

ESCALONAMENTO DE TAREFAS EM SISTEMAS DISTRIBUÍDOS BASEADO NO CONCEITO DE PROPRIEDADE DISTRIBUÍDA

JOSÉ NELSON FALAVINHA JUNIOR

Ilha Solteira
Estado de São Paulo – Brasil
Junho de 2009

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

“Escalonamento de Tarefas em Sistemas Distribuídos
baseado no Conceito de Propriedade Distribuída”

JOSÉ NELSON FALAVINHA JUNIOR

Orientador: Prof. Dr. Alcardo Manacero Júnior

Co-orientador: Prof. Dr. Miron Livny

Tese apresentada à Faculdade de Engenharia - UNESP – Campus de Ilha Solteira, para obtenção do título de Doutor em Engenharia Elétrica. Área de Conhecimento: Automação.

Ilha Solteira – SP
Junho de 2009

FICHA CATALOGRÁFICA

Elaborada pela Seção Técnica de Aquisição e Tratamento da Informação/Serviço Técnico de Biblioteca e Documentação da UNESP-Ilha Solteira

F177e Falavinha Junior, José Nelson.
Escalonamento de tarefas em sistemas distribuídos baseado no conceito de propriedade distribuída / José Nelson Falavinha Junior. -- Ilha Solteira : [s.n.], 2009
103 p. : il., color.

Tese (doutorado) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira, 2009

Orientador: Aleardo Manacero Júnior
Co-orientador: Miron Livny

Bibliografia: p. 98-103

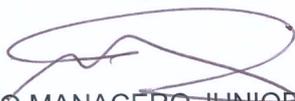
1. Escalonamento de tarefas em sistemas distribuídos. 2. Escalonamento baseado no conceito de propriedades distribuída. 3. Propriedade de recursos em sistemas distribuídos.

CERTIFICADO DE APROVAÇÃO

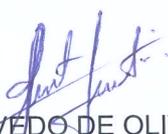
TÍTULO: Escalonamento de Tarefas em Sistemas Distribuídos baseado no Conceito de Propriedade Distribuída

AUTOR: JOSÉ NELSON FALAVINHA JUNIOR
ORIENTADOR: Prof. Dr. ALEARDO MANACERO JUNIOR

Aprovado como parte das exigências para obtenção do Título de DOUTOR em ENGENHARIA ELÉTRICA, Área: AUTOMAÇÃO, pela Comissão Examinadora:



Prof. Dr. ALEARDO MANACERO JUNIOR
Departamento de Cienc Comp e Estatística / Instituto de Biociências, Letras e Ciências Exatas de São José do Rio Preto



Prof. Dr. SERGIO AZEVEDO DE OLIVEIRA
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira



Profa. Dra. RENATA SPOLON LOBATO
Departamento de Cienc Comp e Estatística / Instituto de Biociências, Letras e Ciências Exatas de São José do Rio Preto



Prof. Dr. ALFREDO GOLDMAN VEL LEJBMAN
Departamento de Ciência da Computação / Universidade de São Paulo



Prof. Dr. HENRIQUE MONGELLI
Departamento de Computação e Estatística / Universidade Federal de Mato Grosso do Sul

Data da realização: 25 de maio de 2009.

Dedicatória

Dedico este trabalho e todo meu esforço a toda minha família, em especial a minha querida e amada avó Alzira.

Agradecimentos

Agradeço a todos do departamento de Ciência da Computação e Estatística da Unesp de São José do Rio Preto e do Programa de Pós-Graduação em Engenharia Elétrica da Unesp de Ilha Solteira pela qualidade de ensino, dedicação e incentivo à carreira acadêmica.

Agradecimento especial ao Prof. Dr. Aleardo Manacero Júnior, pela extrema dedicação tanto na orientação acadêmica como na orientação profissional e pessoal.

Agradeço a Capes pelo apoio e suporte financeiro durante todo doutorado, incluindo também o financiamento do estágio de doutorado no exterior pelo período de um ano.

Agradeço também ao grupo de pesquisa Condor da Universidade de Wisconsin em Madison nos EUA, em especial ao Prof. Dr. Miron Livny pela recepção, dedicação e orientação durante o meu estágio de doutorado no exterior.

E por fim, agradeço à toda minha família e amigos, meus pais, meu irmão e minha noiva Daiana, pessoas essenciais na minha vida que me ajudaram e apoiaram muito durante essa caminhada.

Obrigado.

Resumo

Em sistemas distribuídos de larga escala, onde os recursos compartilhados são de propriedade de entidades distintas, existe a necessidade de refletir o fator propriedade dos recursos no processo de escalonamento de tarefas e alocação de recursos. Um sistema de gerenciamento de recursos apropriado deve garantir que os proprietários de recursos tenham acesso aos seus recursos ou ao menos a uma parcela de recursos que seja equivalente a eles. Diferentes políticas podem ser estabelecidas para que o sistema garanta esse direito aos proprietários de recursos, e nessa tese defende-se uma política de escalonamento e alocação de recursos chamada *Owner-Share Enforcement Policy* (OSEP) ou **Política de Garantia da Porção do Proprietário**, que tem por objetivo garantir o direito de acesso aos recursos através de um sistema de escalonamento baseado em preempção de tarefas e realocação de recursos. Avalia-se a política através da análise de testes e resultados envolvendo métricas de desempenho que descrevem fatores como violação da política, perda da capacidade de processamento, custo da política e satisfação do usuário. Os testes ainda envolveram a análise de desempenho da política em ambientes com a possibilidade de *checkpointing* de tarefas, minimizando assim o desperdício de processamento. Fez-se ainda comparações com a política de compartilhamento justo *Fair-Share*, que permitiram estabelecer as vantagens e desvantagens de cada política e ainda identificar futuros problemas. Por fim, conclui-se a tese identificando as contribuições oferecidas por este trabalho e os trabalhos futuros que podem ser desenvolvidos.

Abstract

In large distributed systems, where shared resources are owned by distinct entities, there is a need to reflect resource ownership in resource allocation. An appropriate resource management system should guarantee that owners of resources have access to their resources or at least to a share of resources proportional to the share of resources they provide. Different policies can be established for guaranteeing the access to resources, and in this thesis we introduce a policy for scheduling and resource allocation named Owner Share Enforcement Policy (OSEP). This policy is based on the concept of distributed ownership and it guarantees the owner's right of accessing their share of resources in a distributed system with a preemptive share space. We evaluate this policy through tests and results analysis involving performance metrics that describe policy violation, loss of capacity, policy cost and user satisfaction. The tests were also conducted in environments with and without job checkpointing, and comparisons with the Fair-Share scheduling policy were made in order to capture the trade-offs of each policy. Finally, we conclude the thesis describing the contributions achieved with this work and pointing directions for future work.

Lista de Figuras

1	Modelo de grade computacional (SILVA, 2003).....	p. 19
2	Arquitetura geral de uma grade (FOSTER; KESSELMAN, 2003).	p. 22
3	<i>Middleware - software</i> de gerenciamento (EGEE PROJECT, 2009)	p. 28
4	Interações de uma grade (FOSTER; KESSELMAN, 2003).	p. 29
5	Relacionamento entre cliente, escalonador e recursos.	p. 30
6	Fases do escalonamento (SCHOPF, 2002).	p. 31
7	Ilustração de um cenário de escalonamento distribuído.	p. 32
8	Algoritmo da política OSEP dividido em duas partes.....	p. 43
9	Fluxograma e diagrama de objetos do Algoritmo Dinâmico de Atualização	p. 45
10	Fluxograma e diagrama de objetos do Algoritmo de Decisão	p. 47
11	Diagrama do funcionamento do algoritmo da política Owner-Share	p. 48
12	Comportamento ideal para execução de tarefas.....	p. 51
13	Comportamento ideal para a política OSEP.....	p. 54
14	Comportamento real para a política OSEP	p. 55
15	Área PV da violação da política OSEP.	p. 56
16	Área LC da perda de capacidade.	p. 57
17	Dois usuários e dez máquinas - Variação do parâmetro de atuação.	p. 63
18	Dois usuários e vinte máquinas - Variação do parâmetro de atuação.	p. 64
19	Dois usuários e quarenta máquinas - Variação do parâmetro de atuação..	p. 65
20	Demanda menor que a porção - 5 tarefas por usuário.....	p. 69
21	Demanda igual à porção - 10 tarefas por usuário.	p. 70

22	Demanda maior que a porção - 20 Tarefas por Usuário.	p.72
23	Demanda maior que a porção - 30 tarefas por usuário.	p.74
24	Usuários com demandas diferentes.	p.75
25	Tarefas de curta duração - 600 s.	p.77
26	Comparação da satisfação dos usuários - Tarefas de curta duração.	p.78
27	Tarefas de média duração - 1.800 s.	p.79
28	Tarefas de longa duração - 3.600 s.	p.80
29	Tarefas de duração variada - Entre 600 e 3.600 s.	p.81
30	Tarefas de duração variada - Cenário com tarefas maiores primeiro.	p.82
31	Comparação da satisfação dos usuários - Tarefas de duração variada.	p.83
32	Tarefas de duração variada - Cenário com tarefas menores primeiro.	p.84
33	Testes com 4 usuários com porções iguais - Tarefas iguais.	p.85
34	Satisfação dos usuários - Porções iguais e tarefas iguais.	p.86
35	Testes com 4 Usuários com porções iguais - Tarefas diferentes.	p.87
36	Satisfação dos usuários - Porções iguais e tarefas diferentes.	p.88
37	Testes com 4 usuários com porções diferentes - Tarefas iguais.	p.90
38	Testes com 4 usuários com porções diferentes - Tarefas diferentes.	p.91
39	Satisfação dos usuários - Porções diferentes e tarefas diferentes.	p.92

Lista de Tabelas

1	Resultados das métricas de desempenho para 10 máquinas.	p.66
2	Resultados das métricas de desempenho para 20 máquinas.	p.66
3	Resultados das métricas de desempenho para 40 máquinas.	p.66
4	Resultados dos testes de demanda menor que a porção - 5 tarefas.	p.69
5	Resultados dos testes de demanda igual à porção - 10 tarefas	p.70
6	Resultados dos testes de demanda maior que porção - 20 tarefas.....	p.72
7	Resultados dos testes de demanda maior que porção - 30 tarefas.....	p.73
8	Resultados dos testes entre os usuários com demandas diferentes.	p.75
9	Resultados dos testes com tarefas de curta duração	p.77
10	Resultados dos testes com tarefas de média duração.....	p.78
11	Resultados dos testes com tarefas de longa duração.	p.80
12	Resultados dos testes com tarefas de duração variada.	p.81
13	Resultados dos testes com tarefas de duração variada - Cenário com tarefas maiores primeiro.....	p.82
14	Resultados dos testes com tarefas de duração variada - Cenário com tarefas menores primeiro.	p.83
15	Resultados dos testes com maior quantidade de usuários - Porções iguais e tarefas iguais.....	p.85
16	Resultados dos testes com maior quantidade de usuários - Porções iguais e tarefas diferentes.	p.87
17	Resultados dos testes com maior quantidade de usuários - Porções dife- rentes e tarefas iguais.	p.89

18	Resultados dos testes com maior quantidade de usuários - Porções diferentes e tarefas diferentes.....	p.91
19	Desvio Padrão das satisfações dos usuários para as políticas OSEP e <i>Fair-Share</i>	p.93

Sumário

1	Introdução	p. 14
1.1	Motivação	p. 14
1.2	Estado da Arte	p. 15
1.3	Objetivos	p. 16
1.4	Organização da Tese	p. 17
2	Grades Computacionais e o Escalonamento	p. 18
2.1	Grades Computacionais	p. 18
2.1.1	Tipos de Grades	p. 20
2.1.2	Arquitetura de uma grade	p. 22
2.2	<i>Middlewares</i> para Grades	p. 24
2.2.1	Globus	p. 24
2.2.2	Legion	p. 24
2.2.3	Condor e Condor-G	p. 25
2.2.4	gLite	p. 26
2.2.5	OGSA-DAI	p. 26
2.2.6	OurGrid	p. 27
2.2.7	Nimrod-G	p. 27
2.3	Gerenciamento de Recursos e Serviços	p. 28
2.3.1	Escalonamento em Grades	p. 30
2.4	Algoritmos de Escalonamento	p. 33
2.4.1	<i>Dynamic</i> FPLTF	p. 33

2.4.2	<i>Workqueue with Replication</i>	p.35
2.4.3	<i>XShare</i>	p.36
2.4.4	<i>Storage Affinity</i>	p.37
2.4.5	<i>Multiple Queue - Hierarchy</i>	p.37
2.4.6	<i>Fair-Share Scheduler</i>	p.38
2.4.7	Rede de Favores	p.39
2.5	Considerações Finais	p.39
3	Escalonamento <i>Owner-Share</i>	p.40
3.1	Definições Fundamentais	p.40
3.1.1	Propriedade Distribuída e Compartilhamento Distribuído	p.40
3.1.2	Definição de Porção ou <i>Share</i>	p.40
3.1.3	Definição de Propriedade Distribuída	p.41
3.2	Política <i>Owner-Share</i>	p.41
3.2.1	Definições do <i>Owner-Share</i>	p.42
3.2.2	Algoritmo da Política <i>Owner-Share</i>	p.43
3.2.2.1	Algoritmo Dinâmico de Atualização	p.44
3.2.2.2	Algoritmo de Decisão	p.45
3.2.2.3	Validação do Algoritmo	p.48
3.3	Métricas de Desempenho	p.50
3.3.1	Comportamento Ideal Esperado pelo Usuário	p.50
3.3.2	Satisfação do Usuário	p.51
3.3.3	Caso Ótimo para a OSEP	p.52
3.3.4	Violação de Política	p.54
3.3.5	Perda de Capacidade	p.55
3.3.6	Custo da Política	p.56
3.4	<i>Checkpointing</i>	p.57

3.5	Implementação do OSEP num Caso Real	p. 58
3.6	Considerações Finais	p. 59
4	Testes e Resultados	p. 60
4.1	Ambiente de Testes	p. 60
4.2	Ajuste do Algoritmo	p. 61
4.2.1	Estudo de Caso 1: Ajuste do Parâmetro de Atuação	p. 61
4.3	Variação da Demanda dos Usuários	p. 68
4.3.1	Estudo de Caso 2: Demanda Menor ou Igual à Porção	p. 68
4.3.2	Estudo de Caso 3: Demanda Maior que a Porção	p. 71
4.3.3	Estudo de Caso 4: Usuários com Demandas Diferentes	p. 74
4.4	Variação no Tamanho das Tarefas	p. 76
4.4.1	Estudo de Caso 5: Tarefas de Curta Duração	p. 76
4.4.2	Estudo de Caso 6: Tarefas de Média Duração	p. 78
4.4.3	Estudo de Caso 7: Tarefas de Longa Duração	p. 79
4.4.4	Estudo de Caso 8: Tarefas de Duração Variada	p. 80
4.5	Testes com Maior Quantidade de Usuários	p. 83
4.5.1	Estudo de Caso 9: 4 usuários e porções iguais	p. 84
4.5.2	Estudo de Caso 10: 4 usuários e porções diferentes	p. 88
4.6	Análise Global dos Resultados	p. 91
4.7	Considerações Finais	p. 94
5	Conclusões e Trabalhos Futuros	p. 95
5.1	Conclusões	p. 95
5.2	Trabalhos Futuros	p. 96
	Referências	p. 98

1 *Introdução*

Este capítulo descreve o problema tratado nessa tese e os diferentes caminhos e pontos de vista para sua solução. Ainda nesse capítulo apresenta-se uma visão geral do estado da arte com as soluções existentes e a solução proposta. Descreve-se então os objetivos que motivaram a tese e a última seção contém a organização dos capítulos restantes.

1.1 *Motivação*

A relação custo/desempenho oferecida por sistemas distribuídos de grande porte, como grades computacionais, é a principal razão pela qual essa infraestrutura tornou-se um paradigma computacional para solução de aplicações de grande desafio na ciência, engenharia, economia e diversas outras áreas. Usuários de todos os tipos e de diversos lugares querem compartilhar seus recursos a fim de criar um ambiente computacional em que eles possam também utilizar os recursos pertencentes a outros usuários. O aumento do poder de processamento com baixo custo caracteriza assim a computação em grade, que é a utilização de recursos distribuídos através de diversos domínios administrativos com serviços de qualidade, como definido por (FOSTER, 2002) e (MORENO, 2004).

A união de recursos originados de diferentes domínios administrativos e pertencentes a diferentes entidades ou usuários caracteriza por fim o conceito de propriedade distribuída. Uma vez que a infraestrutura computacional é criada, o objetivo comum é compartilhar os recursos e fazer o melhor uso possível da infraestrutura. No entanto, existem termos e condições estabelecidas entre os usuários proprietários dos recursos, como por exemplo políticas de acesso e de escalonamento, determinando como a infraestrutura deve ser utilizada.

Diferentes formas de atuação podem ser definidas. Na maioria dos casos o acordo entre os usuários determina que os recursos devem ser utilizados através de um compartilhamento e escalonamento justo entre os usuários, mas definir o que é justo é algo

complexo e depende das necessidades dos proprietários dos recursos do sistema. Nessa tese, tratou-se o caso em que a justiça é determinada através do fator de propriedade dos recursos, um requerimento importante a ser considerado pois os usuários têm o direito de acessar seus recursos, ou uma porção correspondente aos seus recursos, quando desejarem.

Um fator importante a ser destacado é que nessa tese posicionou-se nosso foco em um ambiente computacional em que as tarefas são do tipo *Bag-of-Tasks*. Assim como foi mencionado por (ANDRADE, 2004), essas tarefas são independentes e especialmente adequadas para execução em grades computacionais onde a largura de banda entre os componentes do sistema pode ser limitada.

1.2 Estado da Arte

O direito de acesso aos recursos estabelecido pelos usuários através do conceito de propriedade distribuída permite a definição de dois modelos de compartilhamento de recursos. No primeiro modelo os recursos dos usuários são identificados fisicamente. Assim, quando o usuário requisita seus recursos ele deve receber exatamente os recursos que ele forneceu para o sistema. Já a segunda forma de compartilhamento é definida sobre o conceito de porção equivalente ou *share*. Assim, os recursos formam um conjunto único representado pelas porções de cada proprietário. Nesse segundo modelo, o usuário requisita recursos e pode receber quaisquer recursos do sistema desde que o montante seja equivalente à sua porção.

Essa tese define o sistema *Owner-Share* para escalonamento de tarefas baseado no segundo modelo de compartilhamento, e levando em consideração o direito que os usuários têm de acessar sua porção de recursos assim que necessário e o mais rápido possível. Outras formas de escalonamento podem ser estabelecidas sobre diferentes pontos de vista, como o algoritmo de escalonamento *Fair-Share* ou Compartilhamento Justo, definido em (KAY; LAUDER, 1988), que utiliza informações sobre o histórico de utilização dos recursos do sistema para cada usuário. Os usuários são então priorizados de acordo com a contabilização do seu histórico de utilização, que determina quantos recursos este usuário poderá usar no futuro.

Essas duas políticas de escalonamento, *Owner-Share* e *Fair-Share*, são caracterizadas pela presença de um escalonador centralizado, sendo que a primeira busca por uma solução imediata através da preempção de tarefas e a segunda trabalha por uma solução em longo prazo sem a interrupção de tarefas. Cada uma dessas políticas tem seu objetivo, vantagens

e desvantagens, e uma comparação entre elas é de fato interessante. Na seção de testes apresenta-se uma análise mais detalhada sobre o escalonamento *Owner-Share*, descrevendo suas vantagens, desvantagens e fazendo comparações com o escalonamento *Fair-Share*.

O trabalho realizado e descrito em (ANDRADE; BRASILEIRO; MOWBRAY, 2004) e em (ANDRADE et al., 2004), define o compartilhamento de um ambiente computacional de grade como uma rede de troca de favores na qual o controle da alocação de recursos é descentralizado e tratado par a par entre os *sites* de uma grade. O histórico de utilização de cada *site* é contabilizado e o mesmo é compensado de acordo com a sua contribuição para outros *sites*. A contribuição, também chamada de doação, é calculada proporcionalmente à quantidade de recursos trocada entre cada par de *sites* do grade ao longo do tempo. Essa política incentiva o compartilhamento de recursos e a criação de grades computacionais porém, assim como a *Fair-Share*, ela também não se preocupa com o direito de propriedade do usuário de acessar seus recursos imediatamente quando necessário.

Independentemente da justificativa de cada modelo de compartilhamento ou política de escalonamento, o objetivo comum é incentivar a computação em grade e proporcionar aos seus usuários um ambiente de alto poder de processamento a baixo custo. A definição de políticas para negociação do acesso aos recursos é inevitável e complexa considerando que cada *site* e seus usuários podem ter interesses diferentes.

1.3 Objetivos

O objetivo desse trabalho é apresentar uma solução de escalonamento para sistemas distribuídos que preserve o direito de propriedade dos usuários, os quais forneceram recursos para a composição do sistema. Do nosso ponto de vista, a preservação do direito de propriedade é um incentivo à união e ao compartilhamento de recursos, pois os envolvidos terão certeza de que no mínimo uma parcela da capacidade de processamento equivalente à sua contribuição estará garantida para sua própria utilização.

Assim, apresenta-se a Política de Garantia da Porção do Proprietário ou OSEP (*Owner-Share Enforcement Policy*). Faz-se depois a análise da política proposta a fim de determinar os pontos fracos e fortes dessa solução, assim como o custo de preservar o direito de propriedade dos usuários. Apesar de terem objetivos diferentes, comparações com a política *Fair-Share* são também necessárias pois ajudam a definir e deixar claras as vantagens e desvantagens de cada abordagem.

1.4 Organização da Tese

No capítulo seguinte apresenta-se todo o embasamento teórico necessário para a familiarização com o ambiente computacional de grade e suas principais características. Descreve-se detalhadamente toda sua arquitetura, modelo de funcionamento e ferramentas que permitem sua construção. Faz-se ainda um apanhado dos algoritmos de escalonamento de tarefas e políticas para alocação de recursos em sistemas distribuídos, com a finalidade de capturar o conhecimento inserido nessas soluções, determinar problemas ainda não resolvidos e buscar novas soluções.

O capítulo 3 apresenta a política de escalonamento *Owner-Share*, contextualizando todo o trabalho desenvolvido para o alcance da solução proposta. Conceitos como o de propriedade distribuída e as métricas de desempenho usadas na análise da política são também definidos neste capítulo. O capítulo termina com a descrição de um método muito utilizado com a interrupção de tarefas, o *checkpointing*. Ele representa uma ferramenta de grande valia para o aumento da eficiência da solução proposta.

A avaliação, análise, comparações, testes e resultados são apresentados no capítulo 4. Primeiramente são apresentados os testes feitos para a determinação da melhor configuração da política *Owner-Share*, uma vez que diferentes formas de atuação resultam em um custo maior ou menor para o sistema. Em seguida são apresentados os testes de comparação com a política *Fair-Share*, os resultados permitem clarear as abstrações sobre cada solução, perceber suas vantagens, desvantagens e determinar suas dificuldades. Todos os resultados foram analisados através da comparação das métricas de desempenho definidas no capítulo 3, como o custo da política, a satisfação do usuário, a violação da política e a perda ou desperdício da capacidade de processamento.

Por fim, o capítulo 5 apresenta as conclusões sobre a solução e os resultados da avaliação feita. Descreve então os trabalhos futuros possíveis a partir das contribuições oferecidas e do atual estado da arte.

2 *Grades Computacionais e o Escalonamento*

Infraestruturas computacionais distribuídas de larga escala, como grades computacionais, têm como objetivo proporcionar alto desempenho das aplicações e máxima utilização dos recursos, fator de grande relevância que leva a um aumento considerável do número de usuários para esses sistemas. Para uma melhor compreensão deste trabalho apresentam-se a seguir os conceitos relacionados à grades computacionais, as descrições sobre as ferramentas de software ou *middlewares* utilizadas para a implementação, gerenciamento e manutenção de grades, o detalhamento das atividades de gerenciamento de recursos e serviços em grades e os algoritmos de escalonamento de tarefas.

2.1 Grades Computacionais

Grades computacionais são uma realidade como paradigma computacional para solução de aplicações complexas na ciência, engenharia, economia, e diversas outras áreas. A implementação e manutenção de uma grade envolve gerenciamento eficiente de recursos heterogêneos, geograficamente distribuídos, e dinamicamente disponíveis, além da utilização de protocolos e interfaces com o intuito de fornecer qualidade de serviço. A definição de uma grade é concebida de acordo com três requisitos mínimos pré-definidos (FOSTER, 2002):

- **Coordenação de recursos distribuídos não sujeitos a um controle centralizado:** um grade computacional integra e coordena recursos e usuários que estão em locais diferentes, porém este controle não é centralizado.
- **Protocolos e interfaces padronizadas ou abertas:** um grade é construída através de interfaces e protocolos multifuncionais que atendem a todos os problemas fundamentais de processamento remoto, como autenticação, autorização, descoberta de

recursos, e acesso aos recursos. É importante que tais interfaces e protocolos sejam padronizados ou abertos, caso contrário, seria caracterizado um sistema específico.

- **Fornecimento de qualidade de serviços não-trivial:** fornecimento de qualidade de serviços relacionada a variáveis como tempo de resposta, vazão da rede, disponibilidade e segurança, permitindo assim a utilização dos recursos de forma coordenada, pois a utilização de uma combinação correta de recursos é significativamente melhor do que simplesmente a soma de todos eles.

Segundo tais definições, (FOSTER; KESSELMAN; TUECKE, 2001) descreve que uma grade computacional é basicamente uma coleção distribuída de computadores e recursos de armazenamento mantidos para suprir as necessidades de uma comunidade ou de uma organização virtual (VO) (FOSTER; KESSELMAN, 2003). Qualquer quantidade de usuários autorizados e pertencentes a uma comunidade que possui acesso a todos ou a alguns recursos da grade, pode submeter tarefas à grade e aguardar respostas (RANGANATHAN; FOSTER, 2002). O fornecimento de qualidade de serviços (QoS) é o principal indicador de qualidade de uma grade, e a correta utilização dos recursos, da rede e de todos os componentes do sistema são fatores que influem significativamente para um melhor resultado final. A Figura 2.1 apresenta o modelo de uma grade mostrando os seus componentes: computadores, rede de interligação, sistemas finais específicos, aplicações e servidores. Outros tipos de sistemas, recursos e aplicações também podem integrar uma grade (CIRNE; SANTOS NETO, 2005).

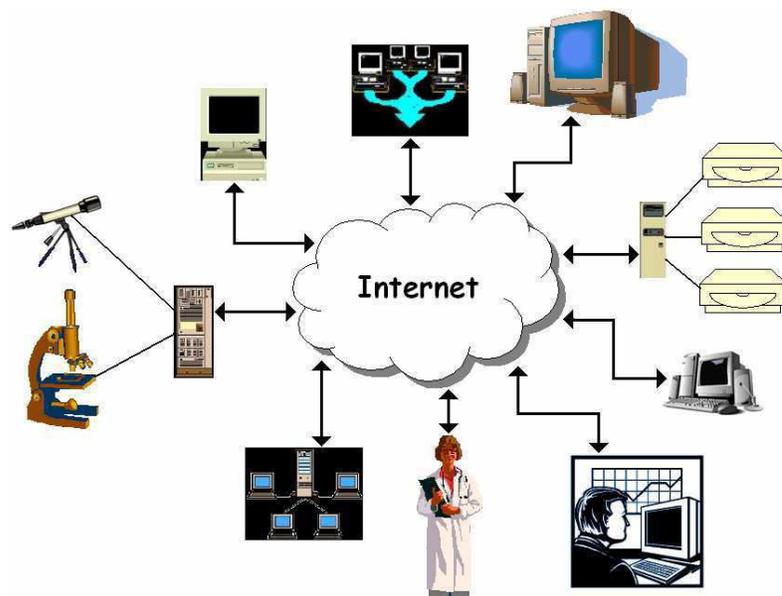


Figura 1: Modelo de grade computacional (SILVA, 2003).

A capacidade de aumentar o poder computacional e o armazenamento sob demanda sem a necessidade de investimentos extraordinários em infraestruturas de hardware é o que incentiva as pesquisas na área e o aparecimento cada vez maior de aplicações para grades (CHTEPEN; DHOEDT; VANROLLEGHEME, 2005).

Os usuários de uma grade computacional são inúmeros e das mais distintas áreas, como cientistas da computação, engenheiros, físicos, biólogos, grandes corporações, consumidores, nações e estados, enfim o mundo todo é um usuário potencial de grade. Com a existência de muitos tipos de usuários e aplicações de diversas áreas voltadas para a infraestrutura de grade, a fim de encontrar suas necessidades de computação e armazenamento, um ambiente de grade necessita de um gerenciamento eficiente de recursos e serviços que suporte todos esses usuários e aplicações. Sistemas de gerenciamento simples não são suficientemente eficientes para tal gerência, e novas tecnologias precisam ser desenvolvidas para que isso aconteça. No estado em que se encontra toda tecnologia de grades e sistemas distribuídos, vários tipos de aplicações são suportados mas o aparecimento de novos tipos de aplicações para esses ambientes é constante, assim torna-se necessário o desenvolvimento de novas tecnologias.

2.1.1 Tipos de Grades

As principais definições e explicações para diferentes tipos de grades computacionais são aqui apresentadas com o intuito de contextualizar a relação entre coordenação de recursos e os tipos de grades. Algumas delas são baseadas na estrutura de organização e outras nos recursos utilizados. A seguir é apresentada uma classificação das várias definições existentes (ABBAS, 2003).

- **Grades Departamentais:** grades departamentais são desenvolvidas para resolver problemas de um grupo particular em conjunto com uma instituição qualquer. Os recursos não são compartilhados por outros grupos dentro da instituição. Exemplos são os chamados *Cluster Grids* (SUN MICROSYSTEMS, 2002), termo usado pela Sun Microsystems que consiste em um ou mais sistemas trabalhando juntos com o objetivo de fornecer um único ponto de acesso para usuários. *Infra Grids* (ADAMS; IBM, 2002), termo usado pela IBM para definir uma grade que otimiza recursos em um empresa e não envolve nenhum outro parceiro interno, também fazem parte desta classificação.
- **Grades Empresariais:** grades empresariais consistem na coleção de recursos es-

palhados em uma empresa fornecendo serviços para todos os seus usuários. *Intra Grids*, caracterizados pelo compartilhamento de recursos entre grupos distintos em uma empresa internos ao *firewall*, e também *Campus Grids* que habilitam múltiplos projetos e departamentos para o compartilhamento de recursos de modo cooperativo, são exemplos desta classificação.

- **Grades Extra-empresariais:** grades estabelecidas entre companhias, seus parceiros de negócios e seus consumidores. Os recursos são geralmente disponibilizados através de uma rede virtual privada. *Extra Grids* e *Partner Grids* são exemplos deste modelo, o primeiro é caracterizado sobre o aspecto da conectividade entre os componentes da grade e o segundo sobre o compartilhamento de recursos e projetos com um objetivo comum.
- **Grades Globais:** grades estabelecidas sobre a Internet pública constituem os grades globais. Elas podem ser estabelecidas por organizações para facilitar negociações, compras e troca de serviços. Os recursos são distribuídos para usuários em qualquer lugar do mundo para computação e colaboração. Um exemplo são os *Inter Grids* que oferecem a capacidade de compartilhar recursos de computação e armazenamento de dados pela Web pública, a utilização é feita através da compra, venda e troca da capacidade e recursos de computação. Outro exemplo são as grades voluntárias, como SETI@HOME, em que usuários espalhados pelo mundo oferecem a capacidade de processamento de seus recursos para computação de dados de uma determinada organização, geralmente relacionada à pesquisa científica.
- **Grades de Computação ou alto desempenho:** essas grades são criadas com o simples objetivo de prover acesso a recursos computacionais. *Desktop Grids* que compartilham recursos de computadores pessoais, *Server Grids*, somente com servidores, e *High-Performance/Cluster Grids* com supercomputadores e clusters, são os integrantes desta classe.
- **Grades de Dados:** grades que requerem acesso ou processamento de dados são chamados de grades de dados. Eles otimizam as operações orientadas a dados. Embora elas possam consumir muita capacidade de armazenamento, estas grades não devem ser confundidas com os provedores de serviço de armazenamento.
- **Grades de Utilidades:** são definidas como sendo recursos computacionais comerciais mantidos e gerenciados por um provedor de serviço. Consumidores com necessidades de computação podem comprar “ciclos” de uma grade de utilidades.

