo a afirmação anterior.

Portanto, o processo de alocação não para antes de todos os usuários terem recursos acima da condição de justiça.

4. Testes

Para verificar o desempenho do HOSEP o mesmo foi aplicado, por simulação, a vários cenários, sendo os resultados comparados com os do OSEP original. As simulações foram realizadas usando o iSPD [Manacero *et al.* 2012], escolhido em virtude do seu desempenho e das facilidades que apresenta para implementar escalonadores, calcular métricas e criar modelos por meio de sua interface icônica.

Aqui serão apresentados os resultados para uma grade de 12 nós, pertencentes a quatro usuários, com porções diferentes entre si. Essa grade é vista na Figura 3, enquanto os dados das máquinas aparecem na Tabela 1. Cada usuário oferece o mesmo número de máquinas para verificar se, de fato, o HOSEP respeita o desempenho dos nós das porções dos usuários e não o número de máquinas como faz o OSEP. Além disso, as porcentagens do desempenho total atribuídas a cada usuário são tais que cada porção é marcadamente diferente das outras, incluindo uma com alto desempenho (User1) e uma com baixo desempenho (User4).

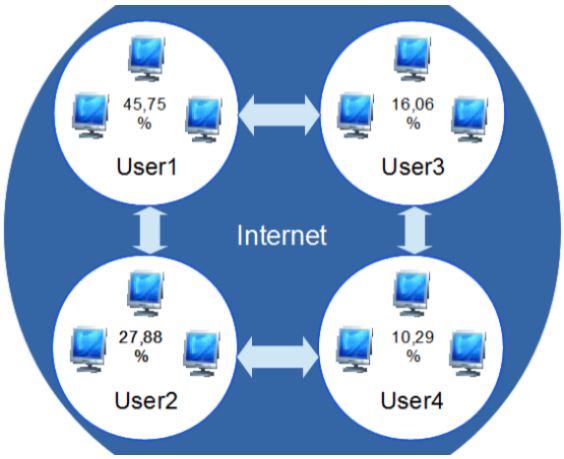

Figura 3. Modelo de grade com quatro provedores/usuários de recursos.

Para os testes se considerou três diferentes conjuntos de tarefas, com cada usuário submetendo dez tarefas. Cada conjunto de tarefas submetido por um usuário tem sua demanda categorizada como baixa, média e alta, de acordo com a mistura de tarefas presentes no conjunto. Em cada conjunto existiam tarefas pequenas, que têm tempo de execução, na melhor máquina da grade, entre 5 a 30 minutos. Também existiam tarefas médias, consomindo entre 30 e 80 minutos, e tarefas grandes, consumindo entre 80 e 240 minutos.

Tabela 1. Dados dos nós da grade

Usuário	Máquinas	% de poder computacional
User1	3 nós de 132250 MFlops	45,76
User2	1 nó de 132250 MFlops 2 nós de 54760 MFlops	27,88
User3	1 nó de 29750 MFlops 2 nós de 54760 MFlops	16,6
User4	3 nós de 29750 MFlops	10,29

Os conjuntos de demanda baixa eram compostos, somando-se todos os usuários, por 24 tarefas pequenas, 12 médias e 4 grandes. Os conjuntos com demanda média tinham 12 pequenas, 24 médias e 4 grandes, enquanto os conjuntos com demanda alta tinham 4 tarefas pequenas, 12 médias e 24 grandes. Essas tarefas vinham em quantidades iguais de cada usuário foram definidas de forma que a grade fique totalmente ocupada antes do User1 submeter, e não submeter um número de tarefas que seja desproporcionalmente maior do que o número de nós no sistema. Os tamanhos das tarefas foram definidos levando em conta que segundo [Di *et al.* 2013], em um ambiente real da Google, 1,5% das tarefas é responsável por 98,5% do tempo total de execução do sistema.

Os gráficos de satisfação do usuário apresentados na Figura 4 foram obtidos considerando-se que o usuário $User1$ submete suas tarefas após os demais usuários. Isso permite verificar se o HOSEP consegue, de fato, garantir que o usuário obtenha recursos equivalentes à sua porção numa situação em que não existem nós livres. Deve ser observado ainda que os testes envolveram também a análise do algoritmo com e sem a aplicação de *checkpoints*.

