Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Um simulador discreto escalável e extensível do OurGrid

Abmar Grangeiro de Barros

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Metodologia e Técnicas da Computação

Francisco Vilar Brasileiro
(Orientador)

Campina Grande, Paraíba, Brasil

B277s Barros, Abmar Grangeiro de
Um simulador discreto escalavel e extensivel do OurGrid / Abmar Grangeiro de Barros. - Campina Grande, 2012.
89 f. : il.

Dissertacao (Mestrado em Ciencia da Computacao) - Universidade Federal de Campina Grande, Centro de Engenharia Eletrica e Informatica.

1. Simulador 2. OurGrid 3. Dissertacao I. Brasileiro, Francisco Vilar, Dr. II. Universidade Federal de Campina Grande - Campina Grande (PB) III. Título

CDU 004.75(043)

"UM SIMULADOR DISCRETO ESCALÁVEL E EXTENSÍVEL DO OURGRID"

ABMAR GRANGEIRO DE BARROS

DISSERTAÇÃO APROVADA EM 24/02/2012

FRANCISCO VILAR BRASILEIRO, Ph.D
Orientador(a)

DALTON DARIO SEREY GUERRERO, D.Sc
Examinador(a)

AYLA DÉBORA DANTAS DE SOUZA REBOUÇAS, D.Sc
Examinador(a)

CAMPINA GRANDE - PB

Resumo

Por falta de padrões de simulação no contexto de computação em grade, a grande maioria dos pesquisadores acaba reescrevendo seus simuladores, e de forma *ad-hoc*. Isso acarreta num retrabalho de *design*, implementação e validação. Cada publicação que apresenta um novo simulador carrega os riscos de uma validação com erros, possivelmente evitados com o reuso de um simulador amadurecido, e entrega um modelo simplista, seja do ponto de vista da aplicação ou da plataforma, que muito provavelmente não será reaproveitado (ou reproduzido) em trabalhos futuros. Esse fato pode ser observado nas publicações relacionadas ao OurGrid, *middleware* de grades computacionais entre pares desenvolvido no Laboratório de Sistemas Distribuídos desta Universidade. Vários autores implementaram diferentes simuladores para avaliar diferentes aspectos do *middleware* em questão. O objetivo deste trabalho é apresentar as técnicas de desenvolvimento de um simulador discreto do OurGrid a partir de um modelo genérico o suficiente para ser facilmente reusado em diferentes níveis de abstração e de funcionalidade. Isso significa abordar no modelo o maior número de aspectos inerentes ao sistema real e tornar a aplicação desse modelo no simulador configurável. Este modelo de simulação é validado contra requisitos de completude, coerência, escalabilidade, velocidade de execução, extensibilidade e testabilidade. Por fim, casos de uso de extensão do modelo proposto são apresentados.

Abstract

Due to the lack of simulation standards in the context of grid computing, the vast majority of the researchers end up writing their simulation tools from the scratch, and in an ad-hoc basis. That leads to an extra effort of design, implementation and validation. Each paper that presents a new simulator not only carries the risk of validation flaws, possibly avoided with the reuse of a mature simulation tool, but also delivers a simplistic model for the application or platform level, which is unlikely to be reused or reproduced in future works. This fact can be noticed in most of the OurGrid publications, a middleware that enables the creation of peer-to-peer computational grids, which is developed in the Distributed Systems Lab of this very University. Many authors write different simulators in order to evaluate different aspects of this middleware. The objective of this work is to present the implementation techniques of a discrete simulator to the OurGrid, starting from a simulation model that should be generic enough to be easily reused in different levels of abstraction and functionality. That means that the model must comprise most of the aspects of the real system and must be easily configurable to the different levels of research needs. We validated the proposed simulation model against completeness, accuracy, scalability, speed, extensibility, and testability requirements. Finally, we present some use cases that depict the extension of this model for specific research needs.

Agradecimentos

Agradeço primeiramente à CAPES por financiar meus estudos de pós-graduação. Sinto-me satisfeito e agradecido em ter trabalhado com todos que fazem parte do Laboratório de Sistemas Distribuídos, em especial meu orientador Fubica, com quem tanto aprendi durante meu processo de formação superior; Thiago Manel, que está sempre disposto a ouvir e lapidar uma ideia; Rodrigo Vilar, que me mostrou que a importância de assumir responsabilidades e correr riscos; e aos meus companheiros do Intevol, com quem dividi horas de programação e mesas de bar.

Um salve à minha familia e à minha namorada, que me aguentaram nos momentos de péssimo humor e que me deram suporte incondicional.

Por fim, um abraço a todos os meus amigos, aqueles de infância, companheiros de bandas, colegas de escola, que nunca me deixaram faltar momentos de descontração.

Criar coisas pro mundo é fascinante. Pense. Faça.

Conteúdo

Lista de Figuras

Lista de Tabelas

Trechos de código

Capítulo 1

Introdução

Um problema recorrente na pesquisa em computação distribuída, particularmente em grades computacionais, é garantir a qualidade dos sistemas com respeito a certas métricas (tempo de execução de tarefas obtido por uma heurística de escalonamento particular, impacto de políticas de tolerância a falhas no desperdício de recursos, etc). Em alguns casos raros é possível testar essas garantias de forma analítica, ou seja, puramente por análise teórica. Esse tipo de prova, na grande maioria das vezes, requer uma série de suposições não realistas, que restringem a validade do modelo. Assim, a maioria dos resultados de pesquisa são obtidos de forma empírica, por meio de experimentos [26].

Dessa forma, uma abordagem óbvia é conduzir experimentos em aplicações reais executando em ambientes de produção ou em grandes *testbeds*. Esse tipo de abordagem é mais prática e comprova que a solução proposta pode ser implementada de fato.

Porém, esse tipo de experimentação pode levar muito tempo, uma vez que as aplicações tomarão o tempo real de execução, e pode demandar muito esforço, pois é necessário implementar todas as alternativas, de *design* ou de algoritmo, planejadas para o experimento na aplicação real. Além disso, os riscos oriundos da falta de controle (no ambiente de produção) ou da falta de representatividade (em *testbeds*) podem afetar diretamente na significância estatística dos resultados.