Em adição a isto, os consumidores podem escolher utilizar esta grade para continuidade de negócios e para propostas de recuperação de desastres. Um exemplo são os chamados *Service Grids* que fornecem acesso à recursos que podem ser comprados por empresas para aumentar seus próprios recursos.

2.1.2 Arquitetura de uma grade

A arquitetura de uma grade identifica fundamentalmente seus sistemas, especifica as propostas e as funcionalidades destes componentes e indica como é a interação entre eles. A Figura 2.1.2 ilustra a arquitetura de uma grade e tem seus princípios baseados no modelo de ampulheta. A parte estreita define um pequeno conjunto de abstrações e protocolos que podem ser mapeados sobre diferentes tecnologias (FOSTER; KESSELMAN, 2003).



Figura 2: Arquitetura geral de uma grade (FOSTER; KESSELMAN, 2003).

As camadas são definidas como:

- **Estrutura Física:** esta camada provê os recursos para os quais o acesso compartilhado é gerenciado pelos protocolos da grade. Estes recursos envolvem sistemas de armazenamento, catálogos, recursos de rede e sensores. Os componentes desta parte implementam localmente as suas operações específicas, sejam elas, lógicas ou físicas, a fim de permitir o compartilhamento nos níveis mais altos.

- **Protocolos de Conectividade:** esta camada define os protocolos de comunicação e autenticação necessários para transações pela rede dentro da grade. Protocolos de comunicação habilitam a troca de dados entre os recursos da camada inferior. Já os protocolos de autenticação são construídos sobre serviços de comunicação para fornecer mecanismos criptograficamente seguros para verificação da identidade dos usuários e recursos.
- **Protocolos de Recursos:** definida junto à camada de protocolos de conectividade, esta camada obedece a regra de que um usuário da grade precisa estar habilitado a interagir com recursos e serviços remotos utilizando para tal, protocolos de negociação segura, inicialização, monitoramento, controle, contagem e pagamento de operações compartilhadas em recursos individuais.
- **Serviços Coletivos:** nesta camada, os protocolos e serviços não estão associados com nenhum recurso específico, mas sim com coleções de recursos. Os principais serviços aqui presentes são: serviço de diretório, responsável por fornecer aos usuários de uma organização virtual informações sobre a existência e propriedades dos recursos de outras organizações; serviços de co-alocação e escalonamento que permitem a alocação de um ou mais recursos para uma proposta específica e o escalonamento de tarefas nos recursos apropriados; serviço de monitoramento e diagnóstico que suporta o monitoramento dos recursos para falhas, ataques, sobrecarga, e outros motivos mais; e por último, o serviço de replicação de dados que suporta o gerenciamento dos recursos de armazenamento de uma organização virtual a fim de maximizar o desempenho no acesso a dados.
- **Aplicações de Usuários:** a última camada é composta pelas aplicações dos usuários que operam com sua respectiva organização virtual. As aplicações são construídas baseadas nos serviços fornecidos por qualquer camada, porém obedecendo a protocolos bem definidos e APIs que fornecem acesso para esses serviços.

Esta arquitetura formou a base para que em 2001, com o aumento das pesquisas em tecnologias para grades computacionais e com o surgimento do *Global Grid Forum* (GGF) (GLOBAL GRID FORUM, 2009), fosse produzida a *Open Grid Services Architecture* (OGSA), uma estrutura orientada a serviços e definida por um conjunto de padrões desenvolvidos e debatidos no fórum. OGSA tem sido adotada comumente pela comunidade de grade como uma estrutura unificada e se mantém firme devido ao apoio da maioria das comunidades acadêmicas e científicas tanto quanto a significativa aceitação nos setores comerciais, desde vendedores a usuários finais.

2.2 Middlewares para Grades

Muitos projetos foram e estão sendo desenvolvidos com intuito de oferecer um ferramental que possibilite a implementação e utilização da infraestrutura de grade. Esse ferramental é composto de sistemas de *software* chamados de *middlewares* sendo os mais importantes descritos a seguir.

2.2.1 Globus

Globus, idealizado por (FOSTER; KESSELMAN, 1997) e disponível em (GLOBUS PROJECT, 2009), fornece uma infraestrutura de *software* que permite a utilização de recursos computacionais distribuídos e heterogêneos como uma simples máquina virtual. O Globus é um projeto de pesquisa multi-institucional que visa permitir a construção de grades computacionais. O elemento central do projeto Globus é o *Globus Toolkit*, um conjunto de ferramentas que definem e implementam os serviços básicos e capacidades para construir uma grade, como segurança, locação e gerenciamento de recursos, e comunicações (FOSTER; KESSELMAN, 1999).

Globus é construído como um conjunto de camadas em que serviços globais de alto nível são concebidos e implementados através de serviços locais de baixo nível. O *Globus Toolkit* é um composto de módulos independentes, e uma aplicação pode explorar suas funcionalidades, como gerenciamento de recursos ou comunicação, sem utilizar suas bibliotecas. Essa possibilidade do uso parcial do Globus ajuda na adaptação de aplicações paralelas pré-existentes para a grade. Pode-se começar usando serviços mais básicos e ir, aos poucos, incorporando funcionalidades mais avançadas. Esta característica tem contribuído para a prevalência do Globus como padrão de infraestrutura para construção de grades. Atualmente o Globus Toolkit se encontra na versão 4.2.1.

2.2.2 Legion

Legion (GRIMSHAW; WULF, 1997) é um sistema baseado em objetos, desenvolvido na Universidade de Virginia para um sistema de milhões de máquinas e trilhões de objetos ligados por conexões de alta velocidade. Usuários trabalhando em suas máquinas de casa têm a ilusão de um simples computador, com acesso a todos os tipos de dados e recursos físicos, tais como bibliotecas digitais, câmeras, fitas de vídeo, etc. Grupos de usuários podem construir espaços virtuais compartilhados, para colaboração em pesquisa e troca

de informações. Esta abstração surge do escalonamento transparente, gerenciamento de dados, tolerância a falhas, autonomia, e a variedade de opções de segurança do Legion.

Legion é portanto um *software* de grade que habilita o compartilhamento seguro, eficiente e efetivo de dados, aplicações e poder computacional (GRIMSHAW et al., 1999). Dado que o Legion permite que estes recursos diversos e distribuídos sejam tratados como um ambiente operacional virtual simples, com uma simples estrutura de arquivos, ele reduz drasticamente o *overhead* do compartilhamento de dados, execução de aplicações e utilização do poder computacional disponível. Sua metodologia tem todas as vantagens de um sistema orientado a objetos, tais como abstração de dados, encapsulamento, herança e polimorfismo. O Legion tem sua versão comercial, chamada Avaki, descrita em (GRIMSHAW et al., 2003).

2.2.3 Condor e Condor-G

O Condor, idealizado por (LITZKOW; LIVNY; MUTKA, 1988) e disponível em (CONDOR PROJECT, 2009), é um sistema de gerenciamento de recursos especializado em tarefas de computação intensiva. Assim como outros sistemas, o Condor fornece mecanismo de gerenciamento de tarefas, definição de políticas de escalonamento, esquema de prioridades, monitoramento e gerenciamento de recursos (TANNENBAUM et al., 2002). Usuários submetem seus trabalhos para o Condor, e este subsequentemente escolhe quando e onde executar os trabalhos (baseado em alguma política), monitora o progresso, e finalmente informa ao usuário o término do trabalho. O Condor monitora um conjunto de recursos computacionais que podem estar distribuídos geograficamente formando assim um grande cluster ou uma grade. O gerenciamento pode ser dividido entre vários componentes do sistema, mantendo assim um controle descentralizado.

A principal característica do Condor é buscar por recursos ociosos, aproveitando assim o máximo que a infraestrutura pode oferecer. Esta idéia é baseada no conceito de *High Throughput Computing* descrito em (LIVNY; RAMAN, 1998) que se preocupa com o fornecimento de grande capacidade de processamento por longos períodos de tempo.

O Condor-G (FREY et al., 2002) representa o casamento das tecnologias dos projetos Globus e Condor. Do Globus são utilizados os protocolos para comunicação segura e acesso padronizado para uma variedade de sistemas remotos. Do Condor são utilizados os conceitos de submissão de trabalhos, alocação, recuperação de erros, e criação de um ambiente de execução amigável. O resultado é muito benéfico para o usuário que pode utilizar grandes conjuntos de recursos distribuídos sobre múltiplos domínios administrativos

como se todos esses recursos fossem locais (THAIN; TANNENBAUM; LIVNY, 2002).

2.2.4 gLite

O gLite (GLITE PROJECT, 2009) é, assim como o Globus, um conjunto integrado de componentes para o compartilhamento de recursos, ou seja, um ferramental para construção de grades. Ele integra componentes originados tanto dos melhores projetos de *middleware*, como Condor e Globus, quanto componentes desenvolvidos pelo *Large Hadron Collider Computing Grid* (LCG), projeto responsável por armazenar e processar a grande quantidade de dados produzida pelo acelerador de partículas, o mais novo experimento do CERN (LHC COMPUTING GRID, 2009). Assim, o gLite é uma solução de *middleware* para grades compatível com escalonadores como PBS¹ (PBS PROJECT, 2009), Condor, e LSF² (LSF PLATFORM, 2009), construída sobre o conceito de interoperabilidade e com o intuito de atender aplicações para grades de todas as possíveis áreas.

O gLite fornece diversas funcionalidades como gerenciamento de dados, gerenciamento de carga de trabalho, informação e monitoramento, contabilidade, segurança, monitoramento de rede e provisionamento. Desenvolvido como um componente do projeto EGEE (*Enabling Grids for E-sciencE*), o gLite segue todos os padrões estabelecidos para grades, possibilitando assim uma melhor interação e conexão com outros serviços para grades.

2.2.5 OGSA-DAI

O OGSA-DAI (OGSA-DAI PROJECT, 2009) é um *middleware* desenvolvido para lidar com o acesso e integração de dados originados de diferentes fontes em uma grade. Ele suporta a exposição de recursos de dados, como base de dados relacionais, para grades. Várias interfaces são fornecidas e muitos sistemas populares de gerenciamento de banco de dados são suportados.

O OGSA-DAI também inclui uma coleção de componentes para enfileiramento, transformação e entrega de dados em diversas formas, e um simples ferramental para o desenvolvimento de aplicações cliente. Além disso, OGSA-DAI é projetado para ser estendido, portanto usuários podem criar e fornecer suas próprias funcionalidades.

¹PBS - *Portable Batch System*

²LSF - *Load Sharing Facility*

2.2.6 OurGrid

O OurGrid, idealizado por (ANDRADE et al., 2003) e disponível em (OURGRID PROJECT, 2009), é um *software* aberto para computação *per-to-per* em grades. Sua característica mais importante é prover uma solução útil e eficiente para uma comunidade de usuários em constante produção de dados. As aplicações alvo são do tipo *Bag-of-Tasks*, e a idéia principal do OurGrid é abdicar da generalidade em alguns casos com o objetivo de obter uma solução que, apesar de simples, seja eficiente e possa ser facilmente usada com a constante produção de dados.

É importante notar que o objetivo do OurGrid contrasta com o objetivo do Globus, que fornece um conjunto de serviços para a construção da grade. Assim, o OurGrid é considerado como uma solução que complementa o Globus provendo um *broker* ou escalonador, o Mygrid (CIRNE et al., 2003; MYGRID PROJECT, 2009), e abstrações que permitem ao usuário usar tanto as funcionalidades do Globus como as funcionalidades do OurGrid. Por outro lado, o Globus também complementa o OurGrid ao fornecer a infraestrutura de serviços para execução de aplicações em larga escala.

O *broker* Mygrid foi desenvolvido com o intuito de motivar a utilização de grades para execução das aplicações paralelas com tarefas *Bag-of-Tasks*, ou seja, aplicações cujas tarefas são independentes, podendo ser executadas em qualquer ordem. Este tipo de aplicação é interessante pois se adequa melhor a ampla distribuição, heterogeneidade e dinamicidade da grade.

OurGrid persegue portanto um objetivo diferente do que seria prover uma solução genérica para computação em grade. No entanto, o desenvolvimento gradativo em busca de novas funcionalidades pode permitir que mais aplicações utilizem a solução, como por exemplo, o suporte a *workflow* e a *data-mining*. Atualmente na versão 4.0, o OurGrid é desenvolvido e mantido no Brasil por um grupo de pesquisa da Universidade Federal de Campina Grande.

2.2.7 Nimrod-G

Nimrod-G (BUYA; ABRAMSON; GIDDY, 2000) é um *broker* para grades que realiza gerenciamento de recursos e escalonamento. O escalonador do Nimrod-G tem a capacidade de alugar recursos da grade e serviços dependendo de suas capacidades, custo e disponibilidade. Ele ainda suporta descoberta de recursos, seleção, escalonamento, e a execução dos trabalhos dos usuários nos recursos remotamente. Os usuários podem selecionar um

limite de tempo no qual seus resultados precisam ser fornecidos e o *broker* do Nimrod-G tenta encontrar os melhores recursos disponíveis na grade com o intuito de atender este limite e minimizar os custos de execução da tarefa.

O Nimrod-G também suporta definições do usuário para otimizações de escalonamento e gerencia a fonte e a demanda dos recursos da grade usando um conjunto de serviços de negociação de recursos chamado *Grid Architecture for Computational Economy* (GRACE). O Nimrod-G é muito utilizado na solução de aplicações computacionais de larga escala com uso intensivo de dados, como por exemplo a modelagem molecular de medicamentos.

2.3 Gerenciamento de Recursos e Serviços

O termo gerenciamento de recursos refere-se às operações utilizadas para controlar o modo como as capacidades providas pelos recursos e serviços existentes são disponibilizadas para outras entidades, usuários, aplicações, ou serviços. Esta tarefa de gerenciamento está localizada no coração de uma grade e possui quatro funções principais: localizar um recurso, serviço ou capacidade, preparar para uso, utilizar e monitorar seu estado. Geralmente, o termo “recurso” é interpretado como um entidade física, como computador, rede, ou sistema de armazenamento. Porém em grades, este termo tem um significado genérico para denotar qualquer capacidade que pode ser compartilhada e explorada num ambiente ligado em rede, ou seja, recursos e serviços virtuais. O gerenciamento de recursos e serviços é mais uma das responsabilidades do ferramental de software presente nos *middlewares* descritos anteriormente e representado na Figura 3.

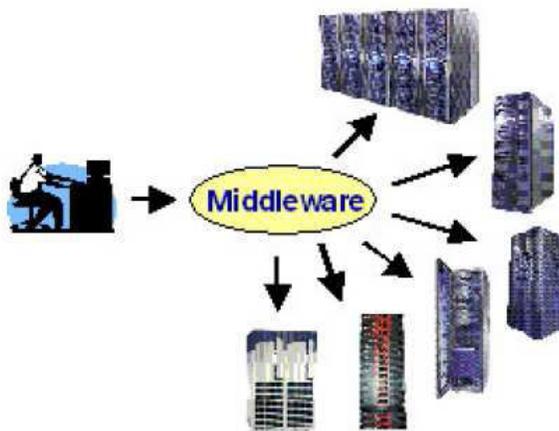


Figura 3: *Middleware - software* de gerenciamento (EGEE PROJECT, 2009)

Gerenciamento de recursos em sistemas computacionais tradicionais é um problema

bem definido e estudado, porém esses sistemas são locais e mantêm total controle dos recursos, podendo facilmente implementar mecanismos e políticas necessárias para o uso do recurso isoladamente. A diferença principal entre o gerenciamento de recursos em uma grade e o de sistemas locais é o fato de que os recursos a serem gerenciados em uma grade se encontram espalhados por vários domínios administrativos.

A distribuição de recursos por diferentes localidades e sob diferentes administrações pode apresentar problemas de heterogeneidade no modo como estes recursos estão configurados e como são administrados. Para resolver este problema, uma estrutura de protocolos e mecanismos padrões é utilizada para expressar um recurso ou uma tarefa. O desenvolvimento de uma gama de abstrações de gerenciamento e interfaces especializadas para diferentes classes de entidades torna-se inapropriado devido à demanda por aplicações cada vez mais complexas.

Funções de gerenciamento de recursos devem atender toda infraestrutura da grade, e serem aplicadas a toda variedade de recursos e serviços de uma única forma. Além disso, outro ponto fundamental para o gerenciamento de recursos em grades é o estabelecimento de um acordo mútuo entre o provedor de recurso e o consumidor, no qual o provedor concorda em oferecer uma capacidade que pode ser usada para realizar alguma tarefa do consumidor. A Figura 4 ilustra o funcionamento de uma grade e as interações entre seus componentes, o escalonador ou *resource broker* que conversa com os DRMs (*Domain Resource Manager*) e com o servidor de informações, com a utilização de mecanismos e protocolos padrões. Os DRMs também se comunicam com os servidores.

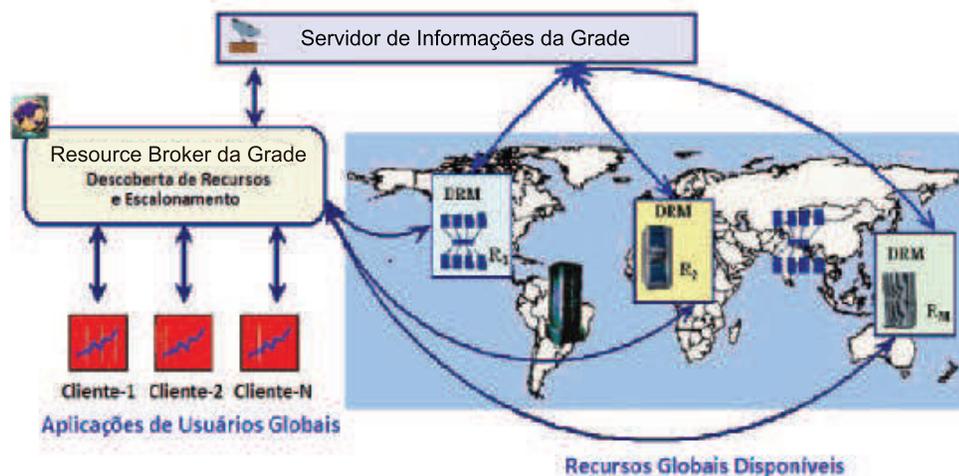


Figura 4: Interações de uma grade (FOSTER; KESSELMAN, 2003).

Os principais sistemas que constituem os gerenciadores de recursos para grades existentes atualmente são: GRAM (*Globus Grid Resource Allocation Manager*) integrado

ao Globus Toolkit, o GARA (*General-purpose Architecture for Reservation and Allocation*) uma extensão do anterior, Condor-G, e os chamados *Resource Brokers* que agem como intermediários entre a comunidade de usuários e o conjunto de recursos associados, geralmente usam o GRAM como protocolo base. Esses sistemas atuam separadamente no gerenciamento de recursos, cada um tem suas funções características e seu modo de atuação.

2.3.1 Escalonamento em Grades

Em sistemas distribuídos o escalonador é um dos componentes mais críticos do ferramental de gerenciamento ou *middleware*, devido à sua responsabilidade de selecionar recursos e escalonar tarefas ao mesmo tempo em que os requerimentos de usuários e aplicações chegam. Ele também é responsável pelo fornecimento de garantias sobre tempo de execução e custo dos recursos utilizados (MORENO, 2004). A Figura 5 ilustra o relacionamento entre usuário, escalonador e recursos.



Figura 5: Relacionamento entre cliente, escalonador e recursos.

O escalonamento de tarefas em uma grade é definido como o processo de tomar decisões envolvendo recursos sobre múltiplos domínios administrativos. Este processo pode incluir a pesquisa dentro desses domínios administrativos para uso de uma única máquina, ou escalonar uma simples tarefa para utilizar múltiplos recursos em um único ou vários lugares. Uma tarefa é definida como uma solicitação ou um processamento que necessita

de um recurso e o termo recurso é utilizado para caracterizar qualquer dispositivo que pode ser compartilhado e alocado: uma máquina, espaço em disco, rede, e tantas outras.

Baseado na sua definição, o escalonamento de tarefas em grades é dividido por (SCHOPF, 2002) em três fases principais: descoberta de recursos, na qual uma lista de recursos disponíveis é gerada; seleção do sistema, em que se faz a coleta de informações dos recursos e seleção do grupo adequado; e execução do trabalho, que além da própria execução, inclui a exibição do arquivo e a coleta dos resultados. Essas fases, e os passos pertencentes a cada uma são mostrados na Figura 6.

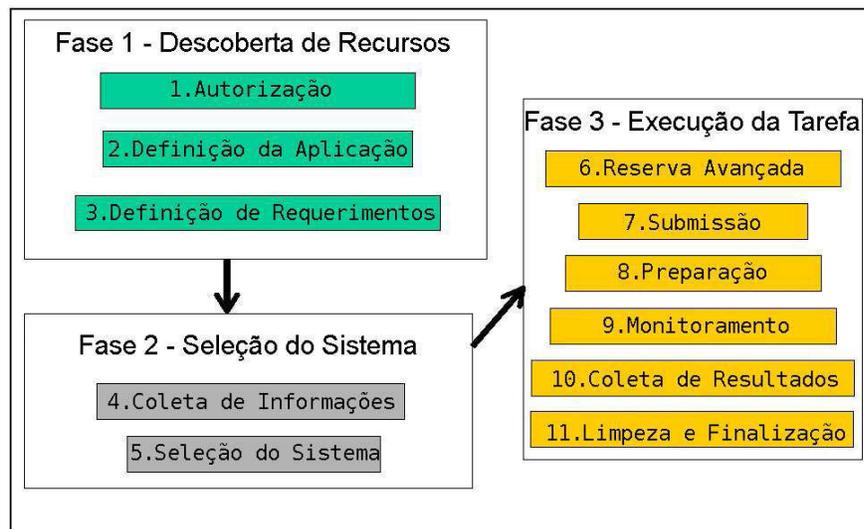


Figura 6: Fases do escalonamento (SCHOPF, 2002).

De acordo com as características de uma grade e sua arquitetura, pode-se ainda definir dois modos de operação para escalonadores de grades. O primeiro modo de operação é chamado de Escalonamento Centralizado, e é aquele em que o escalonador é um elemento central que controla todos os recursos do sistema, além de ser responsável pela alocação de todas as tarefas originadas dos diferentes usuários do sistema. O segundo modo de operação é chamado de Escalonamento Distribuído e baseia-se na divisão do escalonamento em duas partes, o escalonamento de aplicações e o escalonamento de recursos, cada um com responsabilidades distintas.

O Escalonamento Centralizado é adequado para infraestruturas de grades departamentais, empresariais ou de computação, em que o sistema apesar de distribuído é composto por *sites* muito próximos, permitindo assim o controle centralizado dos recursos. Desse modo, o escalonador é um elemento central que recebe as tarefas submetidas pelos usuários do sistema e toma decisões sobre a alocação de recursos para a execução dessas

tarefas, assim como representado pela Figura 5 apresentada anteriormente.

Já o Escalonamento Distribuído, por sua vez, é mais adequado para arquiteturas de grades globais, onde os recursos distribuídos não podem ser controlados de modo centralizado. Problemas de escalabilidade dentro de uma visão global coerente e a dificuldade em convencer os administradores de recursos que compõem a grade a abrir mão do controle de seus recursos são as principais razões pra não se ter um escalonador global. Tem-se portanto, no Escalonamento Distribuído, o escalonador de aplicações que procura melhorar o desempenho da aplicação do lado de quem a submete, escalonando as tarefas para *sites* de diferentes domínios administrativos, onde o escalonador de recursos local trabalha para maximizar a utilização dos recursos do seu *site*, como definido por (CIRNE; SANTOS NETO, 2005). Dependendo da arquitetura de um sistema distribuído, a abstração e a interação entre seus componentes, um sistema de escalonamento centralizado pode servir como um escalonador de aplicações ou um escalonador de recursos. A Figura 7 apresentada por (CIRNE; SANTOS NETO, 2005) e por (DEPARTAMENTO DE INTEGRAÇÃO DE SISTEMAS DE INFORMAÇÃO, 2009), ilustra o cenário de um Escalonamento Distribuído com os dois escalonadores de uma grade global.

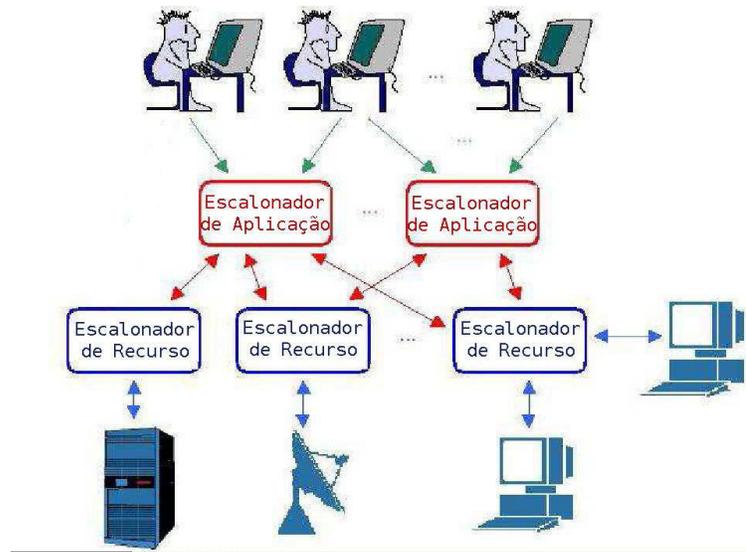


Figura 7: Ilustração de um cenário de escalonamento distribuído.

O papel do escalonador de aplicações é selecionar o *site* que melhor atende as necessidades de sua tarefa, ele obviamente não controla os recursos que usa, mas obtém acesso a tais recursos submetendo suas solicitações para os escalonadores que controlam os recursos. As decisões tomadas pelo escalonador de aplicações são normalmente baseadas

em um modelo de desempenho da aplicação e em dados sobre o estado atual dos recursos de um *site*, tendo portanto que conhecer detalhes das aplicações que escalonam e saber se o *site* selecionado é capaz de processar uma dada tarefa no tempo determinado. Sem isso, é difícil escolher os melhores *sites*, como também determinar qual tarefa alocar para cada recurso. Para obter tal informação, foram desenvolvidos sistemas de monitoramento e previsão do comportamento futuro de diversos tipos de recursos, como por exemplo o *Network Weather Service* (NWS) (WOLSKI; SPRING; HAYES, 1999), Remos (DEWITT et al., 1997), e o Ganglia (MASSIE; CHUN; CULLER, 2004).

Os sistemas de monitoramento e previsão de desempenho são muito úteis para vários tipos de aplicações como por exemplo, aplicações fortemente acopladas. Porém, infelizmente, existem problemas em aplicar esses sistemas em larga escala, especialmente quando se trata de prever o comportamento de canais de comunicação, que crescem com o aumento do número de máquinas na grade. No entanto, algumas aplicações são escalonáveis sem uso de informações de monitoramento ou previsão, um exemplo são aplicações *Bag-of-Tasks* que podem ser escalonadas sem o uso da informação dinâmica da grade.

2.4 Algoritmos de Escalonamento

Nesta seção serão descritos os principais algoritmos de escalonamento para grades, classificando cada um deles nos modelos de escalonamento definidos na seção anterior. Além dessa classificação, existem ainda duas categorias para classificação desses algoritmos, eles podem ser considerados algoritmos estáticos ou dinâmicos. A primeira categoria baseia-se na análise de informações coletadas antes do processo de escalonamento, desse modo as tarefas são escalonadas sem levar em consideração nenhuma informação sobre o estado atual do sistema, seus recursos e usuários. Já a segunda categoria é baseada na análise de informações coletadas em tempo real, assim as decisões de escalonamento mudam de acordo com as variações de estado dos componentes do sistema.

2.4.1 *Dynamic FPLTF*

Um bom exemplo de escalonadores estáticos é o *Fastest Processor to Largest Task First* (FPLTF) (MENASCÉ et al., 1995). Entretanto, o dinamismo e a heterogeneidade dos recursos presentes em uma grade caracterizam os escalonadores estáticos como uma solução não muito boa para ser utilizada. Para solucionar este problema foi desenvolvido o *Dynamic FPLTF* (SILVA; CIRNE; BRASILEIRO, 2003), que possui a habilidade de se adaptar

a alguma variação de um ambiente computacional como grade. O Dynamic FPLTF é considerado um escalonador centralizado, pois controla todas as tarefas e recursos através de um escalonador único.

Dynamic FPLTF necessita de três tipos de informação para escalonar as tarefas: *Tamanho da Tarefa*, *Carga da Máquina* e *Velocidade da Máquina*. Como o próprio nome diz, *Velocidade da Máquina* é a velocidade de um *host* ou nó do sistema e seu valor é relativo, por exemplo, uma máquina com *Velocidade da Máquina*=2 executa uma tarefa duas vezes mais rápido do que uma máquina com *Velocidade da Máquina*=1. A informação contida em *Carga da Máquina* representa o quanto uma máquina está carregada localmente, ou seja, a capacidade de processamento que não está disponível para nenhuma aplicação da grade. E finalmente, *Tamanho da Tarefa* representa o tempo necessário que uma máquina com *Velocidade da Máquina*=1 leva para completar uma tarefa quando *Carga da Máquina*=0. No começo do algoritmo, uma variável TeD (Tempo para estar Disponível), é iniciada com valor 0 para cada máquina e as tarefas são organizadas em ordem decrescente de tamanho. Portanto, uma tarefa grande é alocada primeira e uma tarefa é alocada ao *host* que oferecer o melhor tempo de execução (TE) para ela, em que:

- $TE = TeD + \text{Custo da Tarefa};$
- $\text{Custo da Tarefa} = (\text{Tamanho da Tarefa} / \text{Velocidade da Máquina}) / (1 - \text{Carga da Máquina});$

Quando uma tarefa é alocada em um *host*, o valor TeD correspondente deste *host* é incrementado pelo *Custo da Tarefa*. Tarefas são alocadas até que todas as máquinas da grade estejam em uso. Depois disso, a execução da aplicação começa. Quando uma tarefa termina sua execução, todas as tarefas que não estão executando são escalonadas novamente até que todas as máquinas fiquem em uso. Este esquema continua até todas as tarefas serem finalizadas. Esta estratégia tenta minimizar os efeitos do dinamismo da grade.

Uma máquina que a priori, não é tão rápida e não está carregada, recebe mais tarefas do que uma muito rápida mas muito carregada. O problema acontece quando uma máquina antes não carregada fica muito carregada de repente comprometendo a aplicação, pois tarefas escalonadas para essa máquina serão executadas mais lentamente. O escalonamento contínuo de tarefas soluciona este problema pela alocação de grandes tarefas para as máquinas mais rápidas no momento do escalonamento. Este algoritmo apresenta bom desempenho mas é difícil de ser implementado na prática. Ele necessita de muita

informação sobre o ambiente, o que pode não ser fácil de obter ou nem estar disponível. Isso torna o algoritmo inviável para alguns ambientes computacionais nos quais certas informações são difíceis de obter.

2.4.2 *Workqueue with Replication*

O *Workqueue with Replication* (WQR) (SILVA; CIRNE; BRASILEIRO, 2003) é uma extensão do algoritmo de escalonamento *Workqueue* ou “Fila de Trabalho”, que distribui as tarefas para execução na mesma ordem em que elas chegam ao escalonador. Devido sua simplicidade, o algoritmo *Workqueue* não consegue atender ao dinamismo de um sistema distribuído e torna-se ineficiente diante das variações do ambiente. O WQR foi desenvolvido para solucionar esse problema, através da inclusão do método de replicação de tarefas. Com a replicação, as tarefas e suas réplicas são executadas em várias máquinas distintas, aumentando assim a probabilidade de uma execução bem sucedida e contornando as variações do ambiente sem a utilização de informações sobre as mesmas. Assim como o Dynamic FPLTF, o WQR também é qualificado como um algoritmo de escalonamento centralizado. Em um sistema de escalonamento distribuído, o algoritmo WQR serviria tanto para um escalonador de aplicações quanto para um escalonador de recursos.