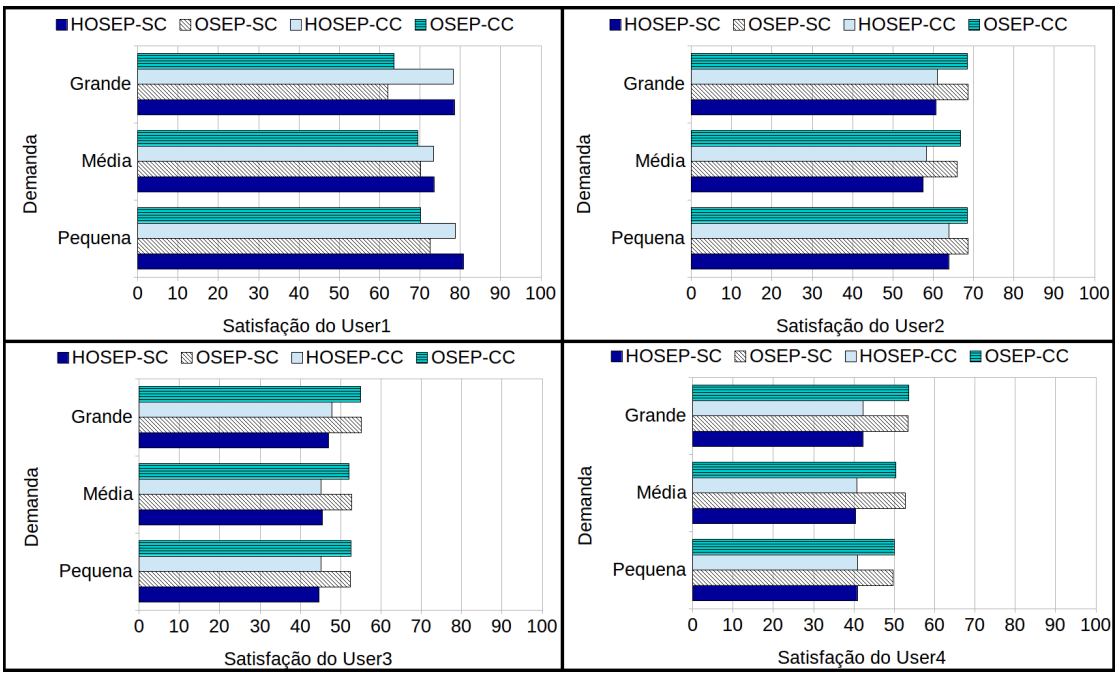

Figura 4. Gráficos de satisfação

Pelos gráficos, nota-se que o OSEP falha em atender ao critério de justiça por entregar menos poder computacional ao **user1**. A satisfação desse usuário é equivalente ao **user2**, que oferece apenas 60% do poder computacional que **user1** oferece. Além disso, sua satisfação é pouco maior do que a de **user4**, que oferece apenas 10% do poder da grade. Já com HOSEP se verifica que *user1* é sempre o usuário mais satisfeito, com cerca de 80% de satisfação, além de que os demais usuários têm sua satisfação proporcional ao poder computacional que oferecem. Observe-se ainda que a diferença de satisfação aumenta com o aumento no tamanho das tarefas, o que é esperado pois para tarefas grandes o custo de atrasos é maior.

5. Conclusões

A proposta de uma abordagem baseada na justiça de propriedade para sistemas heterogêneos busca estimular o compartilhamento de recursos mais poderosos. Dos resultados apresentados na seção anterior é possível identificar que o HOSEP consegue garantir satisfação proporcional ao poder computacional dos recursos disponibilizados. Isso é verdade mesmo quando o usuário, ao submeter suas tarefas, encontra a grade já ocupada por outros usuários. Isso indica que a adoção do HOSEP como política de escalonamento numa grade heterogênea tende a estimular uma maior oferta de recursos por parte dos membros da grade.