Também vale salientar que para um trabalho científico ter, de fato, uma contribuição, deve ser possível reproduzir de forma fácil seus resultados [48, 49]. Dessa forma, além dos problemas citados acima, executar tais experimentos no mundo real dificulta a reprodutibilidade dos resultados [37].

Com todas essas dificuldades para conduzir experimentos no mundo real, a maioria dos resultados publicados é obtida por meio de simulações.

Simulação pode resolver vários dos problemas citados acima: não há necessidade da construção de um sistema real para executar uma simulação; os experimentos são conduzidos de forma controlada; em princípio, não há limitações para as condições de cenários executáveis; e é possível reproduzir os resultados de forma simples.

Um ponto chave no desenvolvimento de simuladores é a validação. É necessário definir um modelo computacional e validá-lo com o mundo real (validação externa). Porém validar um simulador contra um sistema do mundo real não é trivial: é necessário justificar que o modelo é plausível; que o simulador implementa o modelo (verificação); e que o simulador tem resultados razoáveis (comparação de casos especiais, validação contra um simulador existente, resultados seguindo tendências conhecidas, etc.) [16].

Por falta de padrões de simulação no contexto de computação em grade, a grande maioria dos pesquisadores acaba reescrevendo seus simuladores, e de forma ad-hoc [37]. Isso acarreta num retrabalho de *design*, implementação e validação. Cada publicação que apresenta um novo simulador carrega os riscos de uma validação com erros, possivelmente evitados com o reuso de um simulador amadurecido, e entrega um modelo simplista, seja do ponto de vista da aplicação ou da plataforma, que muito provavelmente não será reaproveitado (ou reproduzido) em trabalhos futuros.

Esse fato pode ser observado nas publicações relacionadas ao OurGrid [30], *middleware* de grade computacionais entre pares desenvolvido no Laboratório de Sistemas Distribuídos desta Universidade. Os trabalhos de Cirne et al. [31], Santos et al. [56], Brasileiro et al. [34], Gaudêncio e Brasileiro [39] e De Rose et al. [55], ilustram a implementação de diferentes simuladores para avaliar diferentes aspectos do *middleware* em questão.

1.1 Objetivos

O objetivo desse trabalho é apresentar e desenvolver um simulador discreto do OurGrid a partir de um modelo suficientemente genérico para ser facilmente reusado em diferentes níveis de abstração e de funcionalidade. Isso significa abordar no modelo o maior número de aspectos inerentes ao sistema real e tornar a aplicação desse modelo no simulador configu-

rável.

A condição de sucesso é obter, através do emprego apropriado de técnicas de engenharia de *software*, um simulador preciso, rápido, escalável e facilmente extensível. Mais adiante neste trabalho serão descritas as métricas que serão usadas para avaliar o atendimento desses requisitos, bem como os valores que precisam ser alcançados para que este trabalho logre sucesso.

1.2 Contribuições

A principal contribuição deste trabalho é a proposta de uma ferramenta de simulação discreta que facilite os trabalhos futuros sobre grades P2P e que ajude na validação de trabalhos existentes.

Mais do que isso, espera-se desenvolver e solidificar uma cultura de reuso e empacotamento padronizado de resultados visando o aprimoramento da metodologia envolvida nos trabalhos relacionados ao OurGrid.

1.3 Organização da Dissertação

No Capítulo 2, este trabalho é contextualizado com uma apresentação do estado-da-arte em simuladores discretos de grades computacionais. Esta apresentação é dividida em duas partes: a teoria da simulação discreta e a arquitetura e implementação dos simuladores existentes para grades. Além disso, será apresentada a arquitetura do OurGrid, middleware de grades no qual o modelo do simulador apresentado nesta dissertação é baseado.

No Capítulo 3, são descritos de forma detalhada os requisitos deste simulador, além da apresentação de uma proposta de arquitetura e do relato das técnicas de implementação utilizadas para atingir os requisitos citados.

No Capítulo 4, é feita a validação dos requisitos apresentados no Capítulo 3. Essa validação envolve testes de aceitação para obter os valores de cobertura previamente definidos, executar simulações para comprovar estatisticamente o atendimento dos requisitos de velocidade e escalabilidade e mensurar o esforço de adaptação do simulador para diferentes necessidades, avaliando assim sua extensibilidade.

No Capítulo 5, são apresentados três relatos de extensão deste simulador para finalidades específicas: o primeiro é relativo à implementação de diferentes *schedulers* no componente *Broker*, o segundo ao *design* e implementação de novas estratégias de detecção de falhas e o ultimo à implementação de uma nova política de rede de favores entre os *Peers* de uma grade.

No Capítulo 6, reunimos o conjunto de conclusões desenvolvidas no decorrer deste trabalho bem como perspectivas de trabalhos futuros derivadas das contribuições alcançadas.

Capítulo 2

Trabalhos Relacionados e Fundamentação Teórica

Neste capítulo são apresentados o estado-da-arte em simulação discreta para grades computacionais, contemplando as teorias e técnicas de modelagem e implementação, além da taxonomia das ferramentas existentes, e uma breve descrição da arquitetura do *middleware* OurGrid, sistema cujo modelo é desenvolvido neste trabalho.

2.1 Simuladores Discretos para Grades Computacionais

2.1.1 Teoria e Técnicas de Simuladores de Eventos Discretos

Simulação é a imitação da operação de um processo ou sistema do mundo real ao longo do tempo. Simular envolve a geração de um histórico artificial do sistema representado e a observação de tal histórico para inferir acerca das características operacionais desse sistema [17].

Todo simulador possui um modelo do sistema real, que o representa. Neste trabalho são considerados os modelos de simulação baseados em eventos discretos, que são contrastados com outros tipos de modelo, como os modelos matemáticos, modelos descritivos, modelos estatísticos e modelos de entrada-saída. Um modelo baseado em eventos discretos tenta representar os componentes de um sistema e suas interações na extensão do estudo em questão. Para isto, esse tipo de modelo inclui uma representação detalhada das partes internas do

sistema.