Em sua fase inicial, o WQR é similar ao *Workqueue* tradicional, ou seja, as tarefas são enviadas para execução nos processadores que se encontram disponíveis conforme a ordem de chegada. Quando um processador finaliza a execução de uma tarefa, este recebe uma nova tarefa para processar e assim por diante. Os algoritmos WQR e *Workqueue* passam a diferir no momento em que um processador fica disponível e não há mais nenhuma tarefa pendente para executar. Neste momento, o *Workqueue* já terminou seu trabalho e apenas aguarda a finalização de todas as tarefas. Porém, o WQR inicia sua fase de replicação para tarefas que ainda estão executando, e assim que a tarefa original ou uma de suas réplicas finalizam, as outras réplicas são interrompidas.

O WQR é considerado um bom algoritmo pois alcança bons níveis de desempenho sem a utilização de informação dinâmica sobre os processadores, conexões de rede e tempo de execução das tarefas, considerando aplicações que não processam dados de forma intensiva. Há porém, um problema no uso da replicação, as réplicas que são interrompidas causam desperdício de processamento, pois o resultado de sua computação interrompida não é útil.

Um algoritmo muito similar a este é o proposto em (FUJIMOTO; HAGIHARA, 2004), chamado de algoritmo RR, ele segue o mesmo método de réplicas do WQR, porém mantém

a estrutura de um anel de tarefas, e leva esse nome pois trabalha sobre o modelo *round robin* para seleção da tarefa que será replicada.

2.4.3 *XSufferage*

O *XSufferage*, também referenciado em (CASANOVA et al., 2000) e em (SANTOS NETO, 2004), é uma heurística de escalonamento baseada nas informações sobre o desempenho dos recursos. Este algoritmo aborda o impacto da transferência de dados, em aplicações que usam grande volume de dados, através da reutilização e espalhamento dos dados. O *XSufferage* é uma extensão do algoritmo *Sufferage* (IBARRA; KIM, 1977) no qual a idéia básica é determinar quanto cada tarefa seria prejudicada se ela não fosse escalonada na máquina que a executaria de forma mais eficiente. Portanto, o *Sufferage* prioriza as tarefas de acordo com o valor que mede o prejuízo de cada tarefa. Este valor é denominado **sofrimento** e é definido como a diferença entre o melhor e o segundo melhor tempo de execução previsto para a tarefa, considerando todos os processadores da grade.

A principal diferença entre o *Sufferage* e *XSufferage* é o método usado para calcular o valor do sofrimento. O *XSufferage* também leva em consideração o valor correspondente à transferência dos dados utilizados pela tarefa. Esse método implica no uso das informações já abordadas pelo *Sufferage* mais as informações sobre largura de banda disponível na rede que conecta os recursos. Tanto o *Sufferage* quanto o *XSufferage* são considerados algoritmos para escalonamento centralizado, e dentro de um sistema de escalonamento distribuído eles seriam mais adequados para um escalonador de aplicações.

O efeito esperado pelo algoritmo *XSufferage* é a diminuição do tempo total de execução da aplicação através da redução nas transferências de dados. A sua avaliação mostra que evitando transferências desnecessárias o desempenho da aplicação é melhorado (SANTOS NETO, 2004). Os cálculos para determinação dos valores de sofrimento são baseados no conhecimento da carga de CPU, da largura de banda da rede e dos tempos de execução das tarefas, informações difíceis de obter em alguns casos, dependendo das restrições administrativas sobre os recursos que compõem a grade. A necessidade pela coleta de muitas informações pode tornar o algoritmo de escalonamento lento, atrasando assim a execução das tarefas.

2.4.4 *Storage Affinity*

O *Storage Affinity* (SANTOS NETO, 2004) foi feito com o intuito de explorar a reutilização de dados para melhorar o desempenho de aplicações que utilizam grandes quantidades de dados. A reutilização de dados pode ser classificada em dois tipos básicos, *inter-job* e *inter-task*. O primeiro tipo, *inter-job*, ocorre quando um *job* utiliza dados já usados ou produzidos por outro *job* que executou anteriormente. Já o segundo tipo ocorre em aplicações em que os *jobs* compartilham seus dados de entrada.

O método de escalonamento de tarefas é baseado no conceito de afinidade, cujo valor é calculado para uma tarefa e um *site* da grade, determinando assim quão próximo do *site* esta tarefa está. A semântica do termo próximo está associada à quantidade de dados de entrada da tarefa que já está armazenada no *site* considerado, ou seja, quanto mais bytes de entrada da tarefa já estiverem armazenados no *site*, mais próxima a tarefa estará do *site*, pois possui maior afinidade.

O algoritmo *Storage Affinity* utiliza somente informações sobre o tamanho e a localização dos arquivos de entrada. Tais informações podem estar disponíveis no instante do escalonamento sem dificuldade e perda de precisão. Por exemplo, as informações relacionadas aos dados de entrada de uma tarefa podem ser obtidas através da requisição aos servidores de dados e, caso estes servidores não sejam capazes de responder, uma implementação alternativa da heurística *Storage Affinity* poderia facilmente manter um histórico das informações sobre as transferências de dados efetuadas para cada tarefa.

Além de aproveitar a reutilização de dados, o *Storage Affinity* também realiza a replicação de tarefas, tratando da dificuldade de obtenção de informações dinâmicas sobre a grade e informações sobre o tempo de execução das tarefas. A idéia é que as réplicas tenham a chance de serem submetidas a processadores mais rápidos do que aqueles associados às tarefas originais, reduzindo o tempo total de execução da aplicação.

2.4.5 *Multiple Queue - Hierarchy*

A idéia básica do algoritmo *Multiple Queue - Hierarchy* (MQ-H) é organizar os recursos disponíveis na grade em níveis de hierarquia de acordo com suas características de desempenho. Cada nível de hierarquia é composto de recursos que proporcionam um melhor desempenho para uma determinada tarefa, que é classificada e alocada de acordo com sua complexidade. Em outras palavras, uma tarefa de complexidade específica é melhor atendida por um determinado nível de hierarquia, de acordo com suas necessidades

em termos de *overhead* e recursos disponíveis (SILBERSTEIN; GEIGER; SCHUSTER, 2006).

Neste modelo de níveis, são atribuídas uma ou mais filas de tarefas para cada nível, sendo que cada fila tem dois valores limites de sinalização: T_q (tempo máximo de espera na fila), e T_e (tempo limite para execução de uma tarefa nesta fila); sendo que $T_e \leq T_q$. Neste esquema, todas as tarefas novas são submetidas para filas no primeiro nível hierárquico da grade, englobando sistemas de baixo poder computacional, no qual esperam até serem executadas, ou até estourar o tempo limite da fila, sendo neste último caso transferidas para as filas do próximo nível. A cada nível, o poder computacional dos nós da grade aumenta, assim como aumentam também os valores de tempo limite de espera e tempo de execução.

2.4.6 *Fair-Share Scheduler*

O *Fair-Share Scheduler* (KAY; LAUDER, 1988) ou “Escalonador para Compartilhamento Justo” é uma política de escalonamento de tarefas que busca pela justiça no compartilhamento e na alocação de recursos entre os usuários de um sistema. O escalonamento é centralizado e baseado no cálculo de prioridades dos usuários através de informações sobre a utilização dos recursos do sistema. As informações utilizadas são: histórico de uso dos recursos, porção dos recursos, e o número de processos ativos de cada usuário.

O processo de escalonamento funciona através da constante atualização das informações dos usuários e assim que feitos os cálculos para determinar as prioridades de cada usuário, aquele com maior prioridade tem sua tarefa selecionada e escalonada para execução. A idéia do algoritmo é manter justiça na utilização dos recursos entre os usuários do sistema. Um usuário que utiliza muitos recursos durante um período, terá direito de utilizar menos recursos no futuro, recompensando assim os outros usuários que ficaram na espera por recursos. O processo de atualização ocorre de acordo com a definição do parâmetro de “meia-vida” da política Fair-Share, esse parâmetro determina quão rápido as informações dos usuários são atualizadas e indica o prazo necessário para a garantia de justiça entre os usuários.

Uma característica importante a ser descrita é que a política *Fair-Share* não utiliza da interrupção de tarefas para a realocação de recursos, ou seja, os recursos alocados para um usuário 1 somente serão realocados para um usuário 2 quando as tarefas do usuário 1 terminarem. Essa política defende que a justiça no compartilhamento de recursos está na igualdade da utilização entre os usuários do sistema ao longo do tempo num longo prazo, sem se preocupar com a necessidade que um usuário pode ter de utilizar os recursos o mais

rápido possível. Definir o que é realmente justo no compartilhamento e escalonamento de recursos é algo complexo de se fazer, como mostrado por (KLEBAN; CLEARWATER, 2003).

2.4.7 Rede de Favores

A Rede de Favores (ANDRADE, 2004) considerada uma política para compartilhamento de recursos, pode ser também entendida como uma política de alocação de recursos entre os *sites* de uma grade. Essa política defende a idéia de compartilhamento e união de recursos como uma rede de troca de favores, com o intuito de incentivar a criação de ambientes computacionais como grades.

Diferentemente das políticas anteriores, a Rede de Favores é baseada num sistema descentralizado onde cada *site* do sistema mantém o controle da alocação dos seus recursos, e um sistema de favores par-a-par entre os *sites* da grade é estabelecido. As contribuições ou doações de recursos feitas por um *site A*, em benefício de um *site B*, são contabilizadas pelo *site B* com o intuito de futuramente oferecer uma recompensa para o *site A*.

A Rede de Favores e seu sistema de recompensa representam um grande incentivo à criação de grades e ao compartilhamento de recursos porém, assim como a política *Fair-Share*, ela defende a idéia de justiça ou neste caso, recompensa, ao longo do tempo em longo prazo. A necessidade de um usuário pela pressa na utilização dos recursos também não é considerada nessa política.

2.5 Considerações Finais

Neste capítulo foi descrito o estado da arte na área de grades computacionais, levando em consideração importantes aspectos como arquitetura, modo de funcionamento, ferramentas, métodos de gerenciamento, algoritmos de escalonamento, etc. Um estudo e uma avaliação, sobre alguns dos algoritmos de escalonamento aqui citados, foram feitos como parte inicial deste trabalho. Os resultados obtidos foram publicados no WPerformance 2007 (FALAVINHA JUNIOR et al., 2007), evento integrante do Congresso Anual da Sociedade Brasileira de Computação, e afirmaram a melhor adequação de algoritmos dinâmicos a sistemas como grades, mostrando também a necessidade de tolerância à falhas e soluções de problemas observados com a análise prática dos algoritmos.

3 *Escalonamento Owner-Share*

Neste capítulo apresenta-se a política de escalonamento *Owner-Share*, cujo objetivo é garantir o direito que os usuários proprietários de recursos têm de usar seus recursos o mais rápido possível assim que necessário. Primeiramente, introduz-se conceitos essenciais tais como o de propriedade distribuída e compartilhamento distribuído, para então definir-se a política de escalonamento, seus algoritmos e as métricas de desempenho elaboradas para sua avaliação. Ainda no fim do capítulo, introduz-se o *checkpointing* de tarefas como uma possível funcionalidade para a política *Owner-Share*.

3.1 Definições Fundamentais

A política *Owner-Share* tem como fundamentação os conceitos de propriedade e compartilhamento. Esses conceitos são redefinidos a seguir, segundo a ótica do *Owner-Share*.

3.1.1 Propriedade Distribuída e Compartilhamento Distribuído

Em uma grade os elementos que compõem o sistema estão distribuídos ao longo de sua estrutura. Essa estrutura distribuída implica em que a propriedade dos elementos é também distribuída. Apesar de distribuídos os elementos são compartilhados entre todos os usuários, quer sejam proprietários ou não. Esse compartilhamento é obtido pela partilha dos elementos entre os usuários segundo suas necessidades momentâneas. Definem-se a seguir os conceitos de porção e de propriedade distribuída, que levarão, na sequência ao conceito da política de escalonamento baseada em propriedade.

3.1.2 Definição de Porção ou *Share*

A palavra “partilhar” significa dividir algo com alguém, como por exemplo partilhar o conhecimento em um determinado assunto ou partilhar o uso de recursos de um sistema qualquer. Assim, uma porção ou *share* de um sistema é definida como sendo a parte,

resultante do compartilhamento do sistema como um todo, que é distribuída, fornecida ou de propriedade de uma pessoa ou grupo.

Na política *Owner-Share*, o conceito de porção refere-se à parte dos recursos do sistema pertencentes ao usuário. Cada usuário pode ser ou não um proprietário de uma parte dos recursos do sistema distribuído, sendo que os usuários proprietários e a união dos seus recursos em um único sistema caracterizam o conceito de propriedade distribuída.

3.1.3 Definição de Propriedade Distribuída

Propriedade distribuída é o termo usado para expressar as múltiplas propriedades em infraestruturas agregadas, compostas por várias partes com domínios administrativos independentes mas com um propósito comum, similar a uma parceria ou sociedade. O maior problema é definir o objetivo comum do sistema, uma vez que cada parte integrante tem suas próprias regras e políticas administrativas (FOSTER; KESSELMAN; TUECKE, 2001).

A demanda por sistemas distribuídos construídos através de parcerias é real e muitas pessoas têm necessidades computacionais que não podem ser resolvidas apenas por suas infraestruturas locais. Todos os benefícios desses sistemas à ciência, como acessibilidade, alto poder computacional e mobilidade computacional criam expectativas em todos os campos de pesquisa, comércio e negócios.

3.2 Política *Owner-Share*

A busca pela alta capacidade de processamento e armazenamento sem gastos excessivos com *hardware* é o que motiva a união e o compartilhamento de recursos entre diferentes entidades e usuários, formando assim, sistemas distribuídos de grande porte como grades. Apesar da necessidade de união e compartilhamento de recursos, os usuários donos de recursos podem ainda exigir a garantia de uso da sua porção de recursos quando necessário, afinal ele é proprietário dos recursos e possui o direito de usá-los quando quiser. Baseado na necessidade de oferecer essa garantia ao usuário dono de recursos, criou-se a política de escalonamento e alocação de recursos chamada *Owner-Share Enforcement Policy* (OSEP) ou Política de Garantia da Porção do Proprietário.

A política OSEP garante aos usuários do sistema o acesso e o uso de suas porções de recursos assim que requisitadas. Por garantir isso, a política OSEP representa uma solução inovadora na área de sistemas distribuídos e grades computacionais, além de um

grande incentivo à união e ao compartilhamento de recursos.

3.2.1 Definições do *Owner-Share*

Considere um *cluster* com n máquinas homogêneas em um sistema com m usuários, $U = \{u_1, \dots, u_m\}$, com diferentes quantidades de tarefas, e as seguintes definições:

Owner Share ou Porção do Proprietário é o parâmetro que quantifica a menor porção dos recursos do sistema que cada usuário tem o direito de utilizar quando necessário. Esse valor é equivalente à quantidade de recursos que este usuário forneceu ao sistema e é definido formalmente abaixo:

Definição 1: A porção dos recursos do usuário u_i , denotada por p_i , é o número de recursos, máquinas ou nós n_i que u_i forneceu para a infraestrutura, na qual, se n é o número total de nós do sistema, então $\sum_i p_i = n$.

Feasible Share of Resources ou Porção Efetiva de Recursos (fes_i) determina o número exato de recursos que deveriam ser alocados para cada usuário, considerando a demanda, a **Porção do Proprietário** de cada usuário, e o número de recursos não utilizados pelos usuários que têm $d_i < p_i$, onde d_i representa a demanda do usuário. O valor de fes_i é determinado, seguindo o mesmo método utilizado em (AMAR et al., 2007) para a política *Fair-Share*, por:

Definição 2: A Porção Efetiva de Recursos (fes_i) pode ser determinada pela equação $fes_i \equiv \min\{(p_i + \hat{p}_i), d_i\}$, onde:

- p_i é a *Owner-Share* ou porção correspondente ao número de recursos ou nós que o usuário u_i possui;
- \hat{p}_i é o excesso de nós alocados para o usuário u_i ;
- d_i é a demanda do usuário u_i pelos recursos do sistema;
- $\sum_i fes_i = n$.

3.2.2 Algoritmo da Política *Owner-Share*

Na política OSEP, os usuários do sistema devem receber um número de recursos equivalente ao mínimo valor entre sua demanda por recursos e sua porção do proprietário, ou então, ter alocado quantos recursos forem possíveis se houver máquinas livres. Por isso, o total controle sobre as tarefas e recursos do sistema é indispensável e, tendo em vista essa necessidade, qualifica-se a política OSEP como uma política adequada para o modelo de escalonamento centralizado.

O algoritmo responsável por garantir a política OSEP é portanto centralizado sobre um único escalonador, que controla a distribuição de tarefas para um sistema distribuído em um ambiente de compartilhamento preemptivo. Um ambiente preemptivo se caracteriza pela possibilidade de interrupção de tarefas quando necessário, obedecendo às decisões tomadas pelo algoritmo de escalonamento. O algoritmo tem por objetivo ajustar o número de tarefas em execução ou recursos alocados para cada usuário com o intuito de atender a política OSEP e alcançar a porção do proprietário correspondente a cada usuário.

O algoritmo de escalonamento da política OSEP é ainda dividido em duas partes principais: a primeira é responsável pelo dinamismo do algoritmo e atualização de informações; já a segunda parte é responsável pela análise de informações do usuário sobre os recursos alocados, e pela tomada de decisões obedecendo à política OSEP. Nas seções seguintes define-se esses dois componentes do algoritmo. A Figura 8 ilustra as duas partes do algoritmo da política OSEP.

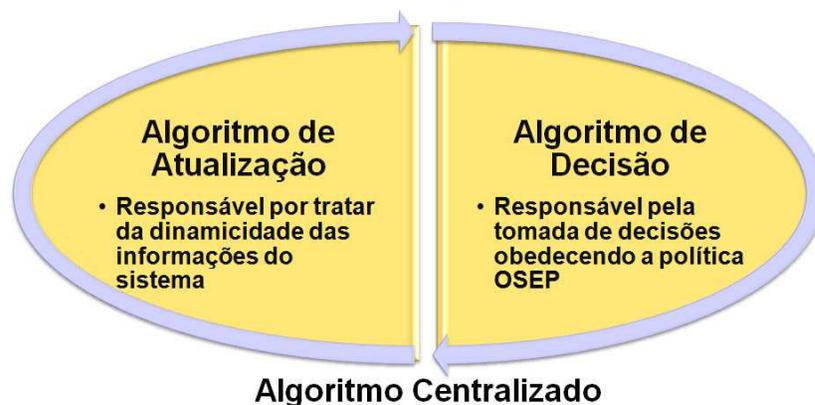


Figura 8: Algoritmo da política OSEP dividido em duas partes.

3.2.2.1 Algoritmo Dinâmico de Atualização

O algoritmo da política OSEP precisa de informações atualizadas sobre o uso dos recursos pelos usuários e suas tarefas a fim de tomar as decisões adequadas de acordo com o estado atual do sistema. Esses dados devem ser armazenados em uma estrutura muito bem organizada, de acesso fácil e rápido.

O monitoramento dos recursos e tarefas é, portanto, uma funcionalidade indispensável para o algoritmo *Owner-Share*. A implementação de um mecanismo como esse não é tão simples, uma vez que os recursos estão amplamente distribuídos e sob diferentes domínios administrativos. Existem sistemas de escalonamento e/ou enfileiramento de tarefas que fornecem funcionalidades como monitoramento, armazenamento e atualização de informações sobre recursos e tarefas, como por exemplo, o sistema Condor. O Condor monitora os recursos e tarefas do usuário, e ainda gerencia suas informações através de uma estrutura organizada e eficiente, chamada ClassAds (RAMAN; LIVNY; SOLOMON, 1998).

O algoritmo de escalonamento para a política *Owner-Share* pode ser implementado sobre qualquer infraestrutura ou sistema de escalonamento, desde que o monitoramento de recursos e tarefas esteja disponível. Uma estrutura para o gerenciamento das informações monitoradas também é um requisito para sua fácil implementação. Funcionalidades como armazenamento, atualização e o acesso dessas informações são essenciais para o controle dos recursos e tarefas dos usuários. A partir desses requisitos e funcionalidades, o algoritmo dinâmico de atualização das informações é definido abaixo.

Algoritmo Dinâmico de Atualização

1. A cada t segundos, o sistema de atualização faz:
2. Para cada tarefa na fila do Escalonador{
3. Carrega as informações monitoradas sobre o recurso alocado e a tarefa;
4. Analisa as informações de *status* da tarefa e de seu dono/usuário;
5. Atualiza as informações do usuário na estrutura de armazenamento;
6. }
7. Chama a função **Algoritmo de Decisão**(*maxt*);

O **Algoritmo Dinâmico de Atualização** considera as variáveis *maxt* e t , em que t é o intervalo de iteração do **Algoritmo Dinâmico de Atualização**, e *maxt* é o número

máximo de tarefas que podem ser suspensas em cada ativação do **Algoritmo de Decisão**. É importante ressaltar que a quantidade de recursos alocados para um usuário ou a quantidade de tarefas em execução desse mesmo usuário refletem exatamente a mesma informação e que essa é fundamental para o funcionamento do algoritmo. O fluxograma e o diagrama de objetos da Figura 9 mostram o funcionamento do **Algoritmo Dinâmico de Atualização**, além dos dados e estruturas utilizadas.

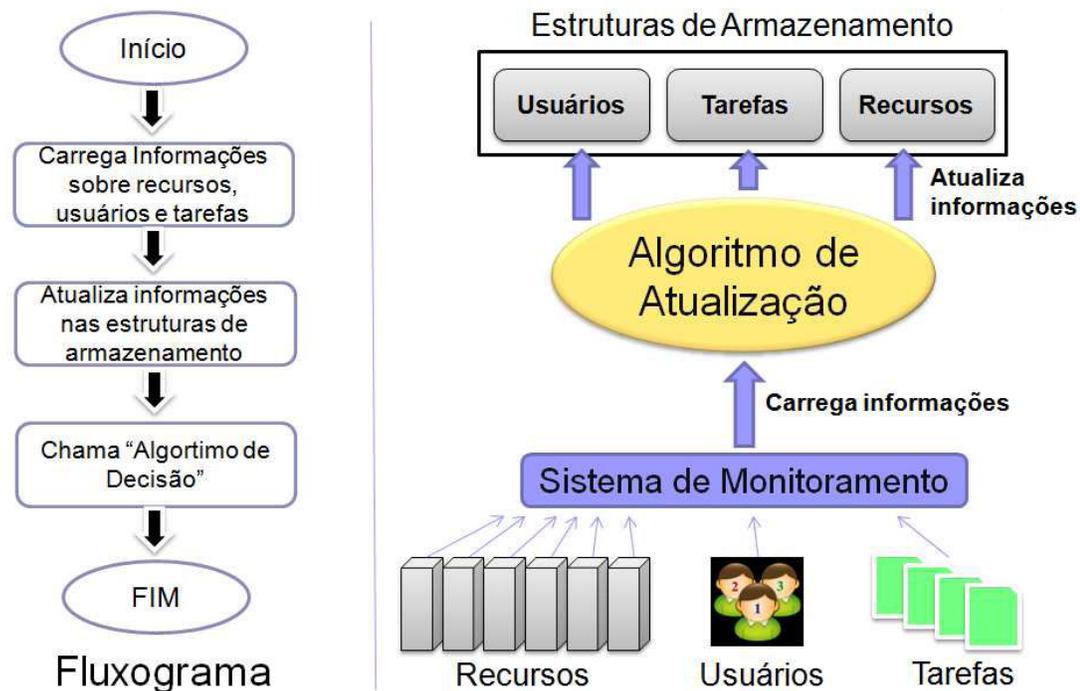


Figura 9: Fluxograma e diagrama de objetos do **Algoritmo Dinâmico de Atualização**.

3.2.2.2 Algoritmo de Decisão

O mecanismo de realocação de recursos/tarefas, descrito pelo **Algoritmo de Decisão**, é ativado a partir do algoritmo anterior. Ele decide quando suspender uma tarefa baseado nas informações sobre os usuários e suas tarefas. Através da informação previamente armazenada pelo **Algoritmo Dinâmico de Atualização**, uma tarefa pode ser considerada **Ativa** (em execução), **Pronta** (pronta pra ser alocada) ou **Em Espera** (bloqueada pelo usuário ou pelo escalonador por alguma razão). Em cada iteração, o algoritmo verifica as tarefas no estado **Pronta** e negocia a alocação dessa tarefa através da preempção de uma tarefa **Ativa**.

A escolha de qual tarefa será suspensa é feita pela comparação dos tempos de execução das tarefas de um determinado usuário, condicionada a:

1. A tarefa a ser suspensa é escolhida dentre as pertencentes ao usuário com a maior

quantidade de recursos alocados que executam a sua porção de proprietário (*owner-share*).

2. A tarefa a ser suspensa, atendida a primeira restrição, é aquela com o menor tempo de execução, a fim de minimizar o desperdício de processamento.

O algoritmo de decisão depende dos seguintes parâmetros e informações:

- Número de tarefas no estado Pronta, ou seja, a demanda do usuário ainda não atendida, representado por $i_{usuario}$.
- Diferença entre sua porção de usuário $p_{usuario}$ e o número atual de tarefas do estado Ativa ($nt_ativa_{usuario}$), dada por: $f_{usuario} = p_{usuario} - nt_ativa_{usuario}$.
- Tempo de execução de cada tarefa do usuário, w_{job} .
- Estado atual de cada tarefa do usuário, s_{job} .
- Número atual de usuários, m .
- Número máximo de tarefas que podem ser suspensas a cada ativação do algoritmo, $maxt$, recebido do Algoritmo Dinâmico de Atualização.

Este algoritmo funciona de forma recursiva até que $maxt$ tenha chegado a zero, ou seja, o número máximo de tarefas a serem suspensas por iteração já foi alcançado. As informações sobre tarefas e usuários são atualizadas a cada modificação feita pelo escalonador, dessa forma se cria um algoritmo dinâmico que atende às necessidades do sistema conforme seu estado atual. Quando p atinge seu limite, a execução volta para o **Algoritmo Dinâmico de Atualização** que controla os intervalos de iteração. A Figura 10 mostra o fluxograma e o diagrama de objetos do **Algoritmo de Decisão** mostrando seu funcionamento e as estruturas utilizadas.

Algoritmo de Decisão (int *maxt*)

1. Se ($maxt > 0 \wedge (\exists u_{max} \mid f_{max} = max(f_1, \dots, f_m) \wedge f_{max} > 0 \wedge i_{max} > 0)$)
2. { Se ($\exists u_{min} \mid f_{min} = min(f_1, \dots, f_m) \wedge f_{min} < 0$)
3. { Se ($\exists \text{Tarefa } j \in u_{min} \mid w_j = min(w_1, \dots, w_l) \wedge (s_j = \text{Ativa})$)
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 as tarefas *j* e *k*;
8. }
9. Chame *Algoritmo de Decisão(maxt)*;
10. }
11. }

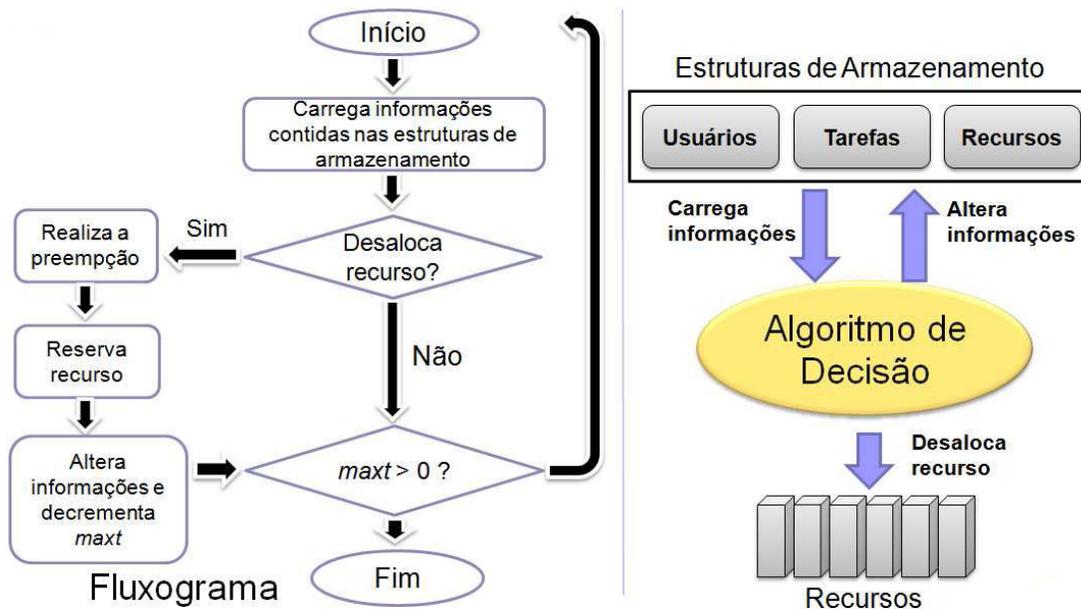


Figura 10: Fluxograma e diagrama de objetos do **Algoritmo de Decisão**.

Para completar o sistema de funcionamento do algoritmo da política *Owner-Share* faz-se uso de um mecanismo de reserva e alocação de recursos chamado de **Negociador**. Esse mecanismo trata-se de um componente responsável pela realização das decisões tomadas

pelo algoritmo, ou seja, ele responsável por garantir a reserva de um recurso e/ou alocar uma tarefa para um recurso. A Figura 11 apresenta o diagrama de funcionamento do algoritmo da política *Owner-Share* com todos os seus componentes.

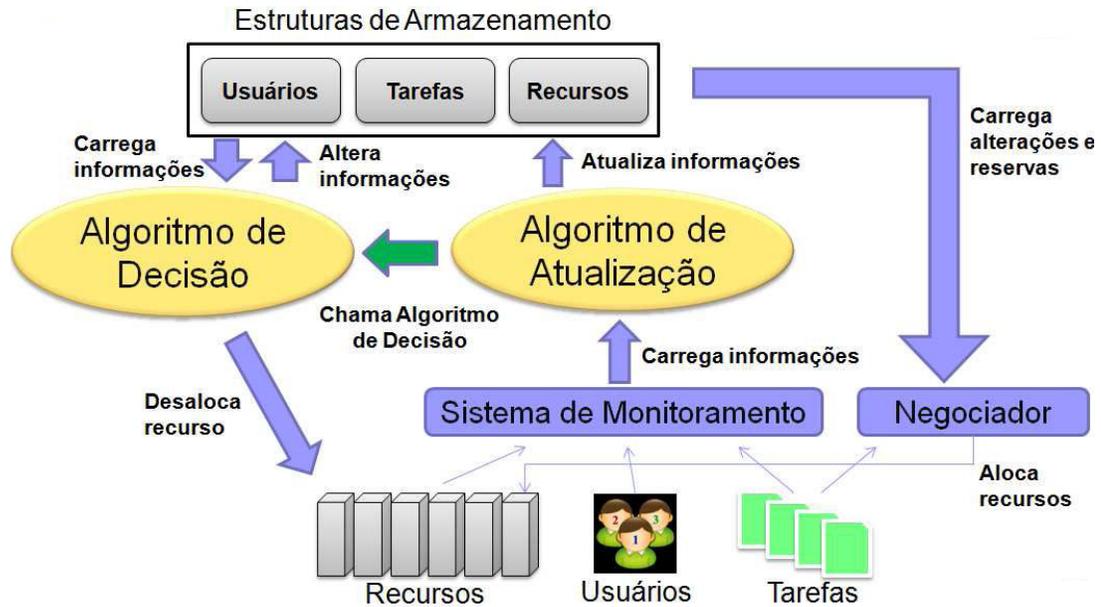


Figura 11: Diagrama do funcionamento do algoritmo da política **Owner-Share**.

3.2.2.3 Validação do Algoritmo

A validação do algoritmo é feita através da verificação do atendimento dos pontos importantes que a política diz garantir ao usuário dono de recursos no sistema. Assim, o algoritmo da política *Owner-Share* deve garantir que:

- (A) O usuário tenha acesso, no mínimo, à quantidade de recursos equivalente à sua porção, ou seja, $fes_i \geq p_i$ quando $d_i \geq p_i$;
- (B) A preempção de tarefas ou desalocação de recursos não deve desalocar mais recursos do que o devido e necessário;

A validação da primeira hipótese é feita a partir do exame das condições avaliadas nas linhas 1 e 2 do Algoritmo de Decisão.