Do ponto de vista operacional, o algoritmo consegue minimizar a defasagem entre o poder oferecido e o obtido. Isso é obtido com a utilização do parâmetro *Poder Remanescente* para definir critérios para preempção de tarefas já em execução. Isso é importante em grades heterogêneas, quando não basta simplesmente contar o número de nós alocados, como faz o OSEP.

Uma vez que o HOSEP se aplica a grades federadas, é interessante que se identifique abordagens equivalentes para outros modelos de grades ou mesmo *clouds*, permitindo formas alternativas de cobrança pelo uso do sistema. Além disso, como o consumo de energia, principalmente em ambientes complexos, se tornou uma preocupação central em centros de serviço, é possível vislumbrar uma abordagem híbrida entre justiça de propriedade e consumo de energia, em que usuários poderiam ter mais prioridade se consumissem menos energia.

Referências

Amar, L., Barak, A., Levy, E., and Okun, M. (2007). An on-line algorithm for fair-share node allocations in a cluster. In *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 83–91, Washington, DC, USA. IEEE Computer Society.

Andrade, N., Brasileiro, F., Cirne, W., and Mowbray, M. (2004). Discouraging free riding in a peer-to-peer cpu-sharing grid. In *Proc. of the 13th IEEE International Symposium on High Performance Distributed Computing*, HPDC '04, pages 129–137. IEEE Computer Society.

Arpaci-Dusseau, A. C. and Culler, D. E. (1997). Extending proportional-share scheduling to a network of workstations. In *In Proceedings of Parallel and Distributed Processing Techniques and Applications (PDPTA'97), Las Vegas, NV*.

Di, S., Kondo, D., and Cappello, F. (2013). Characterizing cloud applications on a google data center. In *2013 42nd International Conference on Parallel Processing*, pages 468–473. IEEE.

Doulamis, N., Doulamis, A., Varvarigos, E., and Varvarigou, T. (2007). Fair scheduling algorithms in grids. *Parallel and Distributed Systems, IEEE Transactions on*, 18(11):1630–1648.

Falavinha Jr., J., Manacero Jr., A., Livny, M., and Bradley, D. (2009). The ownershare scheduler for a distributed system. In *Proc. of 38th Intl. Conf. on Parallel Processing Workshops*, pages 298–305. IEEE.

Fujimoto, N. and Hagihara, K. (2004). A comparison among grid scheduling algorithms for independent coarse-grained tasks. In *Proceedings of the 2004 Symposium on Applications and the Internet-Workshops (SAINT 2004 Workshops)*, SAINT-W '04, pages 674–, Washington, DC, USA. IEEE Computer Society.

Hadoop Project (2016a). *Capacity Scheduler Guide*. http://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html - acessado em 06/08/2016.

Hadoop Project (2016b). *Fair Scheduler*. http://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/FairScheduler.html - acessado em 06/08/2016.

Kay, J. and Lauder, P. (1988). A fair share scheduler. *Commun. ACM*, 31(1):44–55.

Livny, M. and Raman, R. (1998). High-throughput resource management. In Foster, I. and Kesselman, C., editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann.

Manacero, A., Lobato, R. S., Oliveira, P. H. M. A., Garcia, M. A. B. A., Guerra, A. I., Aoqui, V., Menezes, D., and da Silva, D. T. (2012). iSPD: an iconic-based modeling simulator for distributed grids. In *Proceedings of the 45th Annual Simulation Symposium*, ANSS '12, pages 5:1–5:8, San Diego, CA, USA. Society for Computer Simulation International.

Qureshi, M. B., Dehnavi, M. M., Min-Allah, N., Qureshi, M. S., Hussain, H., Rentifis, I., Tziritas, N., Loukopoulos, T., Khan, S. U., Xu, C.-Z., and Zomaya, A. Y. (2014). Survey on grid resource allocation mechanisms. *Journal of Grid Computing*, 12(2):399–441.

Thain, D., Tannenbaum, T., and Livny, M. (2005). Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356.

Tomas, L., Ostberg, P., Caminero, B., Carrion, C., and Elmroth, E. (2012). 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*, pages 226–233.