Além disso, modelos baseados em eventos discretos são dinâmicos, ou seja, a passagem do tempo tem um papel crucial na simulação. Nesse tipo de simulação as variáveis de estado mudam apenas em pontos discretos no tempo em que os eventos ocorrem. A simulação é então conduzida através do tempo por um mecanismo que adianta o tempo simulado, também chamado de motor de simulação.

Na idealização de um modelo de simulação e em sua implementação é desejável que alguns princípios básicos [50] sejam seguidos. Primeiro, conceituar um modelo requer conhecimento do sistema. Antes de criar um modelo de um sistema, é necessário entender a estrutura e as regras operacionais deste e ser capaz de extrair sua essência sem incluir detalhes desnecessários. Bons modelos devem ser de fácil compreensão e, ao mesmo tempo, ter detalhes suficientes para refletir as características importantes do sistema.

Segundo, um modelo deve ser facilmente remodelável. A construção de modelos deve ser iterativa, uma vez que estes são continuamente refinados, atualizados, modificados e estendidos. As seguintes técnicas dão suporte a esse tipo de abordagem:

1. Desenvolver formas de entrada e interfaces facilmente adaptáveis.
2. Dividir o modelo em elementos lógicos relativamente pequenos.
3. Separar elementos lógicos (atividades) e físicos (entidades) no modelo.
4. Desenvolver e manter a documentação diretamente no modelo.
5. Deixar ganchos para a fácil extensão ou modificação do modelo, ou seja, construir um modelo *open-ended*.

Terceiro, o processo de modelagem é evolucionário. O ato de modelar e executar o modelo revela, pouco a pouco, informações importantes que amadurecem a relação entre o sistema estudado e o modelo. A correspondência resultante entre o modelo e o sistema não só estabelece o próprio modelo como uma ferramenta para a solução de problemas, mas provê familiaridade dos desenvolvedores com o sistema e cria um veículo de treinamento para futuros usuários.

Quanto à implementação dos modelos baseados em eventos discretos, uma abordagem de forte apelo é a modelagem orientada a objetos [42]. Com os benefícios herdados da

orientação a objetos, como encapsulamento, herança, polimorfismo, *late binding* e tipagem parametrizada, é muito mais fácil seguir os princípios citados anteriormente. Além disso, a analogia entre as entidades do modelo e objetos é intuitiva.

Joines e Roberts [42] também sugerem um modelo de *framework* orientado a objetos para a criação de simuladores discretos. Nesse *framework* são definidas duas partes claras: um pacote de utilitários de simulação, ilustrado na Figura 2.1, no qual são definidos o motor de simulação, geradores de números aleatórios, coletores de estatísticas, entre outras classes que dão suporte à simulação; e o pacote de hierarquia do modelo, ilustrado na Figura 2.2, que contém as classes abstratas para criação de entidades e eventos. No topo dessas classes, o autor sugere a criação de uma API (*application programming interface*) para uma execução em alto nível de um modelo personalizado.

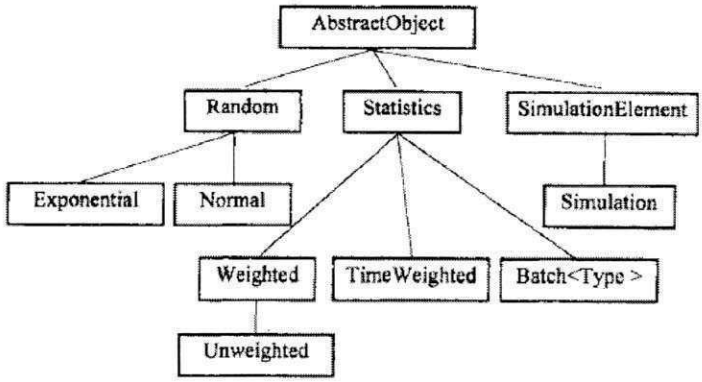

Figura 2.1: Framework de simulação: suporte à simulação

O *framework* de simulação e o modelo básico propostos neste trabalho foram desenvolvidos à luz dos princípios citados, contemplando ao máximo as boas práticas de modelagem e implementação descritos.

2.1.2 Estado-da-arte em ferramentas de experimentos para grades computacionais

Nesta seção é apresentado o estado da arte das ferramentas de experimentação em grades computacionais, classificadas em quatro grandes categorias [26]: plataformas de experimentos, emuladores, simuladores de rede e simuladores de aplicação. Mais especificamente, os

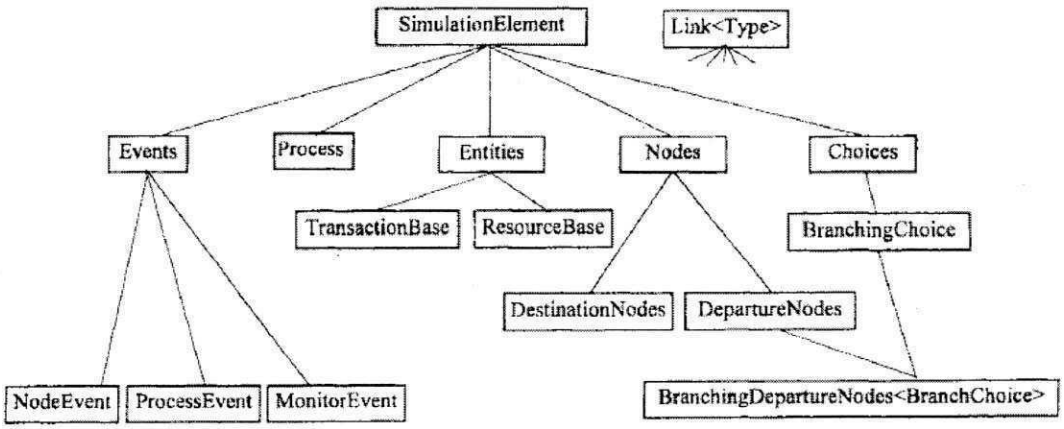

Figura 2.2: Framework de simulação: frame de modelagem

seguintes critérios foram utilizados:

- **Modelagem dos recursos**: Recursos em uma grade (CPU, rede, discos), podem ser modelados de diferentes formas: (i) modelagem puramente analítica (matemática); (ii) por meio de uma cadeia de eventos discretos; (iii) virtualização do recurso; e (iv) utilização do próprio recurso no mundo real.