Pela Linha 1 (identificação de usuário com $fes_i < p_i$ e $d_i > fes_i$, ou seja, ainda sem ocupar sua porção e tendo tarefa não atendida) tem-se:

$$\nexists \text{ alocação} \Rightarrow \forall k : k \in \text{usuário} : f_k \leq 0 \vee i_k \leq 0. \quad (\text{i})$$

Pela Linha 2 (identificação de usuário com $fes_i > pi$) tem-se:

$$\nexists \text{ alocação} \Rightarrow \forall k : k \in \text{usuário} : f_k \geq 0. \quad (\text{ii})$$

A junção das cláusulas (i) e (ii) diz que não haverá preempção de tarefas se não existirem usuários com excesso de recursos (ii) ou se não existirem usuários com falta de recursos em relação à sua porção (i). Como a inexistência de usuários com excesso de recursos implica em:

$$\sum fes_k \leq \sum p_k \quad (\text{iii})$$

E ainda que se (i) for falso então (ii) terá que ser verdade para que não exista preempção, então pela expansão de (iii) tem-se que não pode existir usuário k com $fes_k < pk$ e $d_k \geq p_k$ quando os demais usuários não excedem sua porção.

A segunda hipótese é corolário da primeira. Isso porque para a verificação da primeira trabalhou-se com a prova de quando o algoritmo não faz a alocação de um novo recurso. Assim, pela Linha 1:

$$\exists \text{ preempção} \Rightarrow \exists k : k \in \text{usuário} : f_k > 0 \wedge i_k > 0 \quad (\text{iv})$$

O que significa que o sistema fará a preempção apenas se ela for necessária, isto é, existir usuário ocupando menos recursos que sua porção ($f_k > 0$) e que ainda tenha tarefas não alocadas ($i_k > 0$). A verificação da possibilidade de preempção é dada pela Linha 2:

$$\exists \text{ preempção} \Rightarrow \exists k : k \in \text{usuário} : f_k < 0 \quad (\text{v})$$

O que significa que o sistema apenas fará a preempção se ela for possível, ou seja, se existir um usuário ocupando mais recursos do que sua porção ($f_k < 0$). A junção de (iv) e (v) provam portanto a segunda hipótese de avaliação do algoritmo.

3.3 Métricas de Desempenho

Métricas de desempenho são utilizadas para quantificar e mensurar o desempenho de um sistema. Através delas os resultados podem ser avaliados com o intuito de tirar conclusões e gerar mudanças para melhora do sistema. As métricas utilizadas para avaliação do algoritmo OSEP estão relacionadas à utilização dos recursos e ao atendimento dos objetivos da política *Owner-Share*. Na seção seguinte é definido o comportamento ideal esperado pelo usuário para execução das suas tarefas em qualquer sistema. Isso permite definir uma métrica para Satisfação do Usuário, parâmetro essencial para comparação entre políticas de escalonamento. Na sequência são definidos o que pode ser considerado caso ideal para a política OSEP, bem como as métricas para violação da política, perda de capacidade de processamento e o custo originado pela aplicação da política.

3.3.1 Comportamento Ideal Esperado pelo Usuário

Define-se o comportamento ideal para execução de tarefas dos usuários de qualquer sistema como aquele em que todos os recursos do sistema são alocados para o usuário assim que ele submete suas tarefas, ou seja, para o usuário é ideal que ele utilize os recursos do sistema, na quantidade demandada, assim que necessite, alcançando cem por cento de utilização e finalizando suas tarefas o mais rápido possível.

Esse comportamento ideal não leva em consideração a disponibilidade dos recursos e nem o número de usuários do sistema no momento. O único fator que importa é o número de tarefas do usuário, sendo ideal que o usuário utilize o número de recursos necessários para atender sua demanda. Nesse sentido, se o usuário possui um número de tarefas superior ao número de recursos, seu comportamento ideal esperaria que recursos adicionais executassem suas tarefas. Isso é admissível considerando-se que o usuário desconhece o número de recursos no sistema.

Como um exemplo, considere um ambiente com 10 máquinas ou recursos disponíveis, dois usuários u_1 e u_2 que submetem, cada um, 10 tarefas iguais de 300 segundos cada, todas no tempo $t_0 = 50$ s. De acordo com a nossa definição, o ideal para o usuário u_1 seria utilizar as 10 máquinas do sistema para execução das suas 10 tarefas, e o mesmo comportamento ideal é esperado pelo usuário u_2 , não importando se os dois pretendiam usar o sistema ao mesmo tempo. A Figura 12 mostra os gráficos de utilização dos recursos de acordo com o comportamento ideal esperado pelos usuários u_1 e u_2 .

Como mostrado nos gráficos da Figura 12, ambos os usuários iniciam a execução de

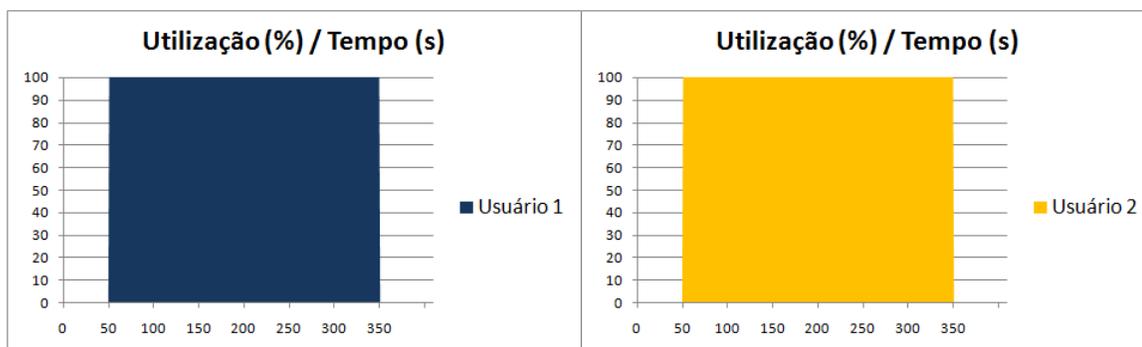


Figura 12: Comportamento ideal para execução de tarefas.

suas 10 tarefas assim que elas são submetidas ao sistema, sem levar em consideração que no sistema há apenas 10 máquinas. O comportamento ideal reflete neste exemplo, um comportamento realmente impossível de ser alcançado para os dois usuários, pois ambos querem acessar todos os recursos do sistema ao mesmo tempo. Por ser o mesmo em qualquer sistema computacional, independente da política de escalonamento adotada, o comportamento ideal nos permite definir uma métrica de desempenho para a Satisfação do Usuário, que pode ser utilizada para comparações com outras políticas de escalonamento.

3.3.2 Satisfação do Usuário

A Satisfação do Usuário (SU) ou *User Satisfaction* é uma métrica de desempenho que mostra quão satisfeito está um determinado usuário com o sistema. Satisfação é algo complexo de definir e de medir, uma vez que muitos fatores podem ser considerados. Qualidade de serviço, resposta rápida e tempo de entrega são os fatores mais comuns que influem na satisfação de um usuário ou cliente em qualquer tipo de sistema.

Baseando-se na nossa definição do comportamento ideal esperado pelo usuário, que deseja utilizar imediatamente os recursos do sistema para execução de suas tarefas, pode-se definir a sua satisfação através de uma relação entre o tempo de entrega das tarefas na situação ideal e na situação real. De acordo com o tamanho da tarefa e o instante em que ela foi submetida, pode-se determinar o instante final esperado para sua finalização. Quanto mais o tempo real de finalização se aproximar do tempo ideal esperado, mais satisfeito estará o usuário. O valor da satisfação do usuário é dado em porcentagem de satisfação, em que 100% indica satisfação máxima.

A satisfação do usuário é calculada para cada tarefa e uma média entre esses valores determina a Satisfação do Usuário com a política de escalonamento adotada pelo sistema em questão. A satisfação do usuário u_i para cada tarefa j ($su_{i,j}$) é definida por:

Definição 3: $su_{ij} = ((t_{j_{ideal}} - t_{j_0}) / (t_{j_{real}} - t_{j_0})) \cdot 100$, onde:

- $t_{j_{ideal}}$ é o instante esperado para o término da tarefa j ;
- $t_{j_{real}}$ é o instante real em que a tarefa j termina sua execução;
- t_{j_0} é o instante de submissão da tarefa.

Em um sistema com m tarefas, a média dos valores de satisfação para cada tarefa determina então, a Satisfação do Usuário com a política de escalonamento do sistema, como descrito na fórmula seguinte:

Definição 4: $SU_i = (\sum_{j=1}^m su_{ij}) / m$.

Considerando ainda o exemplo citado na seção anterior, e uma política de escalonamento qualquer, que faz com que as 10 tarefas do usuário u_1 terminem em $t_1=400$ s, e as do usuário u_2 terminem em $t_2=650$ s. O tempo ideal esperado seria $t_{esperado}=350$ s sendo que as tarefas foram submetidas em $t_0=50$ s e levam 300 s para terminar. Os cálculos da Satisfação do Usuário para u_1 e u_2 seriam então determinados por:

- $su_{1,1...10} = ((350 - 50) / (400 - 50)) \cdot 100 = 85,71\%$;
- $SU_1 = (\sum_{j=1}^{10} 85,71) / 10 = 85,71\%$;
- $su_{2,1...10} = ((350 - 50) / (650 - 50)) \cdot 100 = 50,00\%$;
- $SU_2 = (\sum_{j=1}^{10} 50,00) / 10 = 50,00\%$;

3.3.3 Caso Ótimo para a OSEP

O caso ótimo de comportamento do sistema para a política *Owner-Share* é aquele em que todas as decisões tomadas não geram custos ou atrasos para a infraestrutura e seus usuários. O comportamento ótimo para a política *Owner-Share* é aquele em que se distribui os recursos aos usuários de uma única vez, em uma única iteração e o mais rápido possível, de acordo com a quantidade de recursos requisitada e a correspondente porção do usuário. Baseado nas definições 1 e 2 previamente estabelecidas, é possível determinar o comportamento ótimo para o sistema, e então compará-lo com o comportamento medido em situações reais.

Como um exemplo, considere um ambiente com 10 máquinas ou recursos disponíveis, dois usuários u_1 e u_2 , com porções iguais, 5 para cada um. No tempo t_0 , o primeiro usuário tem 10 tarefas ativas em execução e nenhuma tarefa pronta ou em espera. No tempo t_1 , o usuário u_2 submete 10 tarefas. Antes de t_1 , havia somente um usuário no sistema, u_1 , e ele podia usar todos os recursos.

De acordo com as definições, é possível determinar todas as variáveis do sistema e calcular as porções apropriadas em cada momento do sistema, começando por t_0 :

- Em t_0 :
 - Usuários: u_1 ;
 - Número de recursos: $n=10$;
 - $p_1=5, \hat{p}_1=5$;
 - $d_1=10$;
 - $fes_1=\min\{5 + 5, 10\}=10$;
- Em t_1 :
 - Usuários: u_1 e u_2 ;
 - Número de recursos: $n=10$;
 - $p_1=5, \hat{p}_1=0, p_2=5, \hat{p}_2=0$;
 - $d_1=10$ and $d_2=10$;
 - $fes_1=\min\{5 + 0, 10\}=5$ and $fes_2=\min\{5 + 0, 10\}=5$;

Um sistema real levaria $(t_2 - t_1)$ segundos para alcançar o comportamento adequado da política *Owner-Share*, em que $t_2 > t_1$. Contudo, na situação ótima o sistema deveria atender à política *Owner-Share* instantaneamente e $t_2 = t_1$. A Figura 13 mostra o gráfico de compartilhamento e as porções dos usuários em uma situação ótima de acordo com a política *Owner-Share*.

O número de recursos é distribuído igualmente entre os dois usuários instantaneamente depois da submissão de tarefas pelo usuário u_2 , uma vez que ambos têm a mesma *share* ou porção do proprietário. Os gráficos apresentados na Figura 13 serão úteis para compreensão das métricas de desempenho apresentadas em sequência.

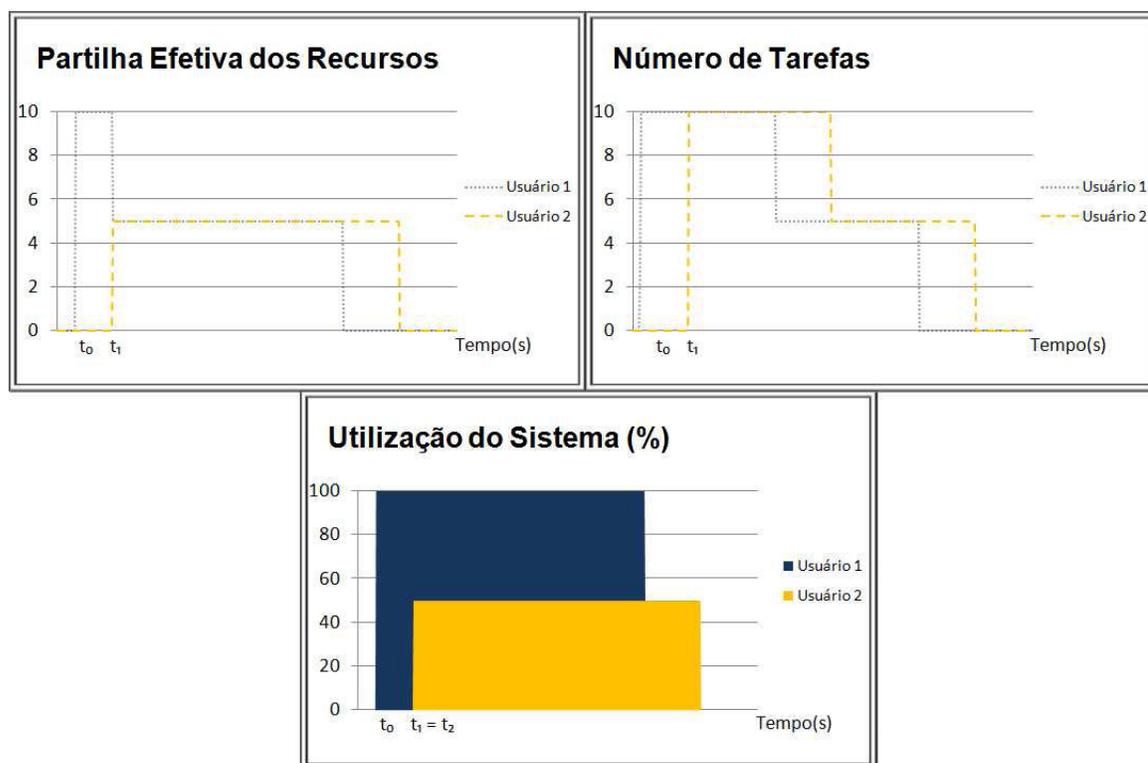


Figura 13: Comportamento ideal para a política OSEP.

3.3.4 Violação de Política

A primeira métrica de desempenho é a Violação de Política ou *Policy Violation* (PV). Essa métrica quantifica a violação da política *Owner-Share* no funcionamento real do sistema quando comparado com o caso ótimo para a política OSEP. O valor ideal para PV é zero, ou seja, o sistema não deveria violar a política *Owner-Share* em nenhum momento.

Considerando mais uma vez o exemplo utilizado para descrição da situação ótima, os gráficos apresentados na Figura 14 mostram o verdadeiro comportamento do sistema com a política OSEP em um sistema distribuído real.

O comportamento real do sistema mostra que $t_2 > t_1$, e cruzando as áreas dos gráficos de utilização do sistema de ambas as situações, ótima e real, é possível visualizar e definir a área correspondente à Violação da Política (PV), em que o usuário u_2 tem tarefas prontas para executar porém não tem os recursos da sua porção de proprietário disponíveis, entre t_1 e t_2 . A PV é representada pela quantidade de tempo por tarefa ou recurso que o sistema leva para atingir a política OSEP e consequentemente a utilização adequada dos recursos. A Figura 15 destaca a área correspondente à violação.

Considerando as curvas das áreas relacionadas à utilização do sistema como SuI_i para

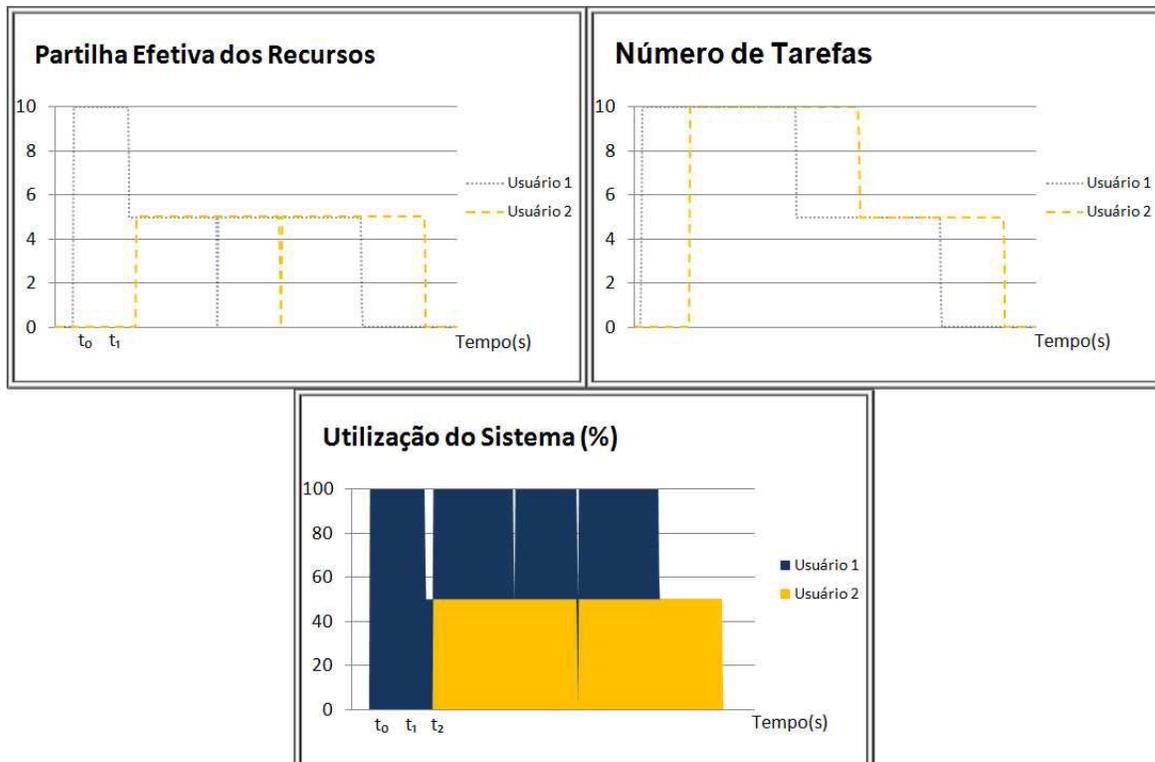


Figura 14: Comportamento real para a política OSEP

o caso ideal e SuR_i para o caso real de um usuário u_i , é possível definir a fórmula que determina a área correspondente à Violação da Política OSEP:

$$\text{Definição 5: } PV_{-u_i} = \int_{t_1}^{t_2} SuI_i - SuR_i.$$

Ainda é importante mencionar que PV é calculada somente para os usuários que estão utilizando um número de recursos menor do que sua porção de proprietário; no exemplo anterior calcula-se a PV somente para o usuário u_2 . Para usuários que estão utilizando mais recursos do que deveriam, deve-se definir outra métrica de desempenho chamada Perda de Capacidade.

3.3.5 Perda de Capacidade

A segunda métrica de desempenho é a chamada Perda de Capacidade ou *Loss of Capacity* (LC), essa métrica é também definida em (JONGH, 2002). Ela quantifica a utilização do sistema que foi perdida devido às ações do algoritmo da política *Owner-Share* e seu valor ideal também é zero. Essa perda de utilização do sistema acontece entre a preempção de tarefas e a alocação de novas tarefas para os recursos liberados.

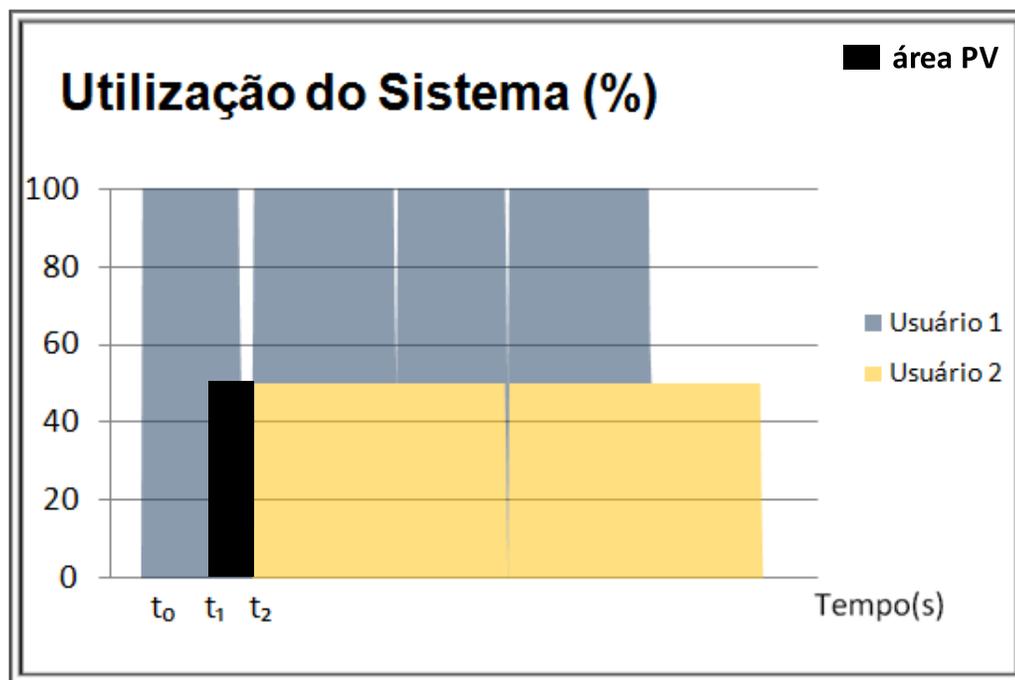


Figura 15: Área PV da violação da política OSEP.

Essa métrica mostra por quanto tempo o sistema desperdiçou capacidade de processamento de um dado recurso, valor representado pela área do gráfico em que a utilização do sistema é menor que 100%. No exemplo utilizado até agora, a área correspondente à LC está localizada no intervalo entre t_1 e t_2 , como destacada na Figura 16.

Para calcular a área LC também se utiliza a equação da Definição 5, mas considerando apenas a região do gráfico correspondente aos usuários com um número de recursos maior do que deveriam ter.

3.3.6 Custo da Política

O Custo da Política ou *Policy Cost*(PC) é a métrica de desempenho que mede o trabalho ou processamento extra ocasionado com a atuação da política. Como pode ser visto como o desperdício de processamento ou ciclos de CPU, seu valor ideal é zero. Comparando-se a área total dos gráficos de utilização para o caso ideal e real é possível calcular o Custo da Política *Owner-Share*, dado por:

Definição 6: $PC = \int_{t_{0_r}}^{t_{f_r}} SuR - \int_{t_{0_i}}^{t_{f_i}} SuI$ em que: - SuR é a curva da utilização medida (real); - SuI é a curva da utilização ideal ou ótima; - t_{0_r} e t_{0_i} são os instantes iniciais medido e ideal respectivamente, e; - t_{f_r} e t_{f_i} são os instantes finais medido e ideal

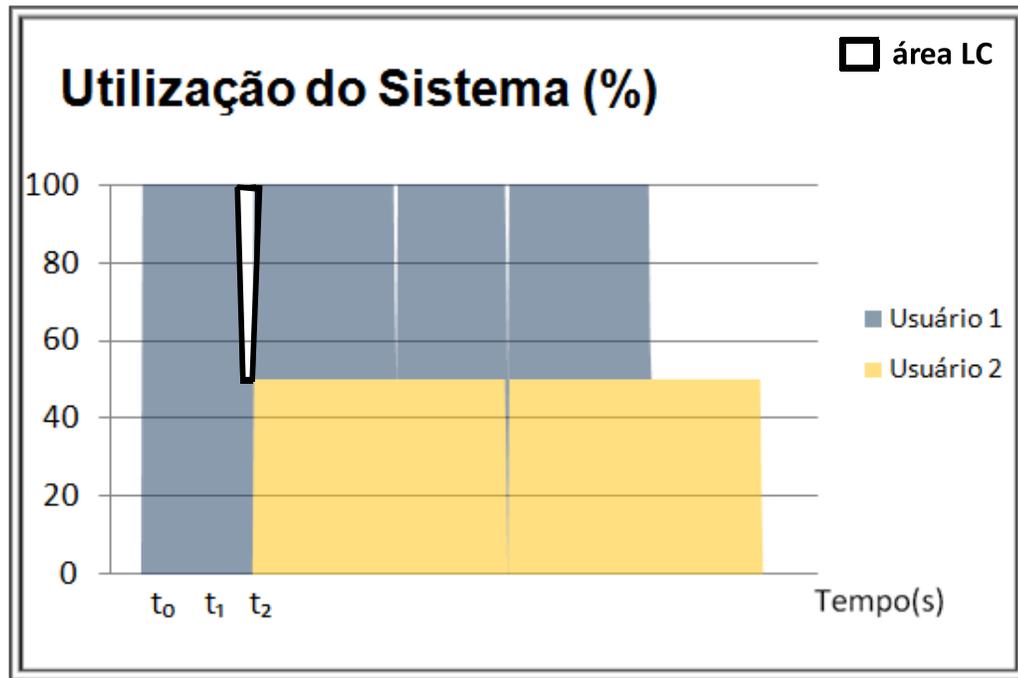


Figura 16: Área LC da perda de capacidade.

respectivamente.

A diferença entre as áreas mostra a quantidade de tempo extra por recurso que foi consumida devido à atuação do sistema OSEP. Esta métrica de desempenho evidencia a diferença entre ambientes que permitem ou não *checkpointing*.

3.4 Checkpointing

Checkpointing é uma técnica para inserção de tolerância à falhas em sistemas computacionais. Basicamente, ela consiste no armazenamento de informações sobre o estado atual da aplicação em determinados eventos (ou instantes), e usa dessas informações para reiniciar a execução da aplicação em um ponto o mais próximo possível daquele em que parou no caso de falha do sistema.

Existem muitos pontos de vista e técnicas diferentes para realização e implementação de *checkpointing*. Ele pode ser implementado diretamente na aplicação pelo próprio programador, com a inserção de pontos de verificação e armazenamento de informações internos ao programa. Algumas ferramentas de escalonamento, enfileiramento, e *Batch Systems* fornecem mecanismos automáticos de *checkpointing*, como é o caso do sistema Condor (LITZKOW et al., 1997). Dessa forma, quando alguma falha acontece ou a ta-

refa é suspensa por algum motivo, a ferramenta automaticamente faz o *checkpointing* da aplicação, salvando todas as informações sobre o estado atual da aplicação e evitando o desperdício do processamento já realizado.

Considerou-se esse processo de *postponing* da execução da tarefa como sendo um *checkpointing* de tarefas. A tarefa interrompida é mantida na tabela de processos e volta à execução assim que for escalonada. Para um sistema distribuído controlado pela política OSEP, o *checkpointing* representa uma grande vantagem para o sistema e para os usuários que devem ceder recursos, pois suas tarefas serão suspensas e o processamento já realizado por elas não será desperdiçado, minimizando assim as métricas de Perda de Capacidade e Custo da Política.

3.5 Implementação do OSEP num Caso Real

Este trabalho foi desenvolvido sobre o sistema Condor, que já fornece um mecanismo de monitoramento de recursos e tarefas, além da estrutura de armazenamento e atualização chamada ClassAds, já citada anteriormente. Algumas novas funcionalidades tiveram que ser criadas e adicionadas ao sistema Condor, a fim de fornecer uma estrutura de ClassAds para os usuários do sistema, pois além das informações dos recursos e tarefas, o algoritmo também precisa das informações dos usuários.

A biblioteca de ClassAds disponível com o sistema Condor permite o armazenamento, atualização e acesso de informações de forma eficiente, além de outras funcionalidades em uma estrutura simples e bem organizada. Por esse motivo e pela eficiência no gerenciamento e monitoramento de recursos e tarefas, escolheu-se o sistema Condor. No entanto, a estrutura fornecida pelo Condor trata do gerenciamento de informações sobre tarefas e recursos apenas, uma vez que as informações sobre os usuários são mantidas de forma independente. As funcionalidades para o tratamento das informações dos usuários e demais funcionalidades adicionadas ao sistema Condor incluem-se em:

- **Estrutura de ClassAds para gerência das informações dos usuários:** a fim de manter o sistema padronizado com a utilização de ClassAds e com a finalidade de tornar o gerenciamento de informações dos usuários mais eficiente e prático, foi desenvolvida uma estrutura de ClassAds para usuários, similar a estrutura já existente para tarefas e recursos.
- **Sistema de atualização de informações dos usuários:** esse sistema foi criado

para a atualização dos parâmetros e variáveis que descrevem um usuário do sistema. Essa atualização é realizada a partir de informações sobre o estado atual das tarefas do usuário.

- **Intercalação dos processos de negociação e atuação do algoritmo:** para o correto funcionamento do algoritmo da política OSEP, cada decisão tomada pelo algoritmo deve ser efetuada na prática pelo próprio processo de negociação do Condor. Assim, para garantir que as decisões feitas pelo algoritmo fossem notadas pelo processo de negociação, esses processos são executados em instantes diferentes, de modo que fiquem intercalados.

O processo de negociação entre recursos e tarefas do Condor é eficiente e rápido, o que permitiu bons resultados de desempenho para a política OSEP. De acordo com o tempo necessário para o processo de negociação do Condor, definiu-se a variável t utilizada pelo **Algoritmo Dinâmico de Atualização** em 20 segundos. Nos testes e resultados apresentados no próximo capítulo, o funcionamento do algoritmo mostrou ser rápido e o equilíbrio desejado é realmente alcançado em questão de segundos. Considerando a complexidade da infraestrutura computacional e o tamanho das tarefas normalmente presentes, valores na escala de segundos representam uma parcela bem pequena de tempo de execução.

O sistema Condor ainda provê o mecanismo de *checkpointing* e migração de tarefas, desejável para a realização de testes com a política OSEP. Essa funcionalidade foi mais um fator determinante para a utilização do Condor na implementação da política.

3.6 Considerações Finais

Neste capítulo foram apresentadas as definições da política de alocação e escalonamento *Owner-Share*, assim como os algoritmos que a implementam e as métricas de desempenho a serem utilizadas para análise e avaliação. No próximo capítulo serão apresentados os testes e resultados obtidos com a avaliação do algoritmo e comparação com a política de escalonamento justo *Fair-Share*.

4 *Testes e Resultados*

Neste capítulo, apresenta-se os resultados da aplicação da política OSEP em diversos cenários, bem como a análise de seu impacto. Inicia-se descrevendo o ambiente e a infraestrutura onde os testes foram feitos. Na Seção 4.2, descreve-se os ajustes de sintonia do algoritmo, feitos considerando as métricas de desempenho descritas no capítulo anterior. Em seguida, descreve-se a avaliação do algoritmo diante de diferentes cenários, organizados de acordo com características, como variação da demanda dos usuários, variação do tamanho das tarefas e variação do número de usuários. Esses testes permitiram fazer comparações entre as políticas OSEP e *Fair-Share*, determinando as vantagens, desvantagens e problemas de cada uma.

4.1 Ambiente de Testes

A infraestrutura computacional utilizada para a realização dos testes da política OSEP constou de uma grade departamental com recursos homogêneos, onde esses recursos estão distribuídos por diversos departamentos da Universidade de Wisconsin em Madison nos Estados Unidos da América. Em conjunto com o grupo de pesquisa responsável pelo sistema Condor, teve-se a oportunidade de implementar a política OSEP em um sistema distribuído real, exposto a todas as condições e problemas de uma infraestrutura computacional amplamente distribuída.

Apesar desse cenário específico é importante salientar que a política OSEP pode ser implementada em qualquer sistema de escalonamento com o auxílio de ferramentas que forneçam as condições necessárias para seu funcionamento. Como exemplo, o sistema precisa de uma ferramenta capaz de monitorar e atualizar informações dos recursos e tarefas em tempo real, além de prover funcionalidades para o controle e alocação dos recursos e tarefas. Pela facilidade de acesso os testes aqui apresentados foram realizados sobre uma implementação da OSEP usando Condor.

Devido ao acesso a um número considerável de recursos espalhados pelos departamentos da Universidade de Wisconsin, criou-se uma grade departamental de acordo com as necessidades para realização dos testes. Fez-se então, as definições das porções dos usuários para estabelecer o conceito de propriedade distribuída e por fim, implementou-se a política de escalonamento OSEP.