- **Modelagem da aplicação e requisitos**: Algumas plataformas sugerem uma API por meio da qual a aplicação é descrita no modelo, em outras, a aplicação real pode ser utilizada.

- **Controle das configurações do experimento**: O controle do ambiente do experimento é essencial para sua reprodutibilidade. Enquanto algumas plataformas dão suporte a tal controle, em outras simplesmente não é possível definir estas configurações.

- **Escalabilidade**: Compreende a quantidade de nós que cada ferramenta pode suportar e varia principalmente de acordo com o interesse da comunidade de pesquisa. De qualquer forma, é um valor que não deve ser avaliado por si só. Sua avaliação deve ser acompanhada de outras métricas de interesse do pesquisador, como velocidade de execução.

Plataformas de experimentos

A principal vantagem de executar experimentos em plataformas como o Grid'5000 [24] ou o PlanetLab [29] é ter resultados intrinsecamente verossímeis, uma vez que não carregam o possível viés de um modelo criado. Por outro lado, a baixa escala, a execução em tempo real e a falta de controle sobre as configurações do experimento, o que impede sua reprodutibilidade, são as principais limitações desse tipo de ferramenta.

Emuladores

Soluções emuladas permitem que as aplicações originais (sem modificações) executem num ambiente específico e controlado aonde chamadas de sistema são interceptadas e tratadas pelo emulador. Dois exemplos desse tipo de ferramenta são o ModelNet [59] e o MicroGrid [61]. Nos dois casos o emulador injeta o tráfego numa LAN emulada, atrasando o movimento dos pacotes para imitar um determinado ambiente de execução. Apesar da tecnologia de emulação utilizada e o benefício de rodar os experimentos em um ambiente controlado, essas ferramentas não escalam bem, conseguindo executar em torno de apenas 100 nós simultâneos.

Simuladores de rede em nível de pacote

Uma série de simuladores de rede têm sido desenvolvidos e amadurecidos ao longo dos últimos anos. O mais famoso deles é o ns-2 [8], que implementa uma grande coleção de protocolos e modelos de filas. Outros simuladores que se destacam são o SSFNet [33] e o GTNetS [53], que são facilmente extensíveis e que atendem uma escala na ordem de centenas de milhares de nós.

O grande problema desse tipo de simulador para a proposta deste trabalho é que eles foram implementados por e para os pesquisadores em redes. Isso os torna mais difíceis de adaptar a simulações de aplicações, nas quais as preocupações estão em um nível mais alto da pilha de comunicação. Além disso, o grande detalhamento das operações em baixo nível faz com que os tempos de execução sejam degradados.

Simuladores de aplicação

Para contornar as dificuldades e limitações dos outros tipos de ferramentas, a comunidade de pesquisa em sistemas distribuídos acabou desenvolvendo uma série de simuladores orientados à aplicação nos últimos anos. Porém, boa parte dessas ferramentas só são utilizadas uma vez, ou, quando abertas, tendem a atender uma comunidade bastante específica.

Como pode-se notar na Tabela 2.1, as diferenças entre os objetivos levam a grandes diferenças no projeto das ferramentas. Enquanto ferramentas para P2P, como PlanetSim [43] e PeerSim [12] buscam alta escalabilidade, elas não modelam CPU ou a própria rede.

Já as ferramentas para grades, como ChicSim [52], OptorSim [19], GridSim [23] e SimGrid [26], buscam um compromisso entre o tempo de execução e o nível de detalhe do modelo, uma vez que a principal métrica de desempenho neste contexto é o *makespan* da aplicação.

Particularmente sobre simuladores do OurGrid, é possível citar o o Efficient Grid Simulator [4], o SRSSim [13] e o OGSim [9]. Este último, o mais genérico deles, foi reutilizado em alguns trabalhos relacionados ao *middleware*. Apesar de lidar com diversos aspectos do OurGrid, como aplicações *Bag-of-Tasks* (BoT), aplicações paralelas cujas tarefas são independentes [32, 58], e a implementação de uma Rede de Favores (NoF), o OGSim peca por não ter um modelo genérico do OurGrid, deixando de lado importantes características do sistema, como o escalonamento em nível de usuário, além de não modelar a camada de rede, e, consequentemente, detecção de falhas. Por fim, o OGSim não descreve claramente ganchos para sua extensão, dificultando seu reuso.

Dada a taxonomia das ferramentas existentes, este trabalho propõe um modelo de simulação mais flexível, que permite ao desenvolvedor implementar de forma simples, diferentes mecanismos para a representação da CPU, armazenamento, rede e aplicação. Dessa forma, o compromisso entre escala, tempo de execução das simulações e nível de detalhamento de modelo deve se moldar às necessidades da pesquisa. Em paralelo, completude e extensibilidade são apontados como requisitos importantes deste modelo, uma vez que são características que facilitam o seu reuso. Mais detalhes destes requisitos são apresentados no Capítulo 3.