Com o intuito de analisar o algoritmo e avaliar suas características, criou-se tipos variados de testes, com diferentes números de usuários e tarefas, além de diferentes tamanhos para as tarefas e diferentes quantidade de recursos no sistema. Em concordância com os tipos de testes criados, definiu-se que o foco da pesquisa deveria estar sobre tarefas do tipo *CPU-bound*, ou seja, tarefas de processamento intensivo. Nesse ambiente ainda realizou-se testes com o algoritmo OSEP em conjunto com a funcionalidade de *checkpointing* disponibilizada pela ferramenta Condor.

4.2 Ajuste do Algoritmo

O algoritmo da política OSEP tem uma funcionalidade interessante relacionada ao parâmetro de atuação do algoritmo. Esse parâmetro determina o número de tarefas ou recursos que devem ser analisados durante cada iteração do algoritmo, ou seja, ele indica quantos recursos são realocados ou quantas tarefas são interrompidas em cada execução do algoritmo. Assim, realizou-se testes variando esse parâmetro com o objetivo de determinar seu valor ótimo de atuação, considerando o relacionamento entre os usuários e suas porções.

4.2.1 Estudo de Caso 1: Ajuste do Parâmetro de Atuação

O ajuste do parâmetro de atuação do algoritmo OSEP foi realizado com a identificação do comportamento das métricas quando se varia o número de recursos movimentados por vez. Nessa análise verifica-se também se esse comportamento é afetado pela variação no número de recursos disponíveis.

Para essa avaliação considerou-se a existência de apenas dois usuários, pois qualquer que seja o número de usuários do sistema, pode-se caracterizar cada iteração do algoritmo como um relacionamento entre dois usuários, aquele que perde recursos e aquele que recebe recursos. Além disso, como o tamanho da tarefa somente influencia a métrica de Satisfação dos Usuários, utilizou-se tarefas de mesmo tamanho (300 s) para ambos os usuários. Por não depender de nenhum outro fator além do tamanho da tarefa, a métrica de Satisfação

do Usuário é a mais adequada para testes de comparação com outras políticas.

Considerou-se também a movimentação sempre segundo três intervalos de quantização, o primeiro com um número mínimo de recursos movimentados, outro com um número médio e outro considerando o total de movimentações necessárias. Como os testes consideraram ambientes com 10, 20 e 40 máquinas, as movimentações de recursos medidas foram (1,2,5), (2,5,10) e (5,10,20) respectivamente. Para cada caso, avaliou-se ainda o impacto do *checkpointing*, cujos testes e resultados estão identificados pela letra Θ .

Em todos os cenários, com 10, 20 e 40 máquinas, tem-se dois usuários donos de 50% dos recursos cada um. Em um primeiro momento o usuário 1 submete tarefas e começa a executá-las em todas máquinas disponíveis do sistema. Em um segundo momento, $t=60$ s, o usuário 2 submete a mesma quantidade de tarefas do usuário 1 ao sistema, desejando utilizar toda sua porção de recursos. A partir do instante t é que o funcionamento do algoritmo deve ser analisado, pois é nesse instante que começa a disputa pelos recursos do sistema.

Os gráficos apresentados nas Figuras 17, 18 e 19 mostram a utilização do sistema (%) ao longo do tempo (s) para cada valor do parâmetro de atuação de acordo com o número de movimentações de recursos, e para cada configuração de ambiente, 10, 20 e 40 máquinas respectivamente. As métricas de desempenho são quantificadas em cada situação e apresentadas em tabelas junto com a análise dos resultados na seção seguinte.

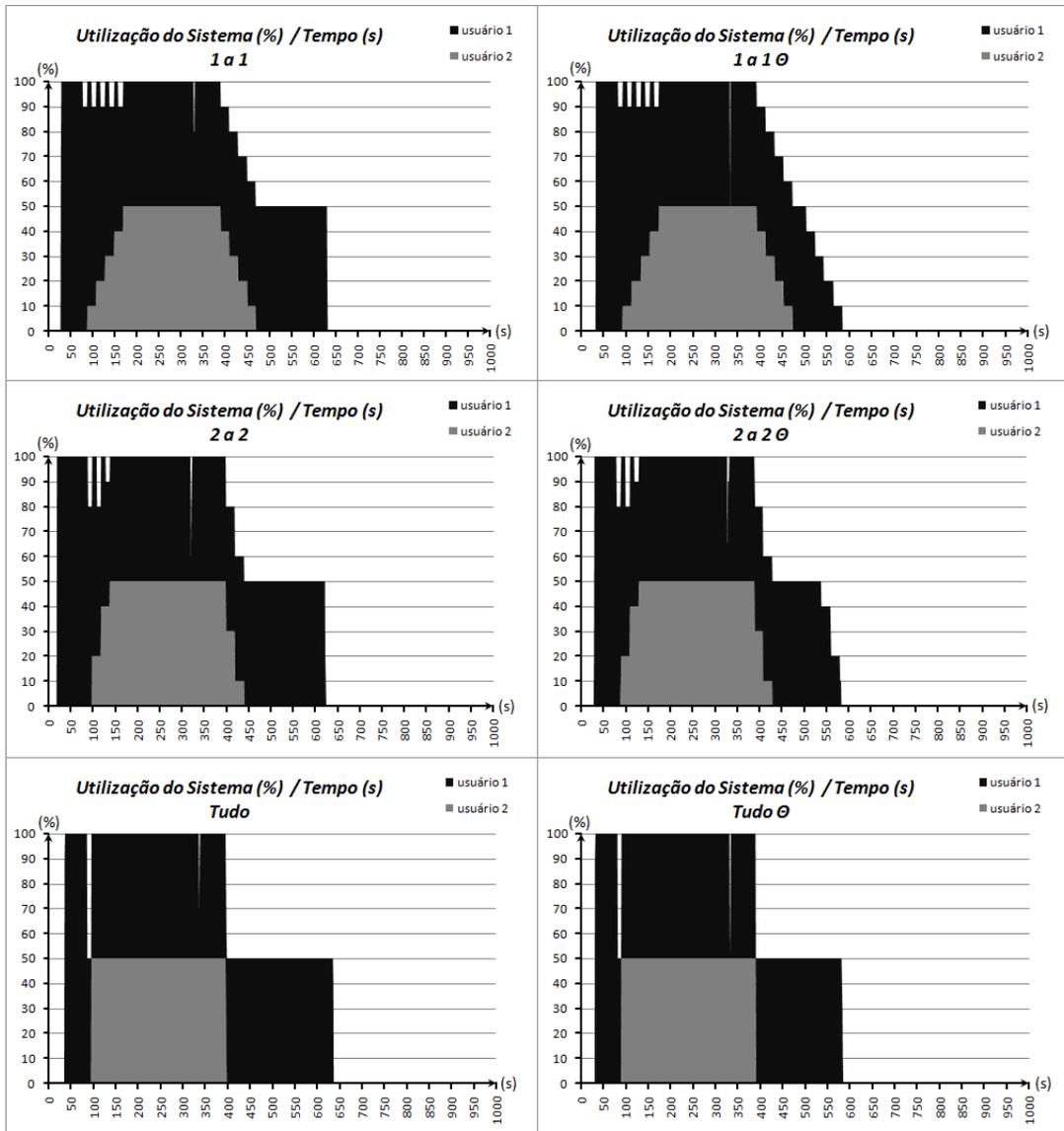


Figura 17: Dois usuários e dez máquinas - Variação do parâmetro de atuação.

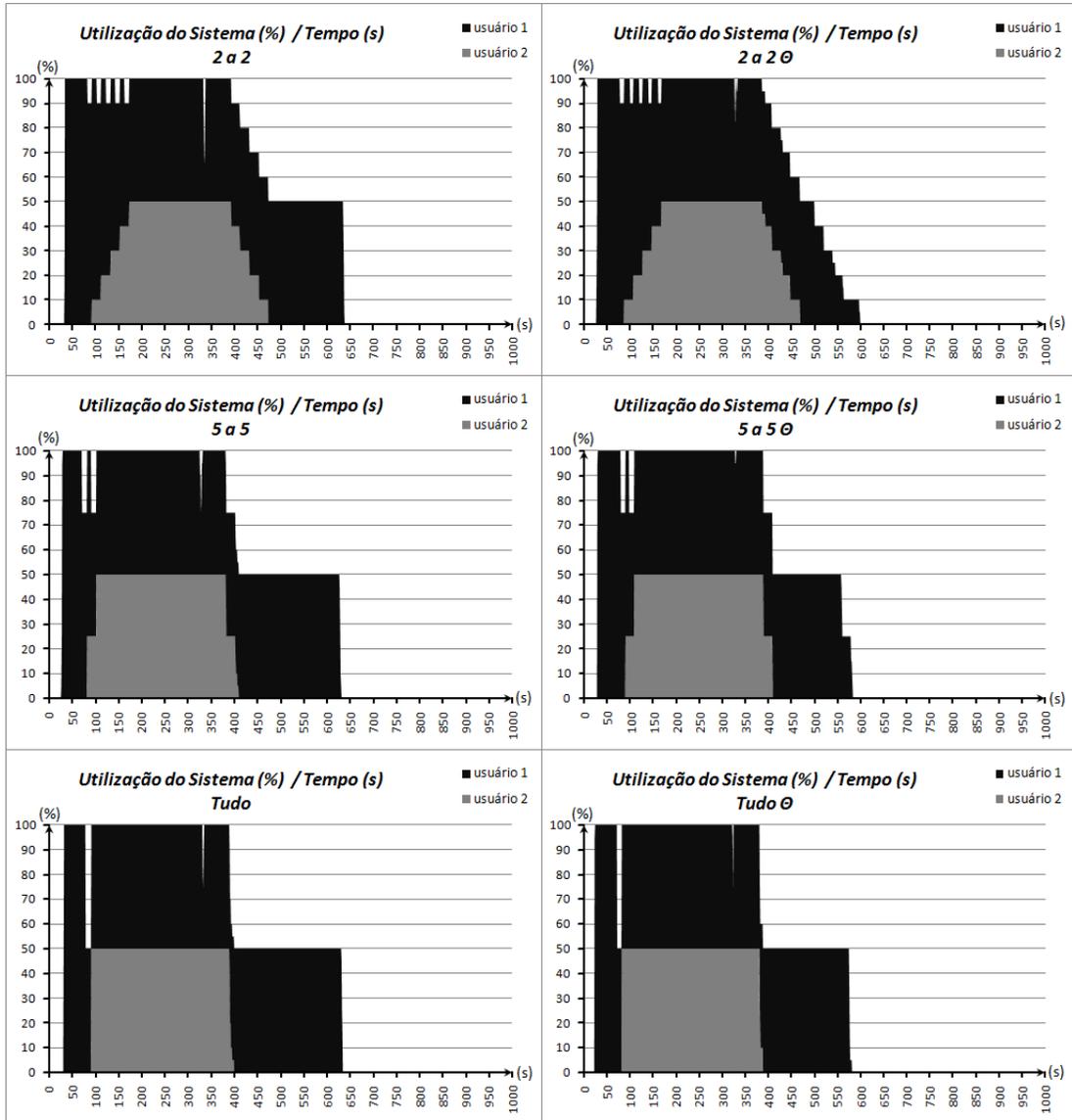


Figura 18: Dois usuários e vinte máquinas - Variação do parâmetro de atuação.

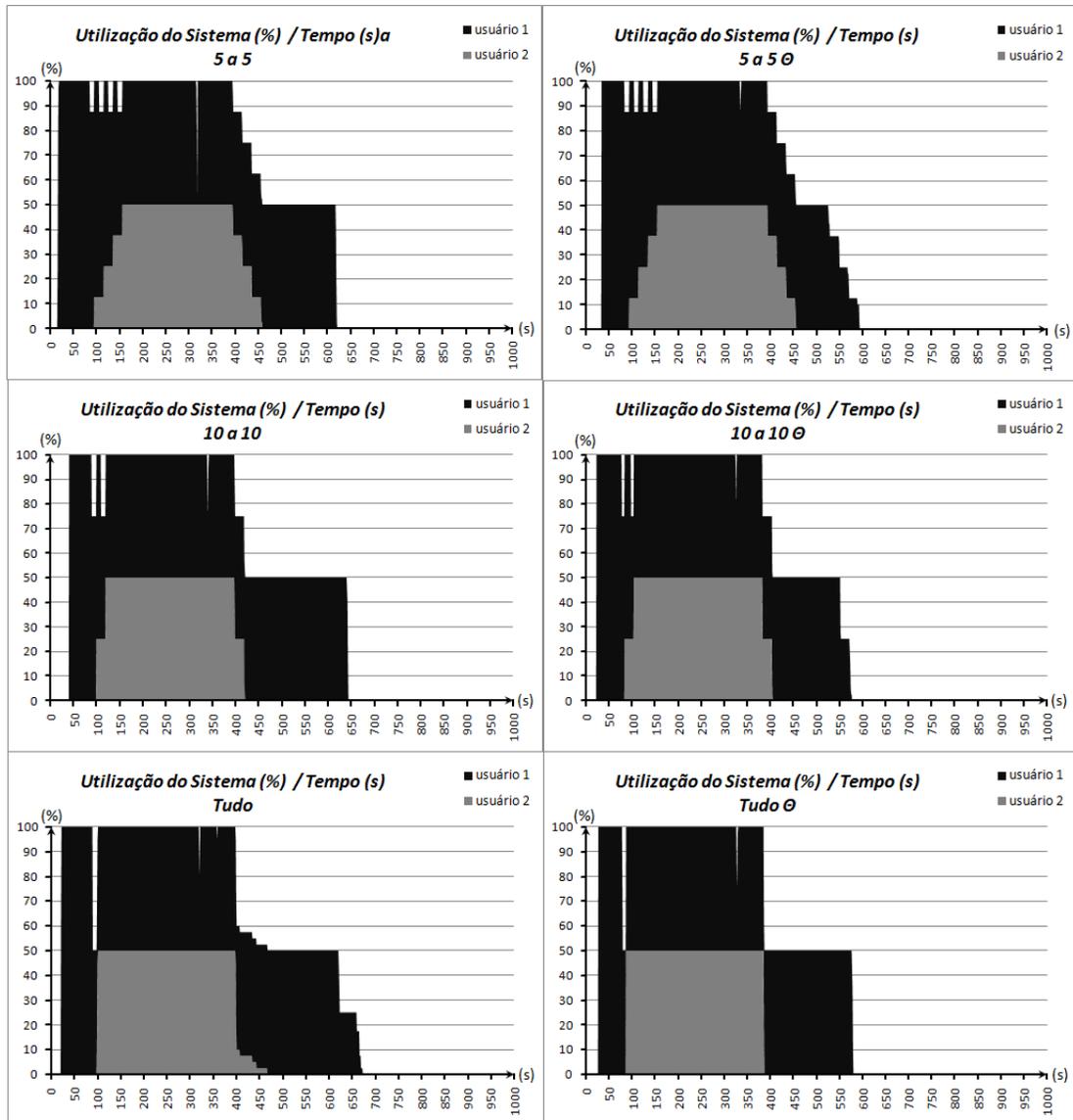


Figura 19: Dois usuários e quarenta máquinas - Variação do parâmetro de atuação.

Análise dos Resultados

As Tabelas 1, 2 e 3 apresentam os resultados obtidos para cada quantidade de recursos no sistema, 10, 20 e 40 máquinas respectivamente. A análise desses resultados permitem determinar alguns fatos característicos da política OSEP e tirar algumas conclusões, descritas nos parágrafos seguintes.

Tabela 1: Resultados das métricas de desempenho para 10 máquinas.

10 Máquinas	1	1 Θ	2	2 Θ	Tudo	Tudo Θ
Violação da Política	3.370	3.630	2.650	2.240	1.750	1.450
Perda de Capacidade	520	490	450	490	450	400
Custo da Política	4.650	70	4.470	50	2.680	20
Satisfação do Usuário 1	69,12%	72,51%	70,72%	71,91%	67,56%	70,67%
Satisfação do Usuário 2	84,09%	83,30%	86,85%	88,65%	90,27%	92,01%

Tabela 2: Resultados das métricas de desempenho para 20 máquinas.

20 Máquinas	2	2 Θ	5	5 Θ	Tudo	Tudo Θ
Violação da Política	3.580	3.310	1.600	1.925	1.500	1.050
Perda de Capacidade	480	370	550	525	600	350
Custo da Política	4.585	160	2.795	140	2.510	120
Satisfação do Usuário 1	68,48%	72,12%	69,56%	71,83%	68,74%	71,46%
Satisfação do Usuário 2	83,30%	84,45%	91,42%	90,23%	91,63%	94,03%

Tabela 3: Resultados das métricas de desempenho para 40 máquinas.

40 Máquinas	5	5 Θ	10	10 Θ	Tudo	Tudo Θ
Violação da Política	3.237,5	3.175	2.450	1.650	1.935	1.300
Perda de Capacidade	487,5	487,5	500	500	485	370
Custo da Política	5.002,5	302	3.057,5	200	4.935	197,5
Satisfação do Usuário 1	71,4%	71,45%	67,15%	72,68%	69,29%	71,6%
Satisfação do Usuário 2	84,56%	85,04%	87,87%	91,61%	88,42%	93,2%

Entre os resultados mostrados na Tabelas 1, 2 e 3, pode-se identificar os mais relevantes, como a grande diferença entre os valores de Custo da Política com e sem a funcionalidade de *checkpointing*. Os valores do Custo da Política com *checkpointing* são baixos, bem próximos do ideal, o que confirma que com essa funcionalidade, o processamento feito por uma tarefa realmente não é perdido quando ela é interrompida. Existe apenas um processamento extra para a realização do *checkpointing* de uma tarefa, um *overhead* considerado pequeno quando comparado com o tamanho da tarefa e o Custo da Política nos casos sem *checkpointing*.

Ainda em relação ao Custo da Política, outro fato que chama a atenção é o aumento do seu valor com a variação da quantidade de recursos no sistema para os casos com *checkpointing*. Apesar de pouca diferença, esses valores para o Custo da Política crescem de uma tabela para outra, e esse fato é explicado pela existência de mais tarefas no sistema, o que causa um maior gasto de processamento para a realização do *checkpointing*. Para os

casos sem *checkpointing* o aumento do número de tarefas não apresenta nenhum padrão de comportamento para o Custo da Política, no entanto, pode-se notar seu valor superior justificado pelo desperdício de processamento das tarefas interrompidas.

A métrica de Perda de Capacidade não apresentou resultados relevantes em nenhum caso, sendo que os valores permaneceram muito próximos. Essa proximidade nos permite concluir que não importam o parâmetro de atuação do algoritmo e nem a quantidade de recursos no sistema, ter-se-á sempre tempos ociosos de utilização muito próximos, necessários para a troca de contexto e realocação de recursos. Já os resultados da métrica de Violação da Política, comprovaram uma característica óbvia e fácil de ser observada, quanto mais rápido o algoritmo fornecer os recursos que deve para o usuário em questão, por menos tempo ele violará a política OSEP e, portanto, a métrica terá um valor menor.

Os valores para Satisfação do Usuário 1 não diferiram consideravelmente entre todas as situações. Entretanto, conseguiu-se notar o aumento da satisfação desse usuário, proporcionado pelo *checkpointing* de tarefas. Já para o usuário 2, o aumento do número de movimentações de recursos representou uma melhoria na sua satisfação, pois, ele recebe seus recursos mais rápido.

Considerando todas as configurações de ambientes, 10, 20 e 40 máquinas, e os resultados obtidos, pode-se concluir que o melhor parâmetro para atuação do algoritmo da política OSEP para estes testes é aquele em que as tarefas são interrompidas todas de uma vez, e os recursos realocados em uma única iteração. Com esse parâmetro e a funcionalidade de *checkpointing*, o sistema alcança os menores valores para as métricas de Violação da Política, Perda de Capacidade e Custo da Política, e também os maiores valores para a Satisfação do Usuário 2. A diferença de menos de 1% entre os resultados para a Satisfação do Usuário 1 na quinta linha da quinta e sétima coluna da Tabela 3, nos casos em que o *checkpointing* de tarefas é utilizado, é irrelevante na determinação do melhor parâmetro de atuação.

Finalmente, pode-se concluir que em um ambiente de compartilhamento de recursos baseado na política OSEP composto de usuários amparados pelo conceito de propriedade distribuída, o algoritmo de atuação deve fornecer toda a porção de recursos de um determinado usuário de uma única vez, não importando a quantidade de recursos ou tarefas presentes no sistema.

4.3 Variação da Demanda dos Usuários

A variação da demanda entre os usuários do sistema é um fator de avaliação para as políticas OSEP e *Fair-Share*. Essa avaliação é importante para determinar a eficiência da política OSEP em garantir a porção dos usuários na disputa por recursos em diferentes situações de demanda.

Para realização desses testes, montou-se uma infraestrutura computacional composta de 20 máquinas, e definiu-se a existência de dois usuários proprietários de 50% desses recursos cada um. A quantidade de usuários ou o tamanho das tarefas não são parâmetros de análise nesses testes. Assim, a existência de dois usuários com tarefas de mesmo tamanho cria um ambiente satisfatório para análise. Aqui, também, as tarefas de ambos os usuários são tarefas independentes que executam por 300 s cada uma.

Os testes dessa seção estão organizados em três estudos de caso: Demanda Menor ou Igual à Porção dos Usuários, Demanda Maior que a Porção, Usuários com Demandas Diferentes. Nos três estudos de caso, fez-se ainda comparações entre resultados obtidos com a política *Fair-Share* e a política OSEP.

4.3.1 Estudo de Caso 2: Demanda Menor ou Igual à Porção

No cenário em que a demanda de cada usuário é menor ou igual à sua porção, tanto o algoritmo da política OSEP quanto o da política *Fair-Share* não têm decisões complexas a fazer. As tarefas são distribuídas entre os recursos do sistema, e como a demanda é menor que a porção, nenhuma negociação para a realocação de recursos é necessária, pois sobram recursos livres no sistema. Como não há disputa por recursos, os resultados obtidos nesse estudo de caso não nos permitiram fazer nenhuma afirmação sobre a eficiência da política OSEP em garantir a porção dos usuários.

Os gráficos da Figura 20 e a Tabela 4 apresentam os resultados obtidos para testes em um cenário em que a demanda de cada usuário é inferior à sua porção. No ambiente construído com 20 máquinas e na presença de dois usuários proprietários de 50% dos recursos cada um, ou seja, 10 máquinas, cada um desses usuários submete ao sistema uma quantidade de tarefas inferior a 10. No caso aqui apresentado, definiu-se que cada usuário submeteria ao sistema apenas 5 tarefas, o primeiro usuário no instante $t = 0$ s e o segundo em $t = 60$ s.

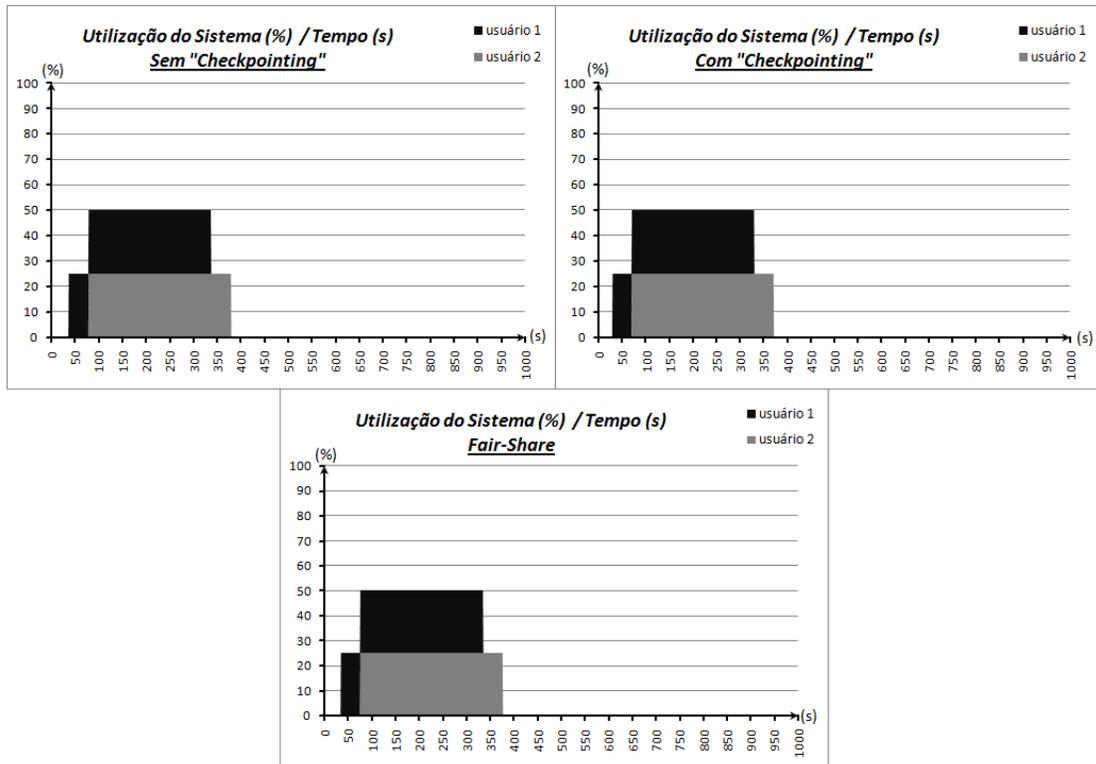


Figura 20: Demanda menor que a porção - 5 tarefas por usuário.

Tabela 4: Resultados dos testes de demanda menor que a porção - 5 tarefas.

20 Máquinas	OSEP	OSEP Θ	<i>Fair-Share</i>
Violação da Política	450	250	-
Perda de Capacidade	0	0	-
Custo da Política	80	50	-
Satisfação do Usuário 1	88,76%	90,63%	89,02%
Satisfação do Usuário 2	94,74%	96,77%	95,47%

Nos gráficos da Figura 21 e na Tabela 5, apresenta-se os resultados dos testes para o cenário em que a demanda é exatamente igual à porção dos usuários. Como cada usuário possui 10 recursos do sistema, cada um deles submete ao sistema 10 tarefas, o primeiro usuário em $t = 0$ s e o segundo em $t = 60$ s.

As métricas de desempenho de Violação da Política, Perda de Capacidade e Custo da Política são definidas para a política OSEP, assim, as tabelas apresentadas não possuem os resultados dessas métricas para a política *Fair-Share*.

Nos dois casos, com demanda menor e igual à porção, os resultados não apresentaram nenhum fato relevante para avaliação das políticas OSEP e *Fair-Share*, devido ao fato de não haver disputa por recursos. Cada usuário obtém os recursos de acordo com a demanda requisitada, e nenhuma negociação de realocação de recursos e interrupção de

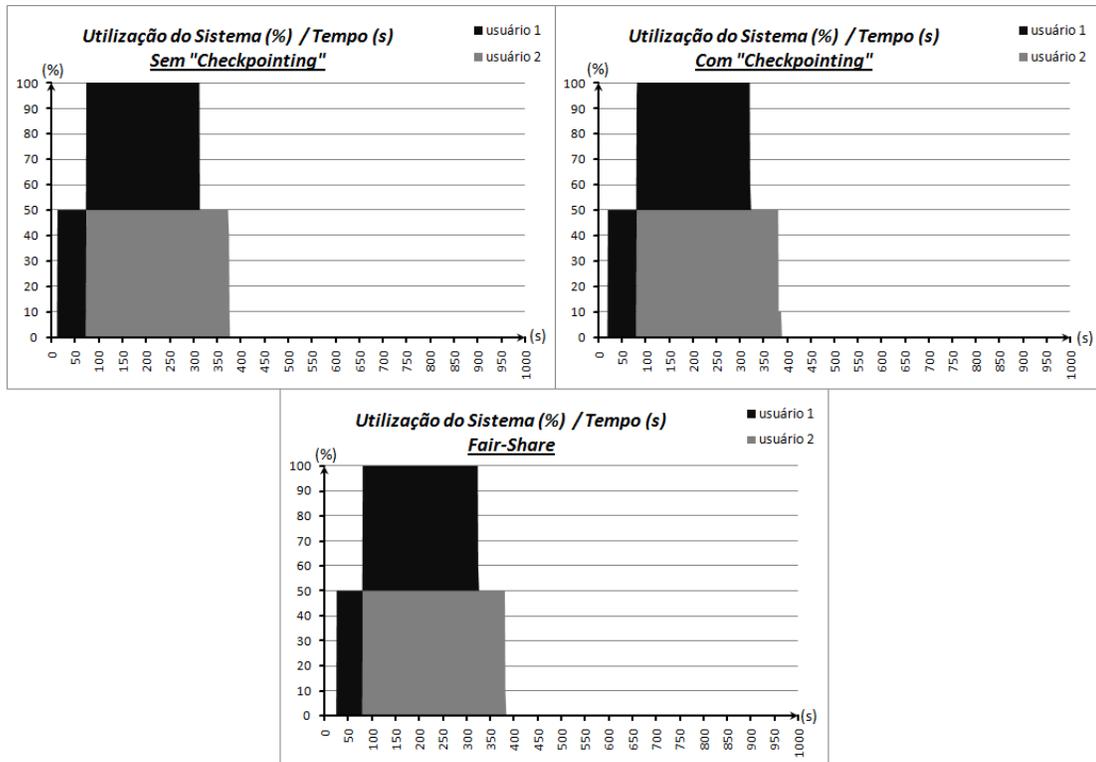


Figura 21: Demanda igual à porção - 10 tarefas por usuário.

Tabela 5: Resultados dos testes de demanda igual à porção - 10 tarefas

20 Máquinas	OSEP	OSEP Θ	<i>Fair-Share</i>
Violação da Política	650	800	-
Perda de Capacidade	0	0	-
Custo da Política	145	225	-
Satisfação do Usuário 1	95,3%	93,08%	92,14%
Satisfação do Usuário 2	95,67%	93,93%	94,17%

tarefas é necessária. Mesmo sem a disputa por recursos entre os usuários, tem-se um resultado correspondente à Violação da Política ocasionado pelo intervalo de tempo de iteração e negociação do algoritmo. Como não há realocação de recursos, a métrica de perda de processamento não é quantificada. Já a métrica de Custo da Política apresenta valores justificados pela carga de processamento presente nos recursos localmente, o que causa um atraso na execução das tarefas. Os altos valores das satisfações dos usuários indicam que esse atraso e os valores de Custo da Política, causados pela carga local dos recursos, não são relevantes.

Os resultados para a Satisfação dos Usuários mostraram valores muito próximos entre os testes. Suas diferenças são causadas pela relação entre o instante de submissão de tarefas e o intervalo de iteração do algoritmo. A submissão pode ser feita exatamente

antes, durante ou depois de uma iteração do algoritmo e, dependente disso, os recursos podem ser alocados na atual iteração do algoritmo ou na seguinte, causando essa pequena diferença. Na implementação feita, o intervalo entre iterações do algoritmo é de cerca de 20 s, o que para tarefas de longa duração é considerado um intervalo pequeno, e por isso proporciona valores próximos para os resultados de satisfação dos usuários.

4.3.2 Estudo de Caso 3: Demanda Maior que a Porção

Dois conjuntos de testes foram realizados para o cenário em que a demanda de cada usuário é maior que sua porção, o primeiro reflete o cenário em que demanda é duas vezes maior que a porção do usuário. Já o segundo conjunto de testes caracteriza uma demanda de tarefas três vezes maior que a porção do usuário. Essas testes foram realizados com o objetivo de obter resultados suficientes para uma comparação precisa entre as políticas OSEP e *Fair-Share*.

O ambiente de testes é o mesmo dos testes anteriores, um sistema com 20 máquinas e dois usuários proprietários de 50% dos recursos cada um. Assim, como cada usuário possui 10 máquinas, no primeiro conjunto de testes cada usuário submete ao sistema 20 tarefas e, no segundo, cada um submete 30 tarefas. As tarefas são todas iguais e executam por 300 s. Como nos testes anteriores, o usuário 1 submete suas tarefas no instante $t = 0$ s e o usuário 2 em $t = 60$ s.

20 Tarefas por Usuário

A Figura 22 mostra os gráficos da utilização do sistema ao longo do tempo para os dois usuários, nos três cenários de análise: OSEP sem *checkpointing*, OSEP com *checkpointing* (representado pelo símbolo Θ) e *Fair-Share*. Na Tabela 6 apresenta-se também os resultados dos testes para as métricas de desempenho e satisfação dos usuários nos mesmos três cenários.