Tabela 2.1: Comparação entre as ferramentas existentes para grades computacionais

	CPU	Disco	Rede	Aplicação	Requisitos	Configuração	Escala
Grid'5000	direta	direto	direta	direta	acesso	fixa	< 5000
PlanetLab	virtualizada	virutalizado	virtualizada	virtualizada	acesso	não controlada	< 850
ModelNet	-	-	emulada	emulada	nenhum	controlada	< 100
MicroGrid	emulada	-	eventos	emulada	nenhum	controlada	< 100
ns2	-	-	eventos	eventos	C++ e Tcl	controlada	< 1000
SSFNet	-	-	eventos	eventos	Java	controlada	< 100.000
GTNetS	-	-	eventos	eventos	C++	controlada	< 180.000
ChicSim	eventos	-	eventos	eventos	C	controlada	< 1000
OptorSim	eventos	fixo	analítico/eventos	eventos	Java	controlada	< 100
GridSim	eventos	eventos	eventos	eventos	Java	controlada	< 100
PlanetSim	-	-	constante	eventos	Java	controlada	< 100.000
PeerSim	-	-	-	máquina de estados	Java	controlada	< 1.000.000
SimGrid	eventos	-	analítico/eventos	emulada/eventos	C ou Java	controlada	< 10.000
OGSim	analítica	-	-	analítica	Java	controlada	< 100.000

2.2 OurGrid

O OurGrid é um *middleware* de computação em grades P2P que dá suporte à execução de aplicações BoT. O objetivo do OurGrid é aumentar a capacidade computacional dos laboratórios espalhados pelo mundo, por meio do compartilhamento de seus recursos ociosos.

Os principais componentes do OurGrid são o *Broker*, o *Worker*, o *Peer* e o *Discovery Service*. *Brokers* e *Workers* fazem parte de um *site* administrado por um *Peer*. *Peers* se descobrem por meio do *Discovery Service*, formando uma comunidade OurGrid. A arquitetura do OurGrid é ilustrada na Figura 2.3.

O *Broker* é a entidade que interage com o usuário e é responsável pela submissão dos *jobs* e escalonamento de tarefas para *Workers*. O *Worker* roda nos recursos computacionais destinados à execução de tarefas e o *Peer* é responsável pela descoberta de recursos.

Uma instalação OurGrid tenta ser não-intrusiva no sentido de que o usuário local tem sempre maior prioridade no acesso aos recursos locais. Na prática, a submissão de uma tarefa local que demanda recursos alocados a entidades remotas sempre provoca a preempção desses recursos. Essa regra assume que o OurGrid nunca degradará o desempenho local.

Além disso o OurGrid é projetado para ser escalável, tanto no sentido do número de sites que uma comunidade pode suportar, quanto na facilidade da entrada de um novo site no sistema. Para habilitar tal escala, uma comunidade OurGrid é projetada como uma rede

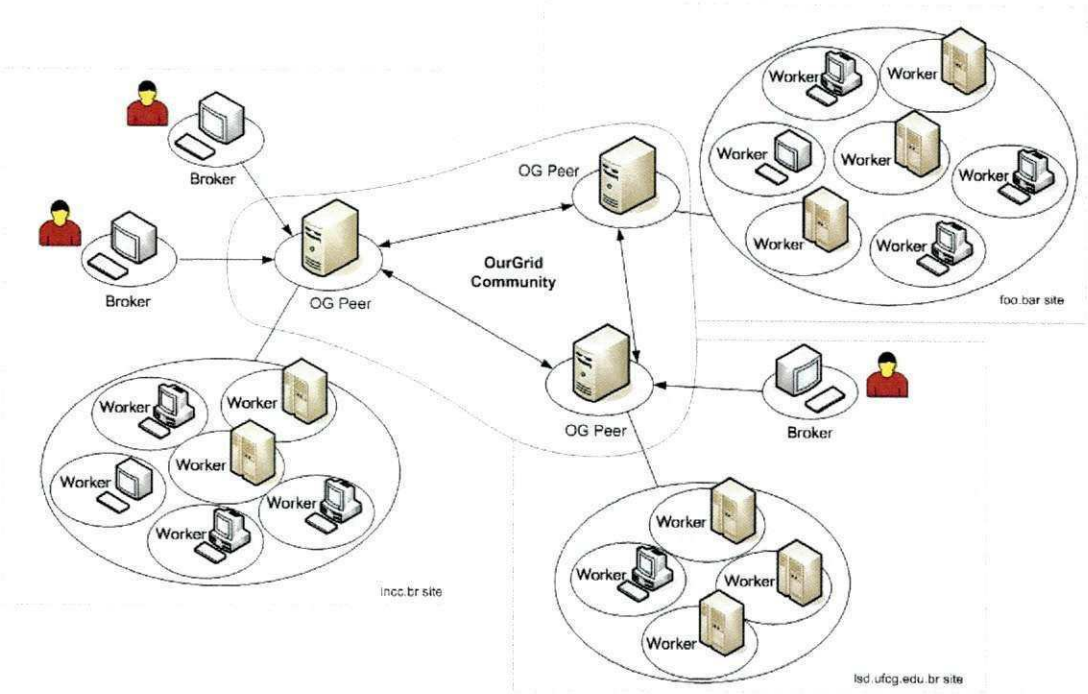

Figura 2.3: Arquitetura do OurGrid

peer-to-peer, com cada *site* correspondendo a um *peer* no sistema. Para evitar o recorrente problema de *freeriding* em redes entre pares [41,54], em que nós do sistema só se interessam em receber recursos e nunca doam recursos de volta para a comunidade, o OurGrid implementa um algoritmo de incentivo chamado Network-of-Favors (NoF), um mecanismo de crédito por doação decentralizado e autônomo que acaba marginalizando os nós *freeriders*.

Atualmente o OurGrid faz uso do *framework* Commune [60], que provê uma pilha de comunicação sobre o protocolo XMPP [36]. Componentes implementados com o Commune seguem o padrão de objetos distribuídos, e todas as chamadas remotas a procedimento são assíncronas. O Commune também fornece à aplicação um modelo de segurança baseado em autenticação com pares de chaves assimétricas e certificados X.509, descreve e implementa um protocolo de conexão, promove um processamento sequencial das mensagens, evitando que o desenvolvedor lide com a complexidade do modelo de *Threading*, e fornece um mecanismo distribuído de detecção de falhas.

O OurGrid é totalmente baseado numa arquitetura orientada a eventos, na qual não há acoplamento entre os processadores de mensagem ou entre os processadores e o *framework* de comunicação. O esquema de objetos distribuídos associado à lógica de *publish-subscribe*

dos processadores, que são classificados em processadores de requisição e de resposta, tornam o OurGrid um sistema flexível e de fácil adaptação a mudanças. A Figura 2.4 apresenta um diagrama de sequência que ilustra a arquitetura de processamento de mensagens adotada no OurGrid.

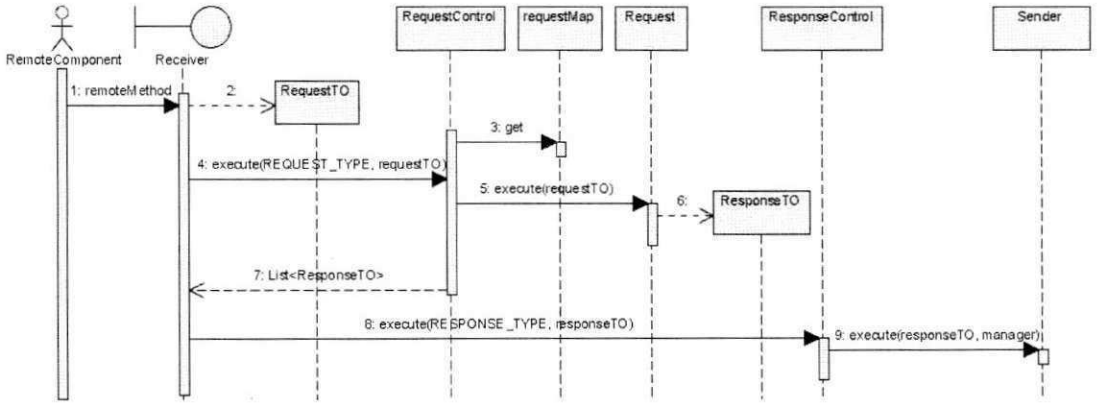

Figura 2.4: Diagrama de sequência de uma chamada remota no OurGrid

Capítulo 3

Uma Proposta de um Simulador Discreto do OurGrid

Como foi antecipado no Capítulo 1, este trabalho apresenta uma proposta de um simulador discreto para o OurGrid. Neste capítulo descrevemos os requisitos deste simulador, uma proposta para sua arquitetura e as técnicas de implementação utilizadas para atender os requisitos citados.

3.1 Requisitos

Com base no levantamento bibliográfico realizado, os seguintes requisitos (funcionais e não funcionais) para o simulador de eventos discreto do OurGrid foram levantados: completude e precisão; escalabilidade e velocidade; extensibilidade; e testabilidade.

3.1.1 Completo e coerente

Para generalizar o modelo deste simulador, é necessário partir do sistema real, o próprio OurGrid. Como este modelo não pressupõe um objetivo de pesquisa específico, o escopo de modelagem foi o mais amplo possível, tentando contemplar todos os aspectos do OurGrid, desde a camada de rede até os algoritmos de distribuição do *Discovery Service*.

Além de completo, este modelo deve ser coerente. A alteração das variáveis de entrada no modelo e no sistema real - sejam quantidade e tamanho de tarefas; quantidade de *sites*; quan-

tidade, disponibilidade e capacidade de Workers; etc. - deve refletir de forma semelhante nas métricas de interesse desse contexto, como utilização e *makespan*. Assim, espera-se que os valores das métricas correspondentes para o modelo e o sistema real sigam tendências semelhantes dadas as mesmas entradas.

3.1.2 Mantendo o compromisso: escalabilidade e desempenho

Outro principal requisito do simulador é a escalabilidade, que classicamente é avaliada por meio da métrica quantidade de nós que uma simulação pode suportar sem erros de execução, como estouro de memória. Porém, de acordo com Casanova et al. [27], o valor de escala por si só não é representativo. Essa métrica deve ser acompanhada dos tempos de execução para se tornar representativa na prática.

Sendo assim, definiu-se como valor mínimo aceitável para a escala do modelo 50000 nós, sendo 1000 *sites* com 50 *workers* cada, valor baseado nos valores máximos de escala alcançados pelos simuladores destinados a grades P2P vistos na Tabela 2.1 e nos planos de expansão da comunidade OurGrid, como o GridUFCG [22]. Essa execução deve considerar a modelagem de rede e de detecção de falhas, e deve executar em contenção, que é o pior caso quando se trata de utilização dos recursos. Espera-se que este cenário execute em menos de 10 horas numa máquina Intel Xeon 2,4 GHz com 1GB de memória RAM, valor aceitável quando comparado aos tempos de execução de modelos de grade com outras ferramentas, como o SimGrid [51].

3.1.3 Extensível

Para se adaptar às diferentes necessidades de pesquisa, este modelo de simulação deve ser facilmente extensível. O esforço de desenvolvimento para criar ou estender funcionalidades, modelar um camada na pilha de comunicação, modificar algoritmos de escalonamento, etc., não deve ser muito maior do que o custo intrínseco de implementação do problema.

Considerando um projeto orientado a objetos, para avaliar a extensibilidade desse modelo utilizaremos algumas das métricas de complexidade propostas por Martin et al. [46], que seguem:

- **Acoplamento Aferente (Aa)**: o número de outros pacotes que dependem de classes

dentro do pacote como indicador de sua responsabilidade.

- **Acoplamento Eferente (Ae)**: o número de outros pacotes que as classes no pacote dependem é um indicador de independência do pacote.

- **Abstração (A)**: a razão entre o número de classes abstratas e interfaces analisadas no pacote e o número de classes deste pacote. O valor varia de 0 a 1. 0 indica um pacote totalmente concreto e 1 indica um pacote totalmente abstrato.

- **Instabilidade (I)**: a razão entre o Acoplamento Eferente e a soma dos Acoplamentos Aferente e Eferente. I = Ae / (Ae + Aa). Indica a capacidade que um pacote tem de passar por mudanças. 0 indica um pacote totalmente estável e 1 indica um pacote totalmente instável. Ou seja, se o número se aproxima de 1, modificações sem efeitos colaterais em tal pacote são difíceis.