Os resultados apresentados na Tabela 6 mostram a eficiência da política OSEP quando aplicada junto com a funcionalidade de *checkpointing*. O Custo da Política diminui consideravelmente sem o desperdício do processamento realizado antes da interrupção de tarefas. Da mesma forma, vê-se uma melhora para a Satisfação do Usuário 1 com a utilização de *checkpointing* para a política OSEP. Logicamente, conclui-se que, se o usuário 2 submetesse suas tarefas em um instante t em que $t_f > t > 60$ s e t_f é o instante em que as

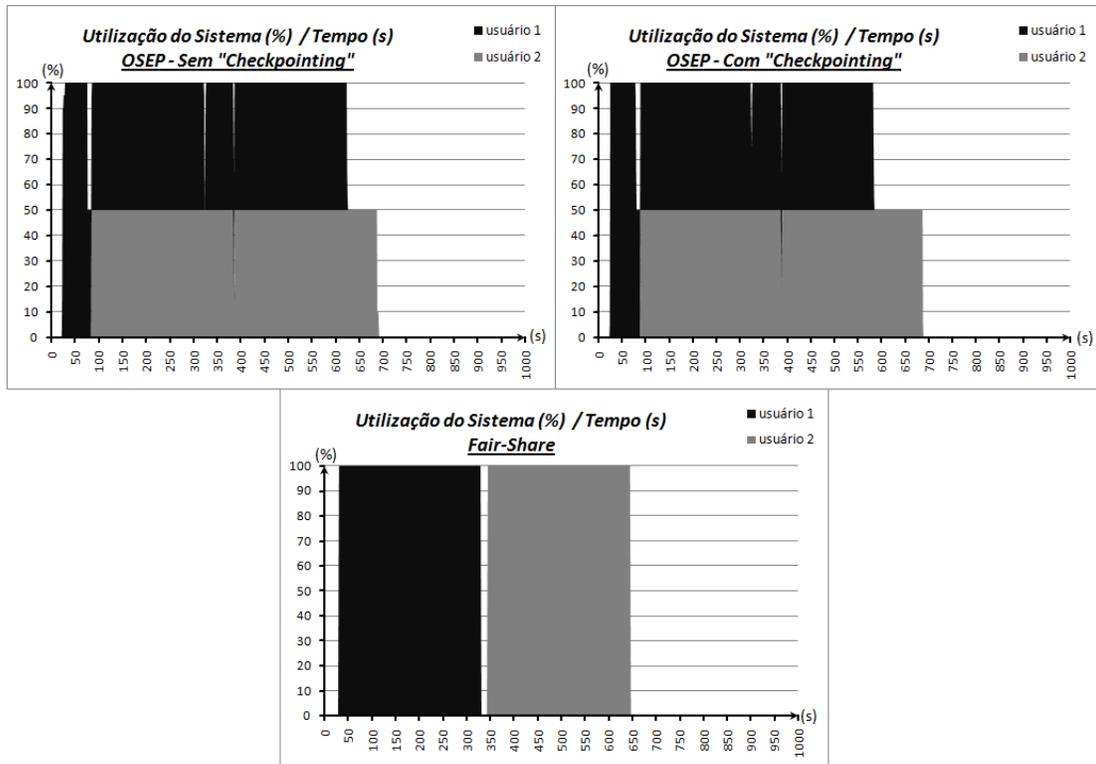


Figura 22: Demanda maior que a porção - 20 Tarefas por Usuário.

Tabela 6: Resultados dos testes de demanda maior que porção - 20 tarefas.

20 Máquinas	OSEP	OSEP Θ	<i>Fair-Share</i>
Violação da Política	1.405	1.400	-
Perda de Capacidade	445	405	-
Custo da Política	2.745	605	-
Satisfação do Usuário 1	70,28%	71,87%	90,39%
Satisfação do Usuário 2	72,85%	72,34%	55,68%

tarefas do usuário 1 terminam, o processamento realizado antes da interrupção das tarefas seria maior, causando o aumento do Custo da Política e a diminuição da Satisfação do Usuário 1, para o caso em que a política OSEP foi aplicada sem *checkpointing*. Já para a política OSEP com *checkpointing*, a Satisfação do Usuário 1 seria maior ainda, pois faltaria menos processamento para a finalização das tarefas interrompidas.

A métrica de Satisfação dos Usuários nos permite comparações entre as políticas OSEP e *Fair-Share*. Conseguiu-se uma boa porcentagem de satisfação para ambos usuário 1 e 2 com a aplicação da política OSEP, valores acima de 70%. No entanto, para a política *Fair-Share*, a Satisfação do Usuário 1 é ótima, acima de 90%, e a Satisfação do Usuário 2 é mediana, em torno de 56%, valores destacados em **negrito** na última coluna da tabela.

O fato importante a ser avaliado com esses resultados é a justiça entre os usuários,

pois o fato do usuário 1 ter submetido seus trabalhos 60 segundos antes do usuário 2 é determinante para a satisfação mediana do usuário 2 e a ótima satisfação do usuário 1, na política *Fair-Share*. Diante da nossa visão de justiça, o usuário 2 deveria receber sua porção dos recursos assim que necessário, proporcionando uma boa satisfação para ambos os usuários, com valores que estão acima de 70% para a OSEP sem ou com *checkpointing*.

30 Tarefas por Usuário

Os resultados para os testes com demanda três vezes maior que a porção do usuário são apresentados nos valores da Tabela 7 e nos gráficos da Figura 23. Para a política OSEP conseguiu-se valores para a satisfação dos usuários próximos a 60%, já para a política *Fair-Share* o usuário 1 tem uma satisfação em torno de 74% e o usuário 2 em torno de 49%, valores destacados em **negrito** na tabela.

Tabela 7: Resultados dos testes de demanda maior que porção - 30 tarefas.

20 Máquinas	OSEP	OSEP Θ	<i>Fair-Share</i>
Violação da Política	1.550	1.600	-
Perda de Capacidade	410	680	-
Custo da Política	1.935	250	-
Satisfação do Usuário 1	55,28%	58,29%	74,02%
Satisfação do Usuário 2	60,86%	59,42%	49,33%

O gráfico de utilização da política *Fair-Share*, apresentado na Figura 23, mostra a compensação na utilização dos recursos ao longo do tempo. A política *Fair-Share* defende que a justiça entre os usuários do sistema é alcançada através da compensação na utilização dos recursos em longo prazo, procurando pela igualdade na utilização do sistema entre os usuários. Mesmo com a compensação na utilização dos recursos, a política *Fair-Share* apresenta uma satisfação de mais de 74% para o usuário 1, porém penaliza muito o usuário 2, que tem uma satisfação de 49%.

Com a política OSEP, os dois usuários alcançam valores de satisfação bem próximos e acima da média, em torno de 60%. Assim, nos casos de demanda maior que a porção e de acordo com a política OSEP, defende-se que a justiça entre os usuários é representada pela proximidade entre os valores de satisfação dos usuários, se esses usuários tiverem valores de porções iguais ou bem próximos.

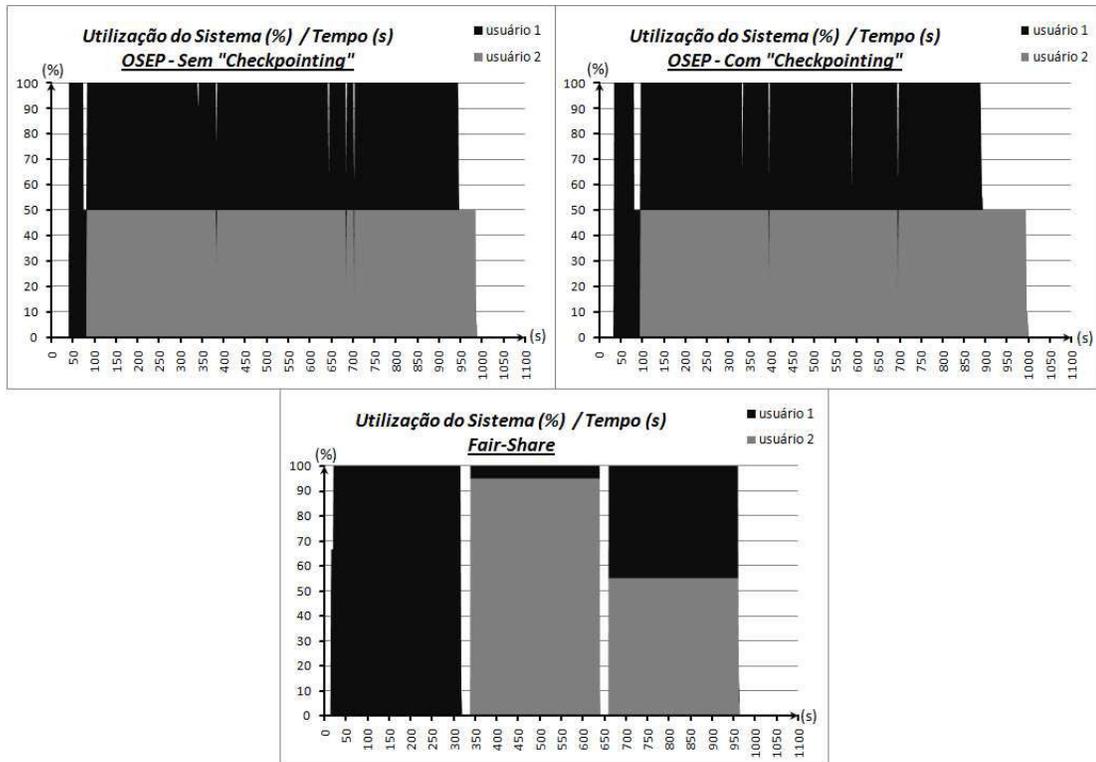


Figura 23: Demanda maior que a porção - 30 tarefas por usuário.

4.3.3 Estudo de Caso 4: Usuários com Demandas Diferentes

Os testes realizados anteriormente nos permitiram tirar conclusões específicas para os casos em que os usuários têm a mesma demanda. Desse modo, nesse terceiro estudo de caso avaliamos o impacto da OSEP para o caso em que os usuários do sistema têm diferentes demandas ou quantidades de tarefas.

Situações em que há uma pequena diferença entre a demanda dos usuários não trazem nenhuma informação nova e relevante, pois os resultados são muito próximos aos apresentados anteriormente. Assim, apresenta-se a situação em que os resultados dos testes têm maior relevância, ou seja, aquela em que um dos usuários tem uma demanda muito alta e o outro tem uma demanda baixa.

Usando a mesma infraestrutura dos testes anteriores, com 20 máquinas e porção de 50% para cada usuário, o usuário 1 possui uma demanda de vinte tarefas, já o usuário 2 de apenas duas tarefas. Os instantes de submissão de ambos os usuários permanecem os mesmos. Os gráficos da Figura 24 apresentam a utilização do sistema nos dois cenários da política OSEP, e também na política *Fair-Share*.

Os resultados da Tabela 8 mostram que a política OSEP alcança uma ótima satisfa-

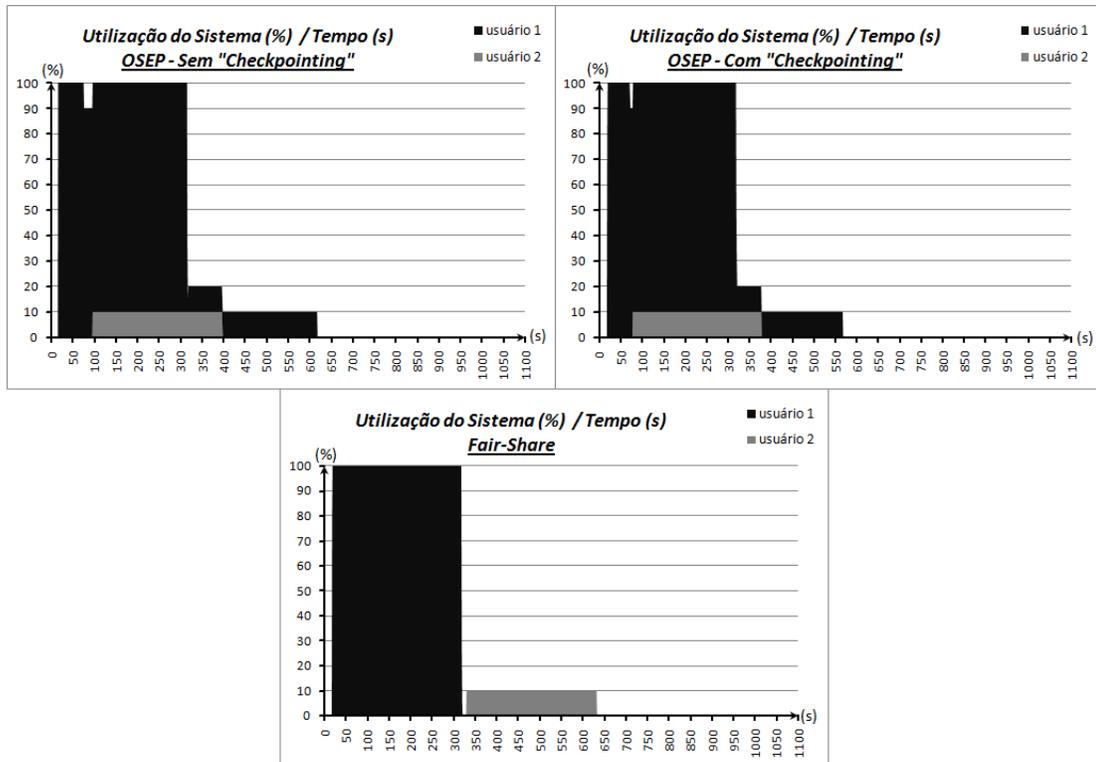


Figura 24: Usuários com demandas diferentes.

Tabela 8: Resultados dos testes entre os usuários com demandas diferentes.

20 Máquinas	OSEP	OSEP Θ	<i>Fair-Share</i>
Violação da Política	350	170	-
Perda de Capacidade	200	65	-
Custo da Política	645	135	-
Satisfação do Usuário 1	90,02%	90,4%	93,66%
Satisfação do Usuário 2	90,57%	94,48%	56,77%

ção para ambos os usuários, acima de 90%, e a política *Fair-Share* alcança uma ótima satisfação para o usuário 1, mas uma satisfação mediana para o usuário 2, em torno de 56%.

A política *Fair-Share* define como justa a igualdade de utilização dos recursos no longo prazo. No entanto, para um curto prazo a política *Fair-Share* pode causar uma diferença muito grande entre as satisfações dos usuários. Como visto pelos resultados até agora apresentados, a política OSEP procura eliminar as diferenças de curto prazo.

4.4 Variação no Tamanho das Tarefas

A variação no tamanho das tarefas é mais um parâmetro importante para análise e comparação das políticas OSEP e *Fair-Share*. Os testes feitos até agora foram realizados com tarefas de curta duração, apenas 300 s. Assim, o objetivo nesses testes é determinar o impacto de variação no tamanho das tarefas no desempenho dos algoritmos e na satisfação dos usuários do sistema.

Para realização desses testes, manteve-se a mesma infraestrutura computacional composta de 20 máquinas e dois usuários proprietários de 50% dos recursos cada um. A quantidade de usuários ainda não é um parâmetro de análise nesses testes, e como se quer uma situação de disputa e negociação entre os usuários, conduziu-se os testes sob um sistema com alta demanda de tarefas dos usuários.

Os testes dessa seção estão organizados em quatro estudos de caso diferenciados pelo tamanho das tarefas em questão. Para tanto, definiu-se a seguir os limites do parâmetro tempo de execução para as tarefas de curta, média e longa duração:

- Tarefas de Curta Duração: tarefas de até 600 s (10 min) de execução.
- Tarefas de Média Duração: tarefas com tempo de execução t , em que $600 < t < 3.600$ s.
- Tarefas de Longa Duração: tarefas com tempo de execução igual ou maior que 3.600 s (1 hora).

4.4.1 Estudo de Caso 5: Tarefas de Curta Duração

Neste caso, utilizou-se tarefas de 600 s e cada usuário tendo uma demanda três vezes maior que sua porção, ou seja, 30 tarefas cada um. A fim de destacar a diferença entre os resultados para a política OSEP sem *checkpointing* e com *checkpointing*, definiu-se uma situação em que o segundo usuário submete suas tarefas 300 s após o primeiro usuário.

Os gráficos da Figura 25 apresentam a utilização dos recursos do sistema em porcentagem ao longo do tempo, para cada política em questão, OSEP sem *checkpointing*, com *checkpointing* e *Fair-Share*.

Nos gráficos da política OSEP, pode-se notar que o usuário 2 recebe sua porção de recursos após o tempo de 300 s, que é quando ele submeteu suas tarefas. Já para a

política *Fair-Share*, não há interrupção imediata de tarefas do usuário 1, mas sim uma compensação posterior na utilização dos recursos. Assim, as tarefas do usuário 2 começam a executar somente após o término das tarefas do usuário 1, por volta dos 600 s.

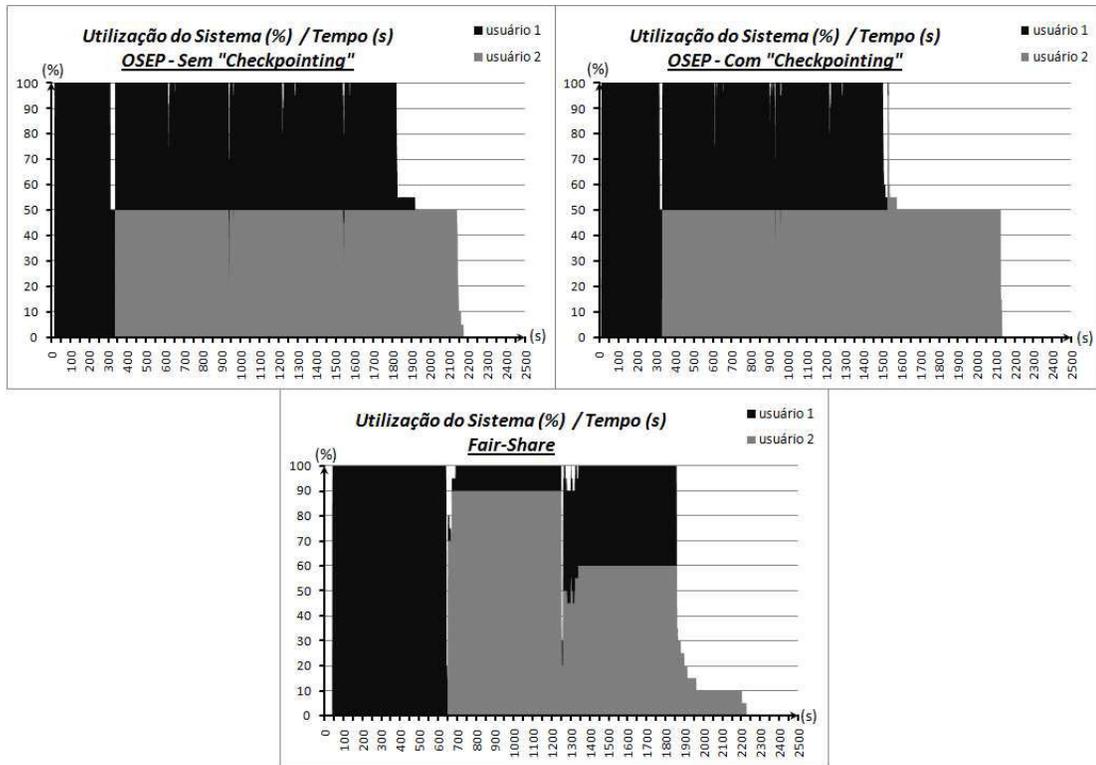


Figura 25: Tarefas de curta duração - 600 s.

Nos resultados de satisfação dos usuários, destacados na Tabela 9, e na comparação entre os níveis apresentados nos gráficos da Figura 26, pode-se notar uma melhora de quase 10% de satisfação do usuário 1, para a política OSEP com *checkpointing* em comparação com a OSEP sem *checkpointing*. Como os usuários têm mesma porção de recursos e demanda de tarefas, a satisfação dos usuários na política OSEP tem valores muito próximos. Já para a política *Fair-Share* vemos novamente uma grande diferença entre a satisfação dos usuários.

Tabela 9: Resultados dos testes com tarefas de curta duração

20 Máquinas	OSEP	OSEP Θ	<i>Fair-Share</i>
Violação da Política	1.850	1.605	-
Perda de Capacidade	1.200	640	-
Custo da Política	15.995	650	-
Satisfação do Usuário 1	59,38%	68,06%	73,62%
Satisfação do Usuário 2	65,24%	65,66%	61,49%

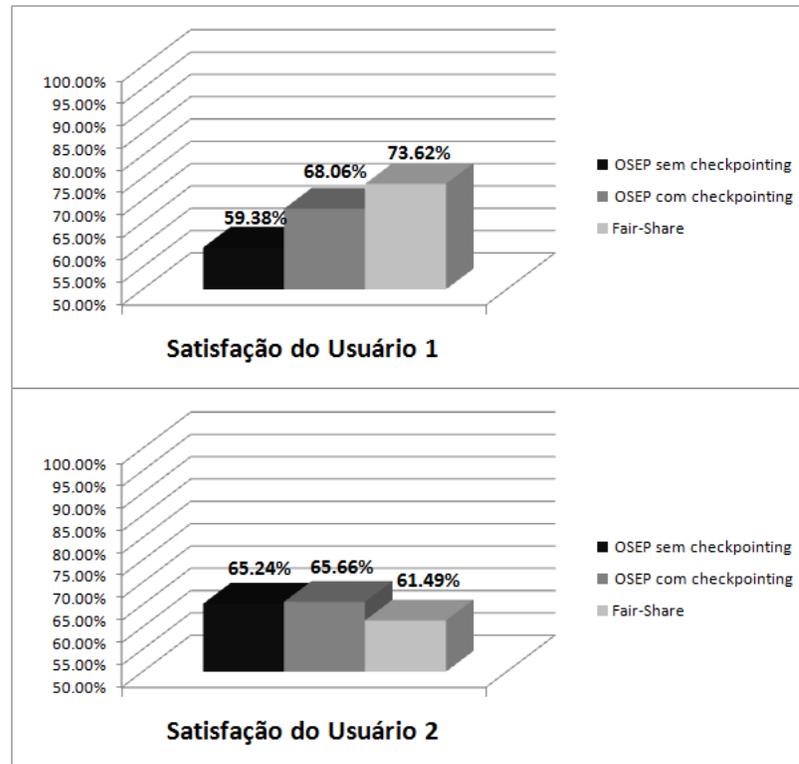


Figura 26: Comparação da satisfação dos usuários - Tarefas de curta duração.

4.4.2 Estudo de Caso 6: Tarefas de Média Duração

Nesse segundo estudo de caso com variação no tamanho das tarefas, utilizou-se tarefas de 1.800 s ou 30 min, e cada usuário tem uma demanda três vezes maior que sua porção, ou seja, 30 tarefas cada um. Nesses testes, o segundo usuário submete suas tarefas 300 s após o primeiro usuário.

A Tabela 10 mostra os resultados obtidos com os testes e a Figura 27 apresenta os gráficos da utilização do sistema ao longo do tempo. O único fato a ser notado nesses resultados é a queda da satisfação do usuário 2 com a política *Fair-Share* em relação aos testes para tarefas de curta duração. Como as tarefas do usuário 1 demoram mais para finalizar, o usuário 2 tem sua satisfação mais prejudicada.

Tabela 10: Resultados dos testes com tarefas de média duração.

20 Máquinas	OSEP	OSEP Θ	<i>Fair-Share</i>
Violação da Política	1.270	1.700	-
Perda de Capacidade	615	775	-
Custo da Política	38.925	4.310	-
Satisfação do Usuário 1	58,2%	60,3%	76,44%
Satisfação do Usuário 2	61,03%	62,32%	49,31%

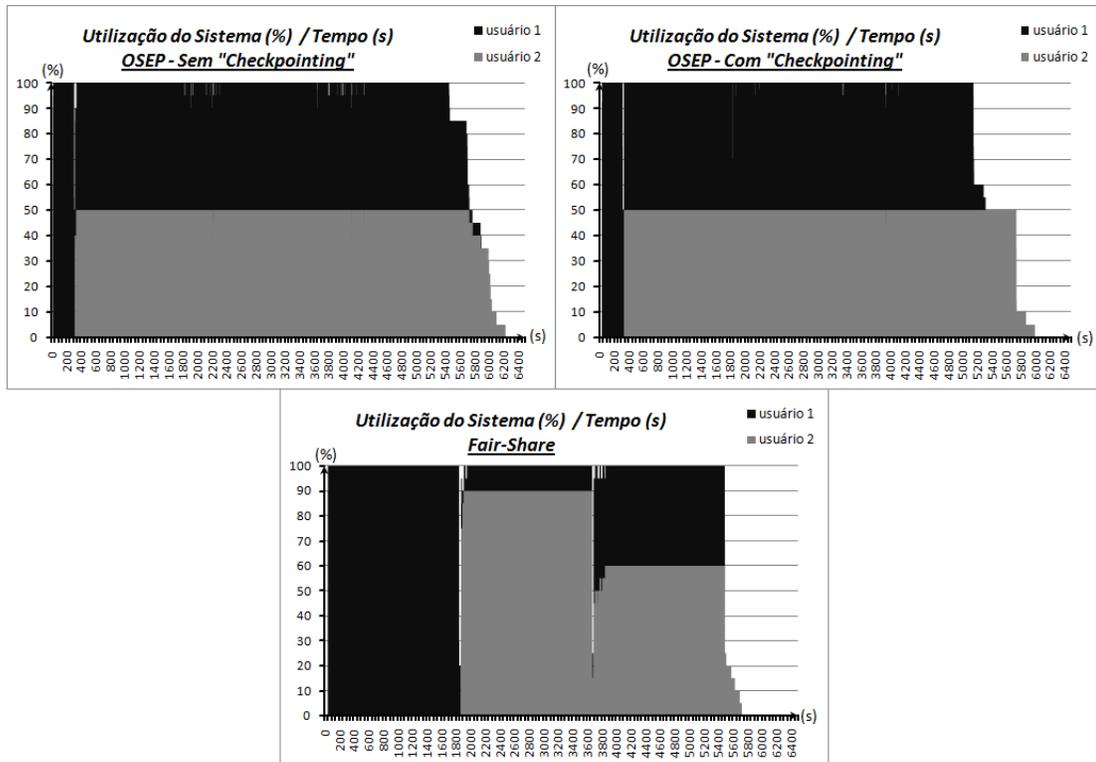


Figura 27: Tarefas de média duração - 1.800 s.

4.4.3 Estudo de Caso 7: Tarefas de Longa Duração

No terceiro estudo de caso com variação no tamanho das tarefas, utilizamos tarefas de 3.600 s ou 1 h. Da mesma forma dos testes anteriores, cada usuário tem uma demanda três vezes maior que sua porção e o segundo usuário submete suas tarefas 300 s após o primeiro usuário. A Figura 28 apresenta os gráficos da utilização do sistema ao longo do tempo.

A Tabela 11 apresenta os resultados obtidos e, assim como nos casos anteriores, nota-se a queda da satisfação do usuário 2 com a política *Fair-Share*. Nos resultados da política OSEP vê-se um alto valor de Custo da Política, mesmo para o caso com *checkpointing*. Esse valor é explicado pelo atraso durante a execução da tarefa devido à carga local nos recursos. Como as tarefas são de longa duração, elas encontram, por mais vezes, os recursos carregados localmente, o que causa lentidão na execução e no *checkpointing* da tarefa. Portanto, um tempo extra, proporcional ao atraso causado pela carga local do recurso, é necessário para finalizar a execução da tarefa.

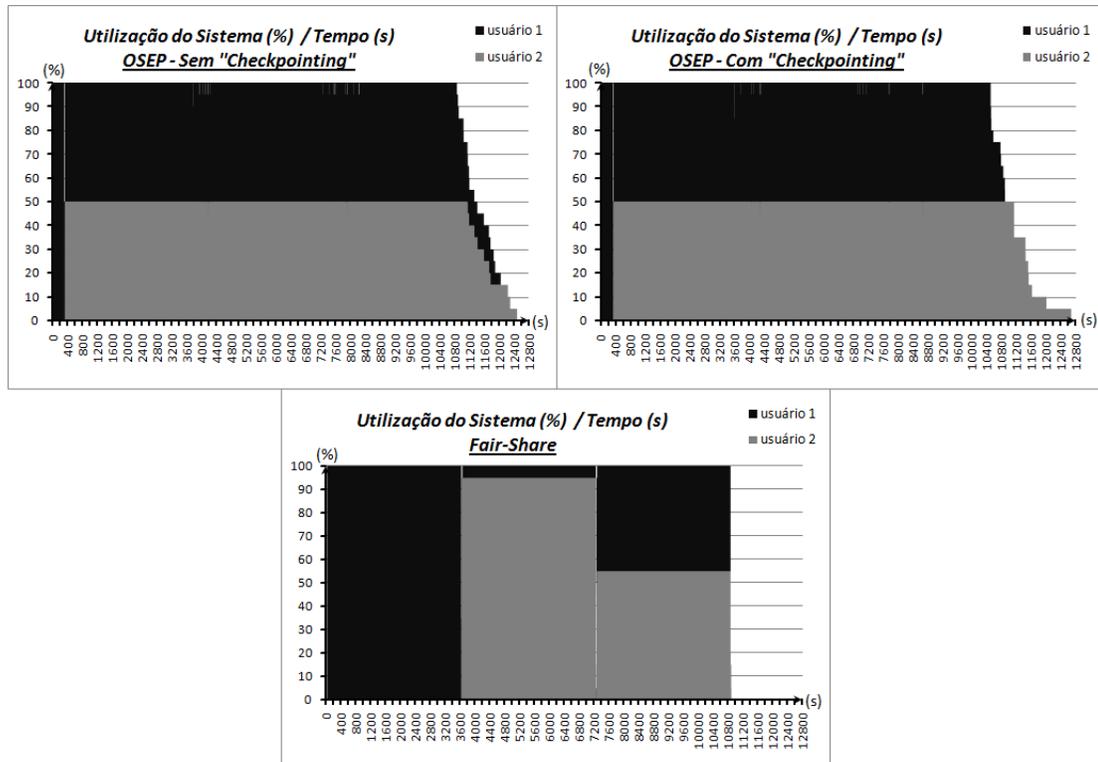


Figura 28: Tarefas de longa duração - 3.600 s.

Tabela 11: Resultados dos testes com tarefas de longa duração.

20 Máquinas	OSEP	OSEP Θ	<i>Fair-Share</i>
Violação da Política	1.710	1.470	-
Perda de Capacidade	860	725	-
Custo da Política	68.625	21.760	-
Satisfação do Usuário 1	58,31%	60,85%	77,42%
Satisfação do Usuário 2	60,39%	59,71%	47,17%

4.4.4 Estudo de Caso 8: Tarefas de Duração Variada

Nesse quarto e último estudo de caso com variação no tamanho das tarefas, apresenta-se três situações mais realistas em um ambiente em que as tarefas dos usuários são de tamanhos variados. Na primeira situação, os usuários continuam com a mesma demanda de tarefas, 30 tarefas cada um, porém as tarefas têm tempos de execução diferentes. Da mesma forma que os testes anteriores, o segundo usuário submete suas tarefas 300 s após o usuário 1.

A Tabela 12 apresenta os resultados obtidos e a Figura 29 mostra os gráficos de utilização do sistema ao longo do tempo. Os resultados confirmaram o esperado e a tendência vista até agora, com a política *Fair-Share* apresentando uma diferença muito

grande entre a satisfação dos usuários.

Tabela 12: Resultados dos testes com tarefas de duração variada.