- **Distância da Sequência Principal (DSP)**: A distância perpendicular de um pacote a partir de uma linha idealizada é tal que Abstração + Instabilidade = 1 (A + I = 1). Isto é um indicador do equilíbrio entre a abstração e a estabilidade do pacote. Pacotes ideais são completamente abstratos e estáveis (A=1, I=0) ou completamente concretos e instáveis (A = 0, I = 1).

Para avaliar essas métricas foi utilizado o software JDepend [5], que analisa o código Java gerando as métricas de qualidade de projeto citadas anteriormente para cada pacote de código fonte.

Além disso, no Capítulo 5 são reportados os passos necessários para a adaptação do simulador em três casos de uso reais, reproduções de trabalhos de pesquisa e avaliações sobre o OurGrid. O primeiro deles é a a implementação e avaliação de cinco detectores de falhas adaptativos para o OurGrid; o segundo aborda o trabalho de Paranhos et al. [57], que avalia diferentes heurísticas de escalonamento no *Broker* OurGrid; e por último o trabalho de Gaudencio et al. [39], que analisa uma nova política para a rede de favores.

3.1.4 Testável

Testabilidade de *software* é o grau em que um artefato dá suporte a testes em um determinado contexto (objetivos, técnicas e recursos do teste). A testabilidade de componentes de

software (módulos, classes) é determinada por fatores [21] tais como:

- **Controlabilidade**: A capacidade de controlar o estado do componente a ser testado como requisitado pelo teste.
- **Observabilidade**: A capacidade de observar o estado do componente e os resultados (intermediários e final) do teste.
- **Isolabilidade**: A capacidade de um componente ser testado de forma isolada.
- **Automatibilidade**: A possibilidade de automatizar os testes dos componentes do sistema.

É necessário implementar um *framework* de testes que viabilize o cumprimento dos fatores acima para um simulador discreto baseado em eventos. Este *framework* deve: habilitar uma simulação passo-a-passo, o que possibilitaria o controle da execução; entregar ao testador o estado do modelo simulado e os eventos recém executados, o que habilita a observabilidade; permitir a adição de eventos isolados e em tempo de execução, o que habilita a isolabilidade; e permitir a definição da topologia e dos eventos de forma programática, o que possibilita a automação dos testes.

3.2 Arquitetura

O projeto da arquitetura deste simulador visa primordialmente sua extensibilidade. Partiu-se de uma arquitetura básica de um simulador discreto baseado em eventos, já descrita nos trabalhos relacionados. Essa arquitetura consiste no pacote de **Eventos**, o pacote de **Entidades**, os **Utilitários de simulação**, que contêm os coletores de estatísticas, por exemplo, e o **Motor de simulação**, junto com a fila de eventos. A partir deste projeto inicial, foram utilizadas técnicas que diminuem o acoplamento entre os eventos e entre o simulador e as formas de entrada.

Fábricas abstratas [38] para a topologia de entidades e para os eventos foram introduzidas, junto com o conceito de **Especificação de evento** (*EventSpec*), que é a análogo a uma mensagem remota no contexto de RPC. Assim, tal qual o OurGrid, o simulador segue uma arquitetura orientada a eventos, que são criados dinamicamente em tempo de execução.

Além disso, a fila de eventos é acompanhada de um **Proxy de eventos**, que abstrai o carregamento em páginas dos eventos. Isso flexibiliza a implementação deste *proxy*, que pode carregar eventos de arquivos, bancos de dados, da memória, etc.

A fila de eventos é alimentada pelo *proxy* de eventos, que provê eventos exógenos, e pela execução dos próprios eventos, que pode criar eventos endógenos. A implementação concreta do evento a ser executada depende da referência de classe presente na fábrica de eventos no momento da sua criação e enfileiramento.

A Figura 3.1 ilustra a arquitetura deste simulador no que diz respeito a sua organização em componentes. Os detalhes de implementação que possibilitaram o atendimento aos requisitos do simulador são descritos na seção seguinte.

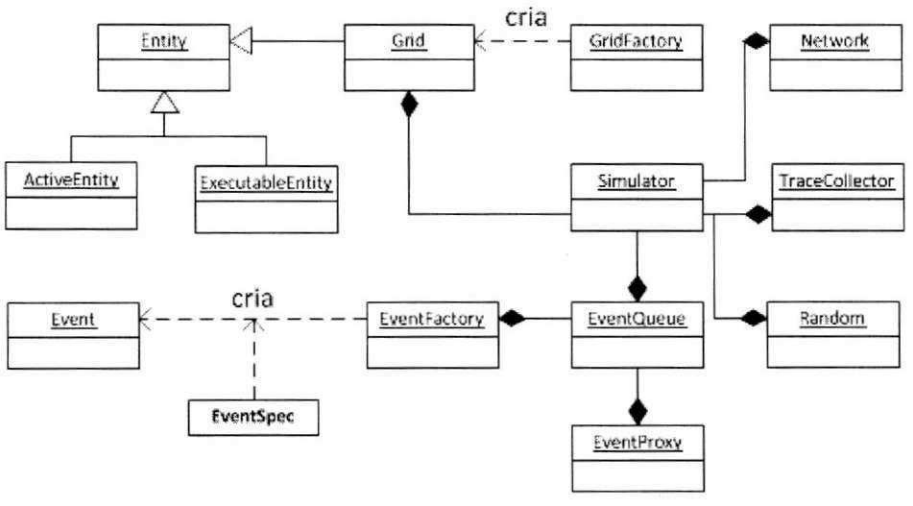

Figura 3.1: Arquitetura do Simulador: Componentes principais

3.3 Implementação

3.3.1 Tecnologias utilizadas

Este simulador discreto orientado a eventos foi implementado em Java [10], com o auxílio das bibliotecas Apache Commons [1], para simplificar as operações de E/S e matemáticas; SSJ [14], para as operações estatísticas relacionadas a simulação estocástica; JSON-java [3], para manipular estruturas no formato JSON; jUnit [6], para a criação dos testes automáticos; e jDepend [5] e Clover [2] para a geração de métricas de qualidade.