20 Máquinas	OSEP	OSEP Θ	<i>Fair-Share</i>
Violação da Política	1.755	2.150	-
Perda de Capacidade	805	1.240	-
Custo da Política	4.095	1.075	-
Satisfação do Usuário 1	62,55%	64,6%	76,41%
Satisfação do Usuário 2	65,02%	66,7%	54,65%

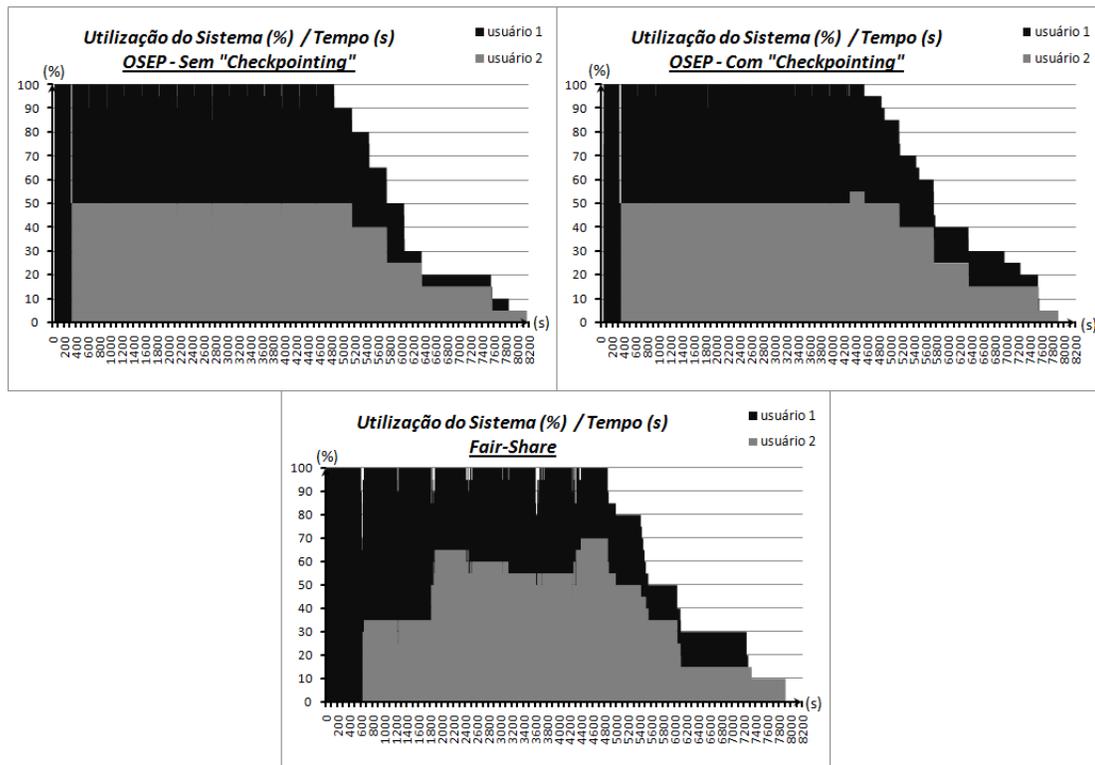


Figura 29: Tarefas de duração variada - Entre 600 e 3.600 s.

Ainda considerando este quarto estudo de caso, com tarefas de duração variada, apresenta-se agora uma situação em que as tarefas do usuário 1 têm o mesmo tamanho, 3.600 s, porém as tarefas do usuário 2 executam por somente 600 s cada uma. Esse cenário permite uma melhor comparação entre as políticas de escalonamento, pois, tipicamente, a variação do tamanho das tarefas acontece entre os usuários, com cada usuário executando programas diferentes entre si, mas com várias reexecuções do mesmo programa por um dado usuário. Apresenta-se na Figura 30, os gráficos de utilização do sistema ao longo do tempo e, na Tabela 13, as estatísticas correspondentes.

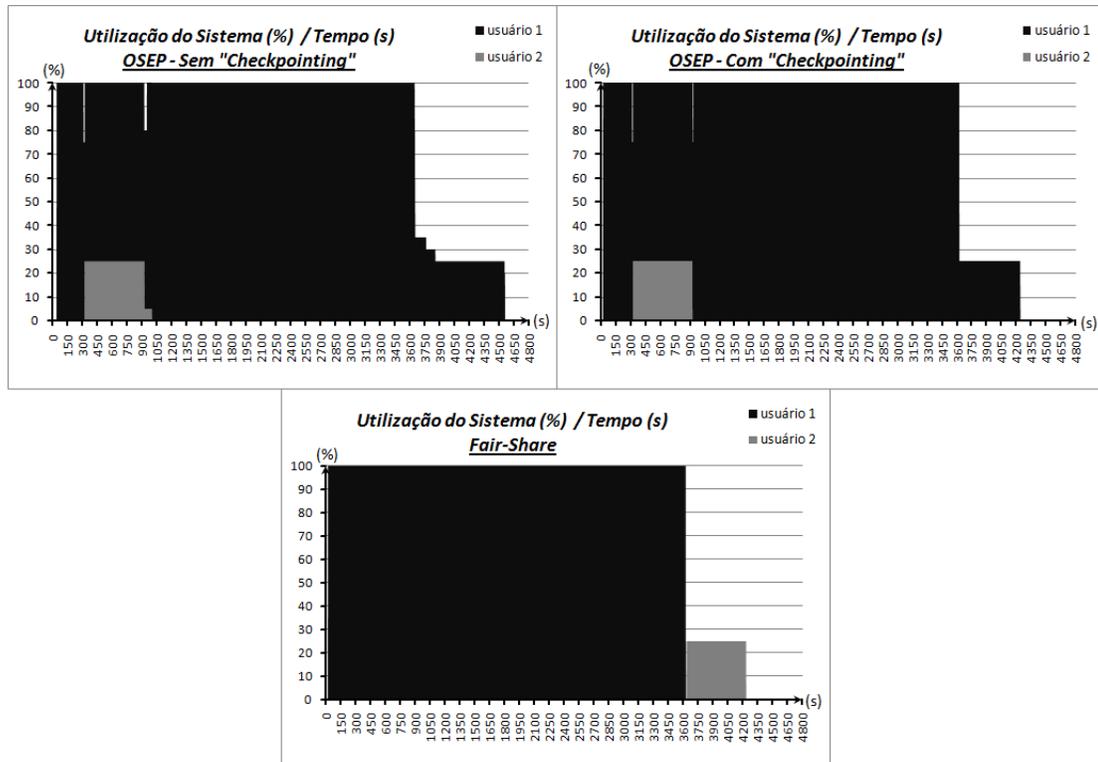


Figura 30: Tarefas de duração variada - Cenário com tarefas maiores primeiro.

Tabela 13: Resultados dos testes com tarefas de duração variada - Cenário com tarefas maiores primeiro.

20 Máquinas	OSEP	OSEP Θ	<i>Fair-Share</i>
Violação da Política	600	550	-
Perda de Capacidade	275	195	-
Custo da Política	9.845	625	-
Satisfação do Usuário 1	93,21%	95,66%	99,26%
Satisfação do Usuário 2	95,63%	97,28%	21,24%

Como se pode notar nos níveis de satisfação apresentados na Figura 31, a política *Fair-Share* apresenta uma satisfação ótima para o usuário 1 mas muito ruim para o usuário 2, enquanto a política OSEP apresenta ótima satisfação para ambos os usuários, tanto com como sem *checkpointing*.

A terceira e última situação apresentada é o cenário inverso do anterior, ou seja, as tarefas do usuário 1 têm o mesmo tamanho, porém elas executam por 600 s, e as tarefas do usuário 2 executam por 3.600 s cada uma. Os resultados desse teste estão apresentados na Tabela 14 e na Figura 32.

Nota-se nessa situação que a política *Fair-Share* apresenta bons níveis de satisfação para ambos usuários, com valores acima de 90%. É interessante observar que embora esse

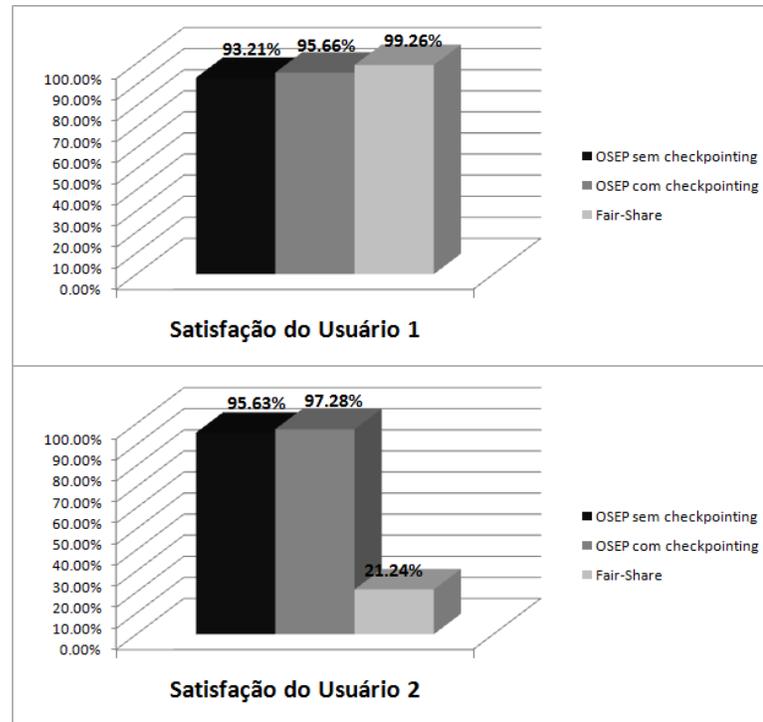


Figura 31: Comparação da satisfação dos usuários - Tarefas de duração variada.

Tabela 14: Resultados dos testes com tarefas de duração variada - Cenário com tarefas menores primeiro.

20 Máquinas	OSEP	OSEP Θ	<i>Fair-Share</i>
Violação da Política	800	850	-
Perda de Capacidade	450	425	-
Custo da Política	8.030	540	-
Satisfação do Usuário 1	85,72%	88,65%	96,54%
Satisfação do Usuário 2	98,85%	98,95%	91,96%

cenário tende a favorecer a política *Fair-Share*, seu desempenho foi apenas ligeiramente superior ao da OSEP. No cenário anterior, favorável a OSEP, mostrou um desempenho marcadamente inferior do *Fair-Share*.

4.5 Testes com Maior Quantidade de Usuários

Apresenta-se nesta seção, os testes realizados com uma maior quantidade de usuários. Dividiu-se os testes em dois casos especiais, o primeiro com 4 usuários com porções iguais de recursos e o segundo com 4 usuários com porções diferentes.

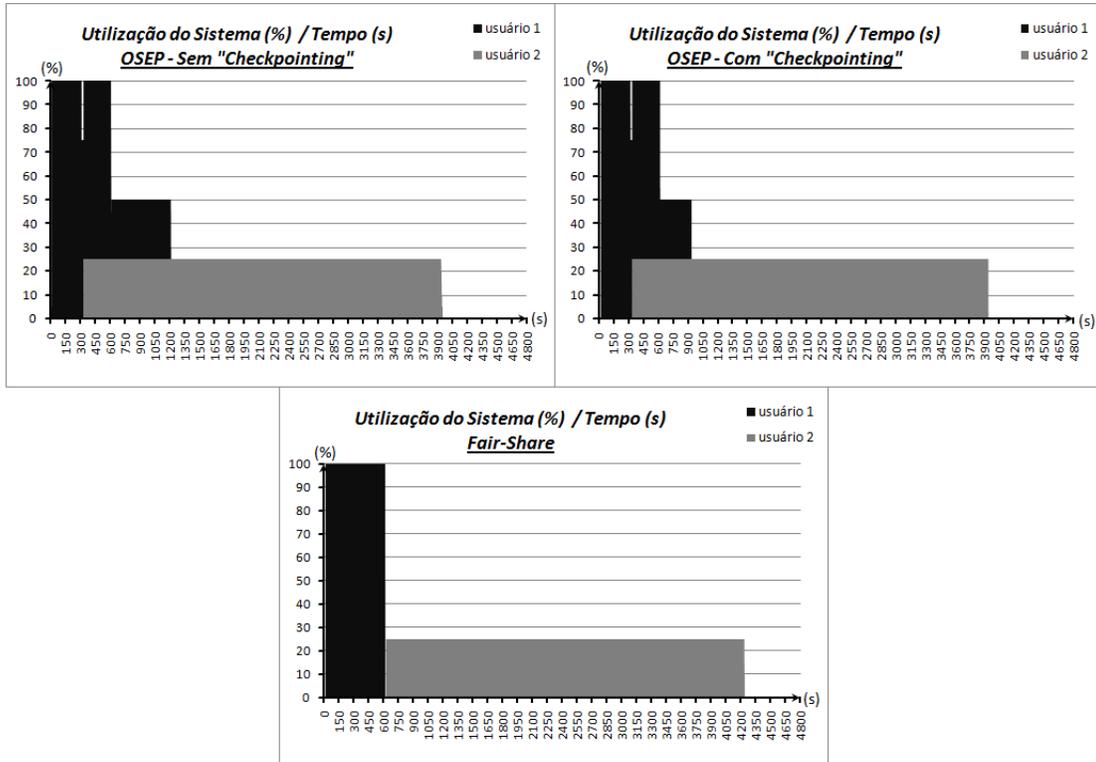


Figura 32: Tarefas de duração variada - Cenário com tarefas menores primeiro.

4.5.1 Estudo de Caso 9: 4 usuários e porções iguais

O primeiro estudo de caso reflete a situação em que 4 usuários submetem tarefas ao sistema e disputam por recursos para execução de suas tarefas. Definiu-se assim nosso primeiro cenário de testes:

- Sistema com 20 máquinas ou recursos;
- 4 usuários com porções iguais, 25% ou 5 máquinas para cada um;
- 15 tarefas por usuário, tarefas de mesmo tamanho (600 s cada uma);
- $t_1 = 0$ s, $t_2 = 100$ s, $t_3 = 200$ s e $t_4 = 210$ s, onde t_x é o instante de submissão das tarefas do usuário x ;

Os gráficos da Figura 33 mostram a utilização do sistema ao longo do tempo para as políticas OSEP e *Fair-Share*. Nota-se que, com a política OSEP, as tarefas do usuário 1 são suspensas para a realocação dos recursos para os outros usuários tão logo o sistema nota a chegada de outras tarefas. Já, para a política *Fair-Share*, podemos notar que os usuários 3 e 4 somente recebem recursos após as tarefas do usuário 1 terem finalizado. O

usuário 2 recebe alguns recursos logo de início pois ainda haviam alguns recursos livres no instante em que ele submete suas tarefas.

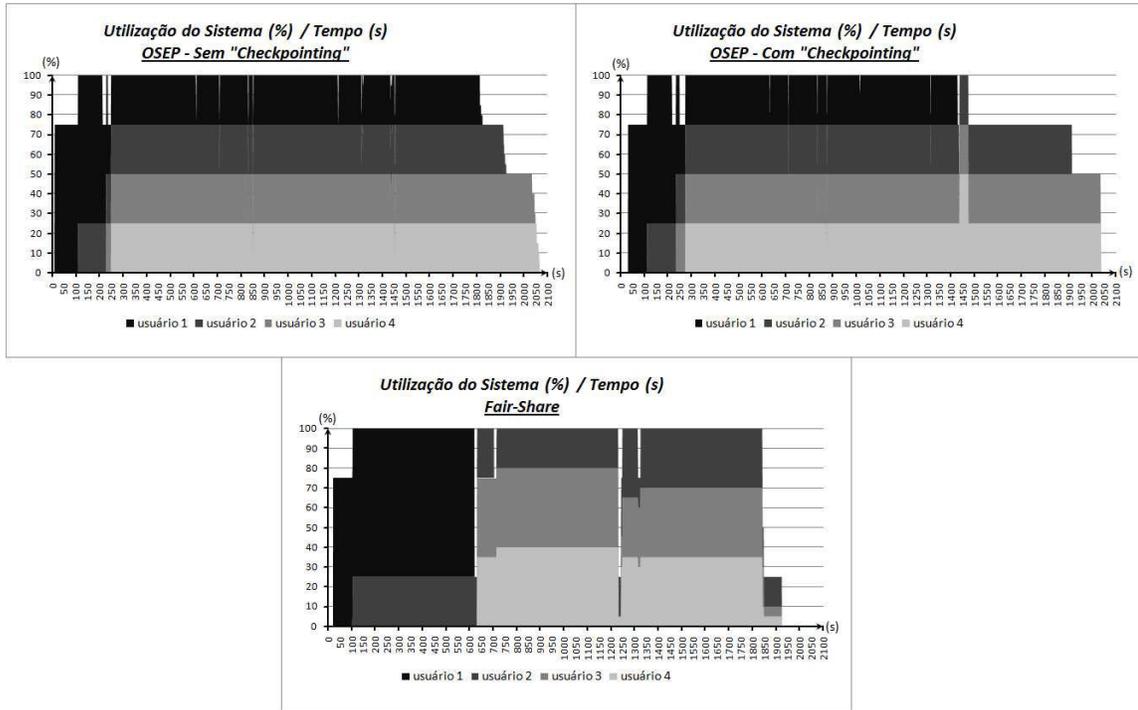


Figura 33: Testes com 4 usuários com porções iguais - Tarefas iguais.

A Tabela 15 mostra os resultados obtidos com os testes para as métricas de desempenho e satisfação dos usuários. Já a Figura 34 apresenta os gráficos de níveis com as informações de satisfação dos usuários obtidas na tabela, para cada abordagem de escalonamento.

Tabela 15: Resultados dos testes com maior quantidade de usuários - Porções iguais e tarefas iguais.

20 Máquinas	OSEP	OSEP Θ	<i>Fair-Share</i>
Violação da Política	1.675	2.300	-
Perda de Capacidade	715	1.005	-
Custo da Política	11.525	100	-
Satisfação do Usuário 1	60,31%	65,32%	96,46%
Satisfação do Usuário 2	62,81%	62,68%	62,14%
Satisfação do Usuário 3	63,77%	63,6%	54,73%
Satisfação do Usuário 4	63,41%	62,36%	55,13%

Pelos resultados, podemos notar que a política OSEP apresenta níveis de satisfação bem próximos para todos usuários, com valores acima de 60%, o que pode ser considerada uma boa satisfação. Nota-se também uma melhora em torno de 5 pontos percentuais na satisfação do usuário 1 entre a política OSEP sem e com *checkpointing*.

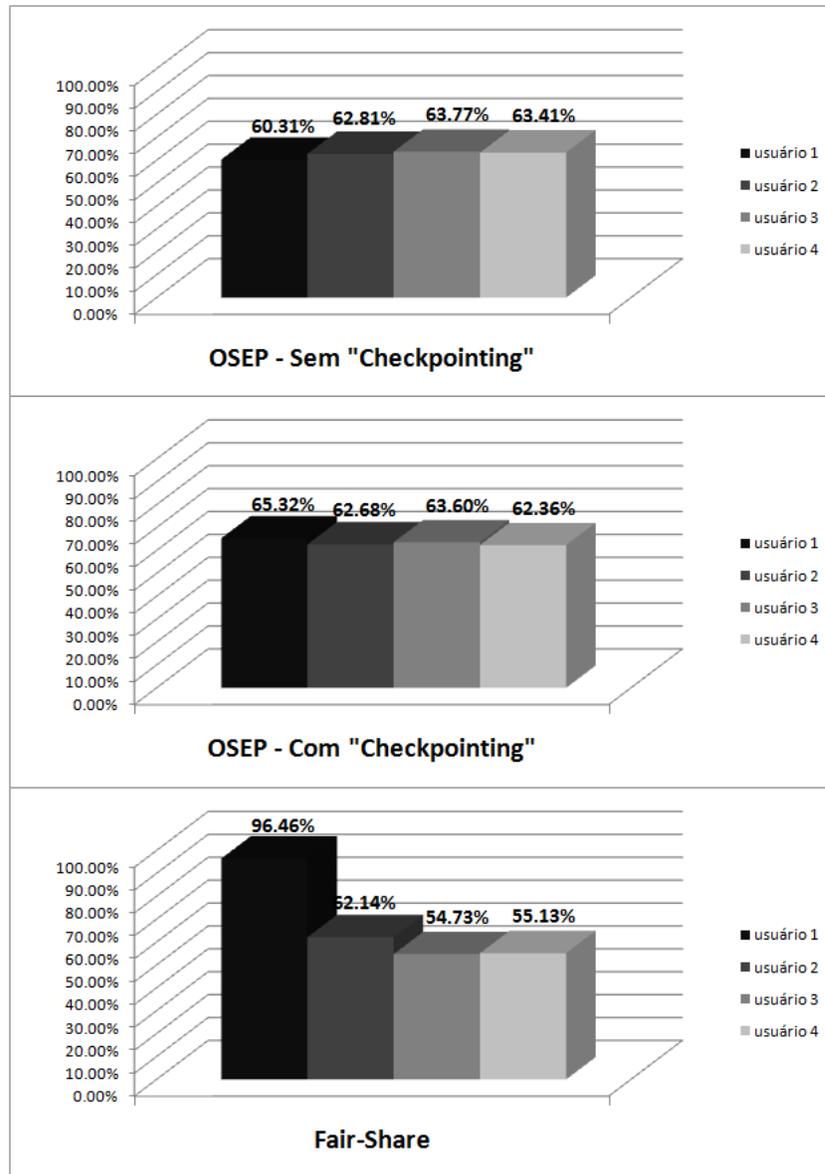


Figura 34: Satisfação dos usuários - Porções iguais e tarefas iguais.

Para a política *Fair-Share*, os resultados mostram uma grande diferença entre a satisfação do usuário 1 e os demais usuários, valores destacados em **negrito** na Tabela 15. O fato de as tarefas do usuário 1 não serem interrompidas para a realocação dos recursos determina uma ótima satisfação para o usuário 1 e, mesmo com a posterior compensação na utilização dos recursos para os outros usuários, eles não atingiram altos níveis de satisfação. O usuário 2 chegou em 62% de satisfação pois recebeu recursos que estavam livres no momento de submissão das tarefas, já os usuários 3 e 4 não ultrapassaram 56% de satisfação.

Ainda neste primeiro estudo de caso, realizou-se testes com tarefas de tamanhos variados, entre 600, 1.800 e 3.600 segundos, distribuídos em proporções iguais dentro da

quantidade de tarefas. Nosso objetivo com esses testes é de simular uma situação cada vez mais próxima da realidade, a fim de obter resultados mais realistas. A Figura 35 apresenta os gráficos de utilização do sistema e Tabela 16 mostra os resultados obtidos.

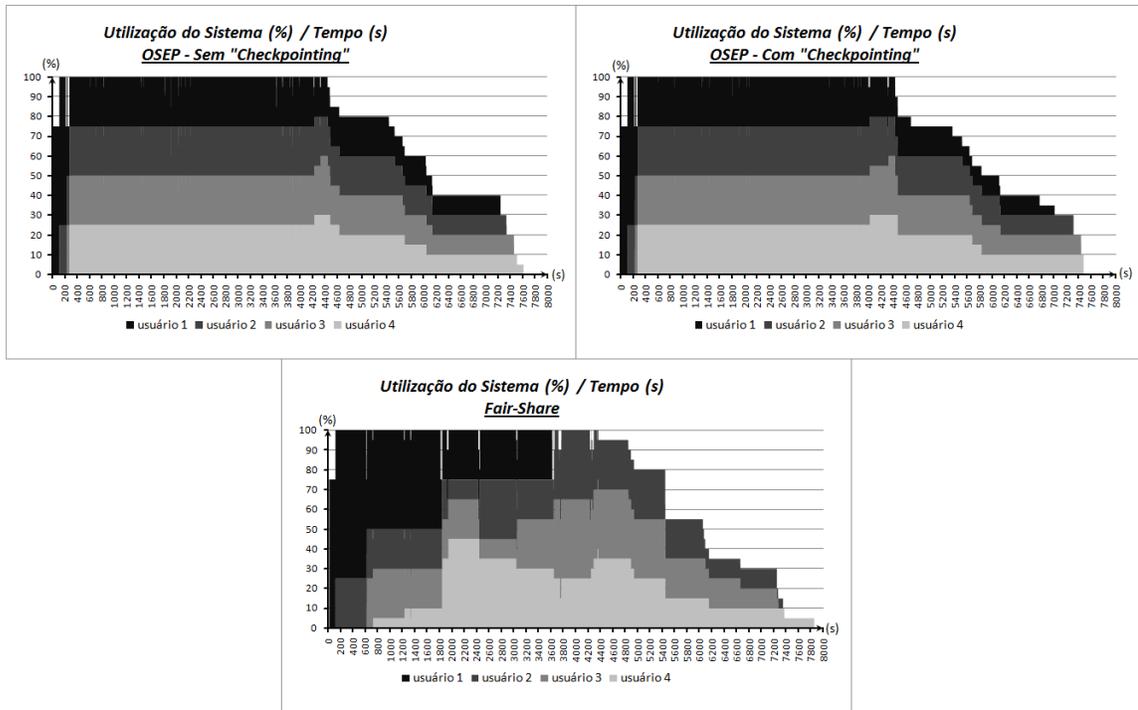


Figura 35: Testes com 4 Usuários com porções iguais - Tarefas diferentes.

Tabela 16: Resultados dos testes com maior quantidade de usuários - Porções iguais e tarefas diferentes.

20 Máquinas	OSEP	OSEP Θ	<i>Fair-Share</i>
Violação da Política	2.300	2.200	-
Perda de Capacidade	1.075	850	-
Custo da Política	14.530	810	-
Satisfação do Usuário 1	60,04%	63,7%	98,22%
Satisfação do Usuário 2	61,21%	61,34%	56,30%
Satisfação do Usuário 3	61,67%	61,9%	50,40%
Satisfação do Usuário 4	60,17%	61,32%	47,16%

Assim como na situação anterior, pode-se notar, pelo gráfico de utilização da política *Fair-Share*, que os usuário 3 e 4 somente recebem recursos quando as tarefas de curta duração do usuário 1 terminam. Como também foram submetidas tarefas de longa duração, o usuário 1 mantém os recursos ocupados por um tempo maior, o que causa uma maior queda na satisfação dos outros usuários, valores destacados também em **negrito** na Tabela 16.

Os gráficos apresentados na Figura 36 deixam mais clara a diferença na satisfação dos usuários. A política *Fair-Share* apresenta uma ótima satisfação para o usuário 1, porém apresenta também um resultado de satisfação abaixo de 50% para o usuário 4, causado pelo simples fato de que as tarefas do usuário 4 foram submetidas após 210 s (3 min e 30 s) da submissão das tarefas do usuário 1.

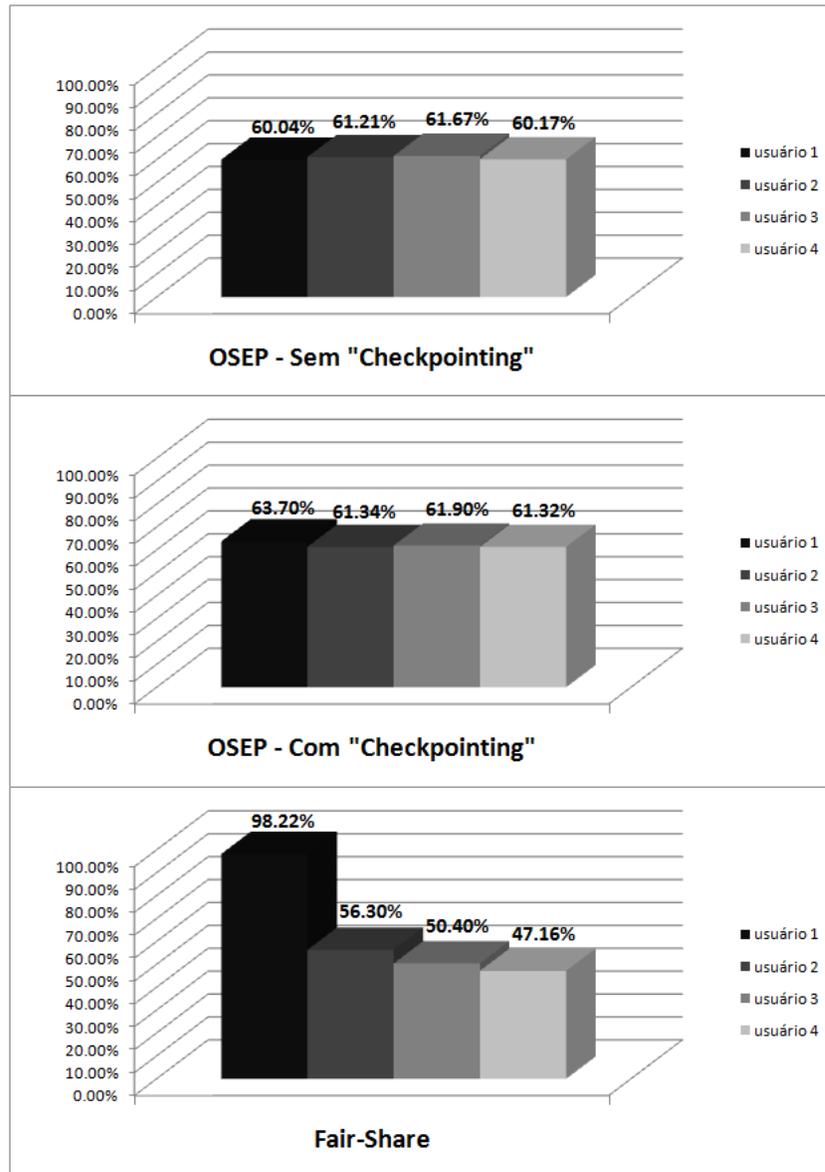


Figura 36: Satisfação dos usuários - Porções iguais e tarefas diferentes.

4.5.2 Estudo de Caso 10: 4 usuários e porções diferentes

Nesse segundo estudo de caso para testes com maior quantidade de usuários, definiu-se o primeiro cenário de análise, em que 4 usuários submetem tarefas ao sistema e disputam por recursos para execução de suas tarefas, com as seguintes características:

- Sistema com 20 máquinas ou recursos;
- Porções diferentes: usuário 1 com 30%, usuário 2 com 40% e os usuários 3 e 4 com 15% cada um;
- 15 tarefas por usuário, tarefas de mesmo tamanho (600 s cada uma);
- $t_1 = 0$ s, $t_2 = 100$ s, $t_3 = 200$ s e $t_4 = 210$ s, onde t_x é o instante de submissão das tarefas do usuário x ;

A Tabela 17 apresenta os resultados obtidos com os testes para as métricas de desempenho e satisfação dos usuários. Pelos valores obtidos, pode-se observar que a política OSEP resulta em porcentagens de satisfação proporcionais à porção dos usuários, ou seja, quanto maior a porção do usuário mais satisfeito ficará pois suas tarefas terminarão mais rápido.

Tabela 17: Resultados dos testes com maior quantidade de usuários - Porções diferentes e tarefas iguais.

20 Máquinas	OSEP	OSEP Θ	<i>Fair-Share</i>
Violação da Política	1.530	1.900	-
Perda de Capacidade	1.420	440	-
Custo da Política	8.660	60	-
Satisfação do Usuário 1	66,2%	71,41%	97,27%
Satisfação do Usuário 2	77,25%	75,92%	61,37%
Satisfação do Usuário 3	55,54%	55,74%	54,86%
Satisfação do Usuário 4	55,53%	55,28%	55,16%

Assim como nos testes anteriores, pode-se notar que a política *Fair-Share* favorece ao usuário que primeiro submete tarefas ao sistema. Entretanto, ela também cria vantagens para os usuários com menor porção pois, como ela trabalha através da compensação pela utilização do sistema, mesmo os usuários com uma pequena porção do sistema têm a oportunidade de utilizar grande parte dos recursos por algum tempo. Devido a isso, pode-se notar que os níveis de satisfação para os usuários 3 e 4 estão bem próximos para ambas as políticas, OSEP e *Fair-Share*.

O resultado que mais chama a atenção é a satisfação do usuário 2, que na política OSEP é favorecido por ter a maior porção e, para a política *Fair-Share*, tem satisfação num patamar mais baixo, valores destacados em **negrito** na Tabela 17. A Figura 37 apresenta os gráficos de utilização do sistema ao longo do tempo para as três políticas em análise.

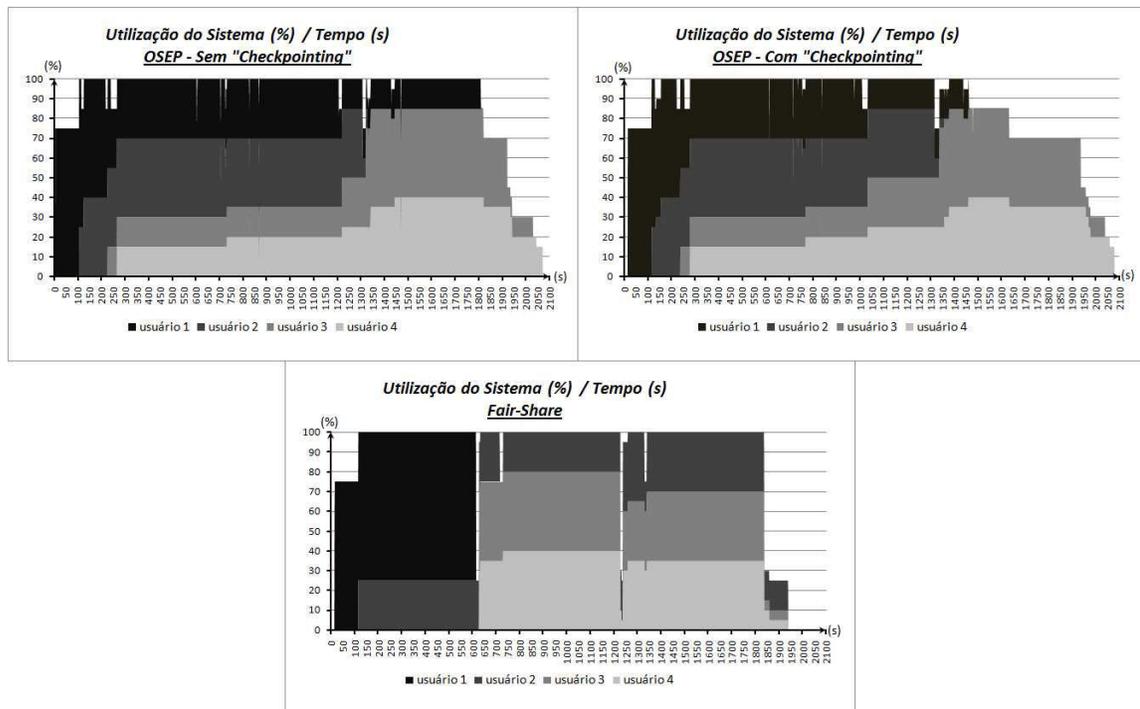


Figura 37: Testes com 4 usuários com porções diferentes - Tarefas iguais.

Ainda neste segundo estudo de caso para testes com maior quantidade de usuários, realizou-se os testes para um segundo cenário em que os usuários submetem tarefas de tamanhos diferentes, com o objetivo de criar um ambiente ainda mais realista. Assim, as características para esse segundo cenário são as mesmas do cenário anterior, com exceção de que cada usuário tem 15 tarefas de tamanho variado, entre 600, 1.800 e 3.600 s, distribuídos em proporções iguais dentro da quantidade de tarefas.

A Figura 38 apresenta os gráficos de utilização e a Tabela 18 mostra os resultados obtidos com os testes. Os resultados mostram que, para uma situação mais próxima da realidade, as políticas OSEP e *Fair-Share* ainda mantêm seus comportamentos e resultados. Para a política OSEP, a satisfação dos usuários ainda permanece proporcional às porções dos usuários. Já a política *Fair-Share* continua favorecendo o usuário que primeiro submeteu tarefas ao sistema, e a compensação na utilização dos recursos não deixa que os usuários com porção pequena de recursos fiquem insatisfeitos. Uma vantagem da política *Fair-Share* que precisa ser destacada é que o conjunto de tarefas termina sua execução mais rapidamente do que na política OSEP, como é visto nos gráficos de utilização. A análise de porque isso ocorre é vista ao final deste capítulo.

Os valores destacados em **negrito** na Tabela 18 mostram a diferença entre a satisfação do usuário 2 para as três políticas em questão. Vê-se que a política OSEP dá prioridade ao fato de que este usuário tem a maior porção no sistema e portanto é o mais satisfeito.

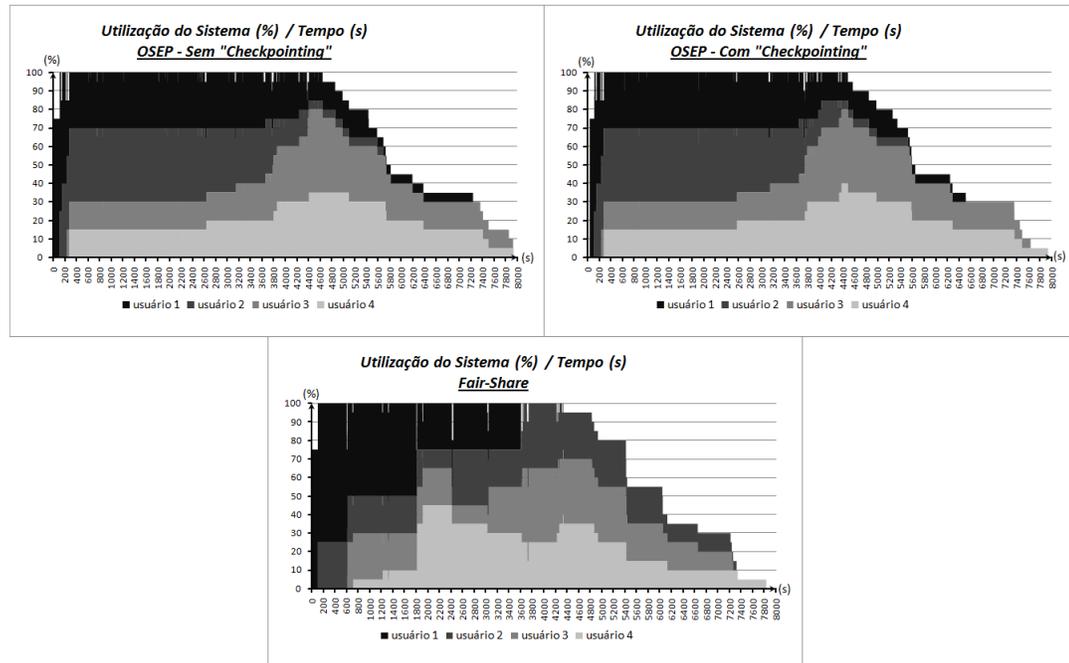


Figura 38: Testes com 4 usuários com porções diferentes - Tarefas diferentes.

Tabela 18: Resultados dos testes com maior quantidade de usuários - Porções diferentes e tarefas diferentes.

20 Máquinas	OSEP	OSEP Θ	<i>Fair-Share</i>
Violação da Política	2.395	2.115	-
Perda de Capacidade	915	800	-
Custo da Política	18.380	790	-
Satisfação do Usuário 1	67,18%	70,01%	99,40%
Satisfação do Usuário 2	77,02%	78,20%	56,7%
Satisfação do Usuário 3	47,98%	50,61%	50,59%
Satisfação do Usuário 4	50,02%	50,60%	50,58%

Já a política *Fair-Share* produz resultados que priorizam muito mais a compensação na utilização do sistema do que a porção do usuário. A Figura 39 apresenta os gráficos de níveis para uma melhor visualização e comparação entre os valores de satisfação dos usuários.

4.6 Análise Global dos Resultados

A defesa de uma política de escalonamento deve partir do objetivo de sua aplicação. Quando justiça é um parâmetro desejado pelos usuários do sistema, várias abordagens podem ser definidas. Essa definição deve ser feita de acordo com as necessidades e os interesses desses usuários.

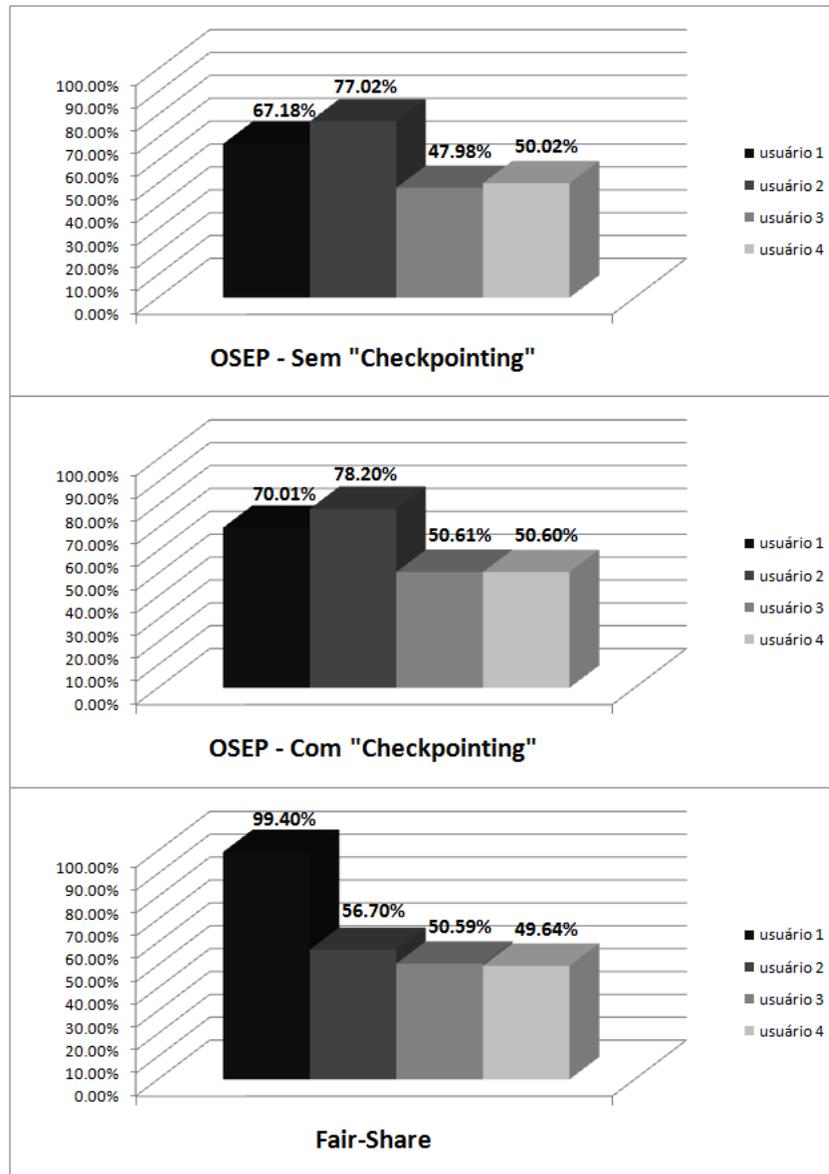


Figura 39: Satisfação dos usuários - Porções diferentes e tarefas diferentes.

A política proposta, OSEP, tem por objetivo garantir ao usuário proprietário de recursos do sistema, o uso de sua porção de recursos quando necessário. A partir desse objetivo, desenvolveu-se o algoritmo *Owner-Share* que implementa a política OSEP de forma eficiente e com bons resultados.

Os resultados obtidos com os testes mostram que o escalonamento das tarefas a partir da política OSEP, atinge a garantia de uso pelo proprietário dos recursos. Os pontos positivos a serem destacados são:

- A satisfação dos usuários é mais homogênea (menor desvio padrão) do que a satisfação proporcionada pelo algoritmo *Fair-Share* como mostrado pela Tabela 19;

- O custo de sua aplicação é relativamente pequeno, se considerar o valor relativo do aumento no tempo de entrega final das tarefas;
- A satisfação dos usuários é proporcional ao montante de recursos que ele disponibiliza, independentemente dos cenários de execução avaliados.

A Tabela 19 mostra os resultados de desvio padrão, calculados entre as satisfações dos usuários nos estudos de caso para situações mais realistas, que são os estudos número 9 e 10 apresentados anteriormente. Pode-se notar nos resultados apresentados, que a política OSEP oferece um desvio padrão inferior ao da política *Fair-Share* para ambos casos, com ou sem *checkpointing*. A diferença nos valores de desvio padrão para a política OSEP surge em função da diferença nas porções de cada usuário no caso 10. Como usuários com porções diferentes terão valores de satisfação também diferentes, é esperado que o desvio padrão nesses valores também aumente. O fato disso não ocorrer para o *Fair-Share* mostra que essa política não considera a propriedade de recursos como um fator relevante para a alocação dos mesmos.

Tabela 19: Desvio Padrão das satisfações dos usuários para as políticas OSEP e *Fair-Share*

Estudo de caso	OSEP	OSEP Θ	<i>Fair-Share</i>
9 com tarefas iguais	1,4595	1,3284	19,8570
9 com tarefas diferentes	0,7950	1,1226	23,7697
10 com tarefas iguais	10,3789	10,6439	20,2930
10 com tarefas diferentes	13,9535	13,9736	23,5653

Considerando-se todos os pontos aqui levantados, conclui-se que a aplicação da política OSEP é bastante favorável. O primeiro ponto destacado permite afirmar que os usuários não sentirão momentos de frustração por terem suas tarefas momentaneamente postergadas, como ocorre no *Fair-Share*. Já o terceiro ponto indicado permite concluir que a sua aplicação pode incentivar a entrada de novos usuários trazendo mais recursos ao sistema, uma vez que saberão que sempre terão uma satisfação proporcional ao montante de recursos que coloquem no sistema.

O segundo ponto destacado é considerado positivo pois a introdução de uma política preemptiva de escalonamento indicaria possíveis atrasos na execução das tarefas interrompidas. Como no caso da política OSEP esse atraso foi relativamente pequeno para todas as situações, pode-se concluir que o custo de sua aplicação não inviabiliza a execução de tarefas, grandes ou pequenas, dentro de tempos razoáveis de execução. Deve ser observado que, com a utilização de um mecanismo de *checkpointing*, o atraso no tempo de entrega é ainda menor.

A principal desvantagem observada em relação à política *Fair-Share* diz respeito ao valor médio de satisfação dos usuários. Na maior parte dos cenários essa média foi ligeiramente maior no escalonamento feito pelo *Fair-Share* (diferença inferior a 2 pontos percentuais). Apesar disso considera-se que os resultados obtidos com a OSEP podem ser considerados melhores por apresentarem um melhor desvio padrão entre as satisfações de cada usuário. Além disso, deve ser observado que, nos casos em que o cenário era mais favorável à OSEP o valor médio de satisfação obtido com *Fair-Share* foi significativamente inferior ao da OSEP.

4.7 Considerações Finais

Neste capítulo descreveu-se os testes de ajuste para a política OSEP, além dos testes para avaliação e comparação das políticas OSEP e *Fair-Share* em diversos cenários e estudos de casos. Organizou-se os estudos de caso pela variação da quantidade de recursos no sistema, variação da quantidade de tarefas por usuário, variação do tamanho das tarefas e variação da quantidade de usuários em um sistema com porções iguais e porções diferentes. Dessa forma, tentou-se abranger todas as situações para teste e comparação das políticas. Os resultados fornecem informações importantes sobre as características de cada política, além de tornar mais claras suas diferenças e seus pontos fracos e fortes. No capítulo seguinte apresenta-se as conclusões obtidas e as possibilidades de trabalhos futuros que seguem a partir dessa tese.

5 *Conclusões e Trabalhos Futuros*

Esse capítulo apresenta as conclusões do trabalho realizado nessa tese, além dos trabalhos futuros. Na seção a seguir apresenta-se as afirmações e conclusões obtidas após a realização dos testes e na seção seguinte propõe-se algumas direções a serem seguidas a partir deste trabalho.

5.1 *Conclusões*

Existem várias formas de alcançar justiça na distribuição dos recursos entre os usuários de um sistema distribuído composto de recursos amplamente espalhados. Cada política de escalonamento defende uma justificativa e promove a sua justiça, como a política *Fair-Share* que defende que a justiça está na igualdade da utilização dos recursos entre os usuários do sistema.

Essa tese defende que a justiça pode também ser alcançada levando em consideração o conceito de propriedade distribuída dos recursos do sistema. Para tanto considera-se que cada usuário é proprietário de uma parcela dos recursos que compõem o sistema e tem todo direito de usar esses recursos quando necessário. Com os testes realizados, mostrou-se que a busca por essa justiça através da política OSEP gera aos usuários garantias e vantagens para utilização de seus recursos e, de certa maneira, até incentiva a colaboração e a criação de sistemas através da união de recursos entre entidades.

Diante do parâmetro de justiça estabelecido, a política OSEP mostrou que a satisfação do usuário deve estar ligada diretamente com a sua porção de propriedade do sistema. Quanto maior a porção de um usuário, maior sua satisfação na execução das tarefas. O sistema de compensação na utilização do sistema oferecido pela política *Fair-Share* permite, na maioria das situações, a obtenção de um valor médio de satisfação ligeiramente superior ao obtido com a OSEP. Entretanto, os resultados mostram um elevado desvio padrão em relação à média, o que é uma deficiência importante. Deve ser observado ainda que o *Fair-Share* depende fortemente dos instantes de ocorrência das tarefas, como

evidenciado em alguns casos avaliados.

A política OSEP pode ser implementada em qualquer sistema criado através da colaboração entre entidades onde haja o compartilhamento de recursos e que tenha um controle centralizado. Sua implementação em ambientes diferentes precisa ser ajustada a partir da determinação de valores ótimos para a forma de atuação (número de tarefas interrompidas, frequência de ativação do algoritmo, etc.), seguindo-se procedimentos como os apresentados na Seção 4.2.

Enfim, pode-se concluir que o objetivo de criar, defender e oferecer à comunidade uma idéia que solucionasse ou promovesse a justiça entre os usuários, de sistemas criados através da união de recursos a partir do conceito de propriedade distribuída foi alcançado. Mostrou-se que a política OSEP oferece a garantia da justiça entre os usuários de um sistema segundo esse conceito. As comparações com a política *Fair-Share* permitem determinar as características e os pontos de vantagem de cada uma. Assim, oferece-se à comunidade mais uma possibilidade e uma visão diferente para administração, gerenciamento e escalonamento de recursos. Cabe ao administrador do sistema escolher a política que melhor se encaixa às suas necessidades.

5.2 Trabalhos Futuros

O trabalho desenvolvido nessa tese trata da elaboração de uma política de escalonamento para sistemas distribuídos baseado no conceito de propriedade distribuída. A partir desse trabalho identifica-se alguns pontos que podem ser mais explorados, dando origem aos trabalhos futuros. Esses pontos são:

- **Tratamento a usuários não-proprietários pela política OSEP:** em sua atual formulação a política OSEP impede que usuários não-proprietários tenham acesso ao sistema quando o mesmo estiver sobrecarregado. Assim, um desdobramento importante envolve a inserção de estratégias para garantir, dentro de certos níveis, o uso de parte do sistema por não-proprietários. Obviamente essa garantia não pode vir em detrimento dos usuários proprietários.
- **Estudo e análise de uma solução envolvendo as políticas OSEP e *Fair-Share*:** esse trabalho futuro trata do estudo, elaboração e análise de uma solução de escalonamento que envolva todos os conceitos de ambas as políticas com o intuito de oferecer uma solução mais abrangente sobre o conceito de justiça pela disputa

e compartilhamento de recursos em sistemas criados através da união de recursos de diferentes entidades. A idéia aqui seria garantir inicialmente o direito de propriedade, passando proporcionalmente ao conceito de compensação fornecido pelo Fair-Share.

- **Investigação da descentralização da política OSEP:** como indicado em vários pontos da tese, a política OSEP é implementada considerando a existência de um escalonador global. Na definição clássica de grades espera-se que o escalonamento seja feito de forma descentralizada. Assim, encontrar um mecanismo para forçar o atendimento do conceito de propriedade através de escalonadores distribuídos é também um desdobramento importante desse trabalho.
- **Avaliação da política OSEP em um sistema com recursos heterogêneos:** nesse trabalho o conceito de propriedade é representado pelo número de máquinas sendo que a infraestrutura considerada contém máquinas homogêneas. Para sistemas com recursos heterogêneos, o conceito de propriedade pode ser representado pela capacidade de processamento dos recursos. No entanto, antes de qualquer conclusão, é necessária uma avaliação completa da política OSEP nesse cenário.

Além das conclusões e trabalhos futuros apresentados, esse trabalho proporcionou a escrita de dois artigos científicos:

- “Avaliação de algoritmos de escalonamento em Grids para diferentes configurações de ambiente”: artigo publicado no Wperformance de 2007 no Congresso da Sociedade Brasileira de Computação;
- “The Owner-Share Scheduler for a Distributed System”: artigo submetido ao “Fifth International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems” (SRMPDS) de 2009 na conferência internacional ICPP’09 (“The 2009 International Conference on Parallel Processing”).

Referências

ABBAS, A. **Grid computing: a practical guide to technology and applications**. Ahmar Abbas: Charles River Media, 2003.

ADAMS, J.; IBM. **IBM grid computing strategy**. 2.ed. ApGrid Workshop. 2002.

AMAR, L. et al. An on-line algorithm for fair-share node allocations in a cluster. In: PROCEEDINGS OF THE SEVENTH IEEE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, CCGRID'07, 7, 2007, Washington. **Proceedings of the...** Washington: IEEE, 2007. p.83-91.

ANDRADE, N. **Reputação autônoma como incentivo à colaboração no compartilhamento de recursos computacionais**. 2004. Dissertação (Mestrado) - Centro de Ciência e Tecnologia, Universidade Federal de Campina Grande, Campina Grande, 2004.

ANDRADE, N. et al. OurGrid: an approach to easily assemble grids with equitable resource sharing. In: WORKSHOP ON JOB SCHEDULING STRATEGIES FOR PARALLEL PROCESSING, 9, 2003, Seattle. **Workshop...** Seattle: IEEE, 2003. p. 61-86.

ANDRADE, N.; BRASILEIRO, F.; MOWBRAY, M. Discouraging free riding in a peer-to-peer CPU-sharing grid. In: PROCEEDINGS OF THE IEEE INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING, HPDC'04, 13, 2004. **Proceedings of the...** S.L.: IEEE Computer Society, p. 129-137. Disponível em: <<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1323511&isnumber=29284>>. Acesso em: 15 jun. 2009.

ANDRADE, N. et al. **When can an autonomous reputation scheme discourage free-riding in a peer-to-peer system**. 2004. p. 440-448. Disponível em: <<http://www2.lsd.ufcg.edu.br/~nazareno/documentos/when-can-autonomous.pdf>>. Acesso em: 10 jun. 2009.

BUYYA, R.; ABRAMSON, D.; GIDDY, J. **Nimrod/G: an architecture for a resource management and scheduling system in a global computational grid**. 2000. p. 283-289. Disponível em:

<<http://www.gridbus.org/papers/nimrodg.pdf>>. Acesso em: 15 maio 2009.

CASANOVA, H. et al. **Heuristics for scheduling parameter sweep applications in grid environments**. 2000. p.349. Disponível em: <http://www.cs.ucsb.edu/~dmitrii/research/pubs/casanova_et_al-heur_sched_par_sweeps_grid-hcw00.pdf>. Acesso em: 10 jun. 2009.

CHTEPEN, M.; DHOEDT, B.; VANROLLEGHEME, P. **Dynamic scheduling in grid systems**. 2005. p. 95-96. Disponível em: <<http://www.ibcn.intec.ugent.be/papers/2558.pdf>>. Acesso em: 12 abr. 2009.

CIRNE, W. et al. **Running bag-of-tasks applications on computational grids: the MyGrid approach**. 2003. International Conference on Parallel Processing, IEEE Computer Society, p. 407-416.

CIRNE, W.; SANTOS NETO, E. Grids computacionais: da computação de alto desempenho a serviços sob demanda. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, SBRC, 2005, Fortaleza. **Anais...** Fortaleza: S.n., 2005. (Mini curso).

CONDOR PROJECT. **Condor website**. 2009. Disponível em: <http://www.cs.wisc.edu/condor>. Acesso em: 01 maio 2009.

DEPARTAMENTO DE INTEGRAÇÃO DE SISTEMAS DE INFORMAÇÃO. **Guia de inovação tecnológica em cluster e grid**. 2009. Disponível em: <http://guialivre.governoeletronico.gov.br/guiaonline/guiacluster> Acesso em: 01 maio 2009.

DEWITT, T. et al. **ReMoS: a resource monitoring system for network-aware applications**. Carnegie Mellon School of Computer Science, 1997. CMU-CS-97-194. (Technical Report).

EGEE PROJECT. **Egee project website**. 2009. Disponível em: <http://www.lip.pt/computing/projects/EGEE/>. Acesso em: 01 maio 2009.

FALAVINHA JUNIOR, J. et al. Avaliação de algoritmos de escalonamento em grids para diferentes configurações de ambiente. In: CONGRESSO ANUAL DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, Wperformance, 27, 2007, Rio de Janeiro. **Anais...** Rio de Janeiro: SBC, 2007.p. 505-524.

FOSTER, I. **What is the grid?** A three point checklist. 2002. GRIDtoday, v. 1, n.6. Disponível em:

<<http://www.mcs.anl.gov/~itf/Articles/WhatIsTheGrid.pdf>>. Acesso em: 02 abr. 2009.

FOSTER, I.; KESSELMAN, C. **Globus**: a metacomputing infrastructure toolkit. **International Journal of Supercomputer Applications**, Cambridge, v. 11, p. 115-128, 1997.

FOSTER, I.; KESSELMAN, C. **Globus**: a toolkit-based grid architecture. In:____. The grid: blueprints for a New Computing Infrastructure. Foster: Morgan kaufmann,1999. p. 259-278.

FOSTER, I.; KESSELMAN, C. **The grid 2**: blueprint for a new computing infrastructure. Foster: Morgan Kaufmann, 2003.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. The anatomy of the grid: enabling scalable virtual organizations. **International Journal of High Performance Computing Applications**, Thousand, v. 15, n. 3, p.200-222, 2001.

FREY, J. et al. Condor-G: a computation management agent for multi-institutional grids. **Cluster Computing**, v. 5, p. 237-246. 2002. Disponível em: <<http://www.cs.wisc.edu/condor/doc/condorg-hpdc10.pdf>>. Acesso em: 15 jun. 2009. Cluster Computing, v. 5, p. 237-246. 2002.

FUJIMOTO, N.; HAGIHARA, K. A comparison among grid scheduling algorithms for independent coarse-grained tasks. In: PROCEEDINGS OF THE SYMPOSIUM ON APPLICATIONS AND THE INTERNET-WORKSHOPS, 2004, Washington. **Proceedings of the...** Washington: IEEE Computer Society, 2004. p. 674-680.

GLITE PROJECT. **Glite project website**. Disponível em: <http://glite.web.cern.ch/glite/>. Acesso em: 01 maio 2009.

GLOBAL GRID FORUM. **Global grid forum website**. Disponível em: <http://www.gridforum.org>. Acesso em: 01 maio 2009.

GLOBUS PROJECT. **Globus project website**. Disponível em: <http://www.globus.org>. Acesso em: 01 maio 2009.

GRIMSHAW, A. et al. **Legion**: an operating system for wide-area computing. Virginia: University of Virginia in Charlottesville, 1999 (Technical Report).

GRIMSHAW, A. et al. From legion to avaki: the persistence of vision. **Grid Computing**: making the global infrastructure a reality. New York: John Wiley & Sons, 2003. p. 265-298.

GRIMSHAW, A.; WULF, W. The Legion vision of a worldwide virtual computer. **Journal of ACM**, New York, v. 40, n. 1, p. 39-45, 1997.

IBARRA, O.; KIM, C. Heuristic algorithms for scheduling independent tasks on nonidentical processors. **Journal of ACM**, New York, v. 24, n. 2, p. 280-289, 1977.

JONGH, J. **Share scheduling in distributed systems**. 2002. PhD (Thesis) - Proefschrift, Technische Universiteit Delft.

KAY, J.; LAUDER, P. A fair share scheduler. **Journal of ACM**, New York, v. 31, n. 1, p. 44-55, 1988.

KLEBAN, S.; CLEARWATER, S. Fair share on high performance computing systems: what does fair really mean? 2003. In: PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, CCGRID'03, 3, 2003, Washington. **Proceedings of the...** Washington: IEEE Computer Society, 2003. p. 146.

LHC COMPUTING GRID. **LHC computing grid website**. 2009. Disponível em: <http://lcg.web.cern.ch/LCG/>. Acesso em: 01 maio 2009.

LITZKOW, M.; LIVNY, M.; MUTKA, M. **Condor** - a hunter of idle workstations. 1988. Disponível em: <http://www.cs.wisc.edu/condor/doc/icdcs1988.pdf>. Acesso em: 25 jun. 2009.

LITZKOW, M. et al. **Checkpoint and migration of UNIX processes in the Condor distributed processing system**. Wisconsin : Computer Sciences Department at University of Wisconsin in Madison, 1997. (Technical Report).

LIVNY, M.; RAMAN, R. High-throughput resource management. In: _____. The grid: blueprint for a new computing infrastructure. Foster: Morgan Kaufmann, 1998. p. 311-337.

LSF PLATFORM. **LSF platform website**. 2009. Disponível em: <http://www.platform.com/Products/platform-lsf>. Acesso em: 01 maio 2009.

MASSIE, M.; CHUN, B.; CULLER, D. The ganglia distributed monitoring system: design, implementation, and experience. **Journal of Parallel Computing**, New York, v. 30, p. 817-840, 2004.

MENASCÉ, D. et al. Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures. **Journal of Parallel and Distributed Computing**, Orlando, v. 28, n. 1, p. 1-18, 1995.

MORENO, R. Job scheduling and resource management techniques in economic grid environments. **Across Grids**, v.2970 p. 25-32, 2004.

MYGRID PROJECT. **Mygrid project website**. 2009. Disponível em: <http://www.ourgrid.org/mygrid>. Acesso em: 01 maio 2009.

OGSA-DAI PROJECT. **OGSA-DAI project website**. 2009. Disponível em: <http://www.ogsadai.org.uk/downloads/>. Acesso em: 01 de maio de 2009.

OURGRID PROJECT. **Ourgrid project website**. 2009. Disponível em: <http://www.ourgrid.org>. Acesso em: 01 maio 2009.

PBS PROJECT. **PBS project website**. 2009. Disponível em: <http://www.openpbs.org>. Acesso em: 01 maio 2009.

RAMAN, R.; LIVNY, M.; SOLOMON, M. Matchmaking: distributed resource management for high throughput computing. In: PROCEEDINGS OF THE SEVENTH IEEE INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING, 7,1998, Chicago. **Proceedings of the...** Chicago: IEEE, 1998. p. 28-31.

RANGANATHAN, K.; FOSTER, I. Decoupling computation and data scheduling in distributed data-intensive applications. High Performance Distributed Computing. In: PROCEEDINGS OF THE IEEE INTERNATIONAL SYMPOSIUM, HPDC,11, 2002, Flórida. **Proceedings of the...** Flórida: IEEE, 2002. p. 352-358.

SANTOS NETO, E. **Escalonamento de aplicações que processam grande quantidade de dados em grids computacionais**. 2004. Dissertação (Mestrado) - Centro de Ciência e Tecnologia, Universidade Federal de Campina Grande,, Campina Grande, 2004.

SCHOPF, J. **A general architecture for scheduling on the grid**. 2002. Chicago: - Argonne National Laboratory - Department of Computer Science, Northwestern University in Chicago, 2002. (Technical Report).

SILBERSTEIN, M.; GEIGER, D.; SCHUSTER, A. Scheduling mixed workloads in multi-grids: the grid execution hierarchy. In PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE

DISTRIBUTED COMPUTING, HPDC'06, 15, 2006, Paris. **Proceedings of the...** Pais: IEEE, 2006. p. 291-302.

SILVA, D. **Usando replicação para escalonar aplicações bag-of-tasks em grids computacionais.** 2003. Dissertação (Mestrado) - Centro de Ciência e Tecnologia, Universidade Federal de Campina Grande, Campina Grande, 2003.

SILVA, D.; CIRNE, W.; BRASILEIRO, F. **Trading cycles for information:** using replication to schedule bag-of-tasks applications on computational grids. 2003. Disponível em: <<http://walfredo.lsd.ufcg.edu.br/papers/WQReplication.pdf>>. Acesso em: 25 jun. 2009.

SUN MICROSYSTEMS. **Sun cluster grid architecture:** a technical white paper describing the foundation of sun grid computing. 2002. White Paper.

TANNENBAUM, T. et al. **Condor:** a distributed job scheduler. Cambridge: Beowulf cluster computing with Linux, MIT Press, 2002. p. 307-350.

THAIN, D.; TANNENBAUM, T.; LIVNY, M. Condor and the grid. **Grid Computing: Making the Global Infrastructure a Reality.** New York: John Wiley & Sons, 2002.

WOLSKI, R.; SPRING, N.; HAYES, J. The network weather service: a distributed resource performance forecasting service for metacomputing. 1999. **Future Generation Computer Systems,** Amsterdam, v. 15, n. 5/6, p. 757-768.