Trecho de código 3.1: Exemplo de utilização de um GridFactory.

```
1 GridFactory gridFactory = new JsonGridFactory(properties,
2     new FileInputStream("resources/grid-example.conf"));
3 Grid grid = gridFactory.createGrid();
```

3.3.2 Modelando as entidades e criando a topologia

As entidades deste modelo foram divididas em **entidades ativas** e **entidades executáveis**. Entidades são identificáveis, mantêm estado, e podem ser estendidas em tempo de execução, uma vez que possuem uma estrutura de dados de dicionário para guardar qualquer tipo de informação.

As entidades ativas são modelos dos componentes do OurGrid: Worker, Broker, Peer e DiscoveryService. São entidades identificáveis, possuem um estado de vida (*UP* ou *DOWN*) e dados de monitoramento sobre outras entidades.

As entidades executáveis são estruturas relacionadas a tarefas submetidas à grade: Job, Task e Replica. Têm um estado de execução (*UNSTARTED*, *RUNNING*, *FAILED*, *FINISHED*, *CANCELLED* ou *ABORTED*), e marcas de tempo para seu início e fim.

O Grid também é uma entidade, que mantêm as referências para todas as entidades ativas. O Grid é o ponto de entrada para a representação da topologia do modelo.

Um Grid é criado através de um **GridFactory**, interface que abstrai a criação da topologia. O simulador provê um JsonGridFactory, que lê arquivos no formato JSON e traduz para instâncias de Grid. Esse *design* permite a extensão em duas frentes: a criação de novas fábricas de Grid; e a modificação das fábricas existentes visando a modificação das entidades. O código em 3.1 exemplifica a utilização de um GridFactory em particular, o JsonGridFactoy, que aceita arquivos de descrição no formato JSON.

3.3.3 Modelando os eventos e sua criação dinâmica

Um evento neste simulador, além de representar o conceito de evento da teoria de simulação discreta, contendo um tempo de execução e sua prioridade na fila, é análogo a uma mensagem de chamada remota a procedimento no OurGrid e seu respectivo processamento.

Uma classe de evento carrega a lógica de processamento para uma mensagem específica,

Trecho de código 3.2: Exemplo da criação dinâmica e enfileiramento de eventos.

```
1 Event requestWorkersEvent = ourSim.createEvent(PeerEvents.REQUEST_WORKERS,
2     time, peer.getId(), request.getSpec(), true);
3 ourSim.addEvent(requestWorkersEvent);
```

e, quando instanciada, possui os parâmetros necessários para esse processamento.

Como já foi antecipado, neste modelo introduzimos o conceito de **EventSpec**, que separa a lógica do evento de seu tipo e parâmetros, e promove a fácil criação e modificação dos eventos do modelo. Eventos são construídos por meio de uma **EventFactory**, que tem associações cadastradas entre nomes de eventos e suas respectivas classes. Assim, os eventos são criados por meio de reflexão em tempo de execução. No trecho de código 3.2 tem-se um exemplo de criação dinâmica de evento, no qual é possível notar que não há instanciação de classes concretas.

A fila de eventos (**EventQueue**), estrutura que armazena os eventos a serem processados pelo motor de simulação, é alimentada tanto pelo **EventProxy** como por consequência do processamento de outros eventos. A implementação do EventProxy tem um papel importante na manutenção de um registro de memória baixo neste simulador, já que a sua interface define requisições paginadas de eventos, evitando a saturação da fila com eventos a serem executados em longo prazo.

3.3.4 Modelando a rede

Em princípio, este simulador modela a rede de forma analítica. A interface **Network** provê um único método para geração dos atrasos da rede, independente do contexto. Todo evento enfileirado através do método *addNetworkEvent* tem adicionado ao seu tempo simulado um atraso gerado por uma instância de Network. Uma implementação da rede que retorna atraso igual a zero desativa o modelo de rede. O trecho de código 3.3 exemplifica a adição de um evento que será afetado pelo modelo de rede. Com a chamada do método *addNetworkEvent*, o evento *WORK_FOR_BROKER* tem seu tempo de execução adiado internamente por uma implementação de *Network*, e, consequentemente, sua posição na fila de eventos é modificada.

Trecho de código 3.3: Exemplo do enfileiramento de eventos passando pelo modelo de rede.

```
1 ourSim.addNetworkEvent(ourSim.createEvent(WorkerEvents.WORK_FOR_BROKER,
2     time, request.getConsumer(), peer.getId(), request.getSpec(), workerId));
```

Apesar de ser modelada de forma analítica, a rede pode ser facilmente modificada para contemplar mais detalhes de uma pilha de comunicação, sendo modelada, por exemplo, por eventos discretos.

3.3.5 Modelando a detecção de falhas

O modelo de detecção de falhas neste simulador aborda o processamento de todas as mensagens de controle trocadas no OurGrid para a monitoração de objetos. O OurGrid implementa um modelo de detecção *pull* e totalmente distribuído, no qual entidades monitoras mandam mensagens de *isItAlive* para entidades monitoradas, que respondem com a mensagem de *updateStatus* quando estão ativas. Para se interessar numa entidade, a entidade monitora invoca o método *registerInterest*, passando como parâmetro a assinatura dos métodos de *callback* que serão responsáveis por tratar a falha e a recuperação da entidade de interesse.

O monitoramento no OurGrid, tal qual no modelo deste simulador, é bilateral. Toda entidade monitora é obrigatoriamente monitorada, formando, conceitualmente, uma conexão.

Subindo para o nível da aplicação, os componentes precisam definir quais métodos, (ou eventos, no caso do simulador), vão tratar a falha e a recuperação das entidades monitoradas. Além disso, para registrar interesse em uma determinada entidade, o método *registerInterest* é invocado, e quando um componente não tem mais interesse na monitoração de um objeto, essa invoca o método *release*, passando como parâmetro sua referência do objeto remoto. No OurGrid, para cada objeto monitorado uma nova *Thread* é criada, que verifica, de tempos em tempos, se uma falha deve ser reportada.

No modelo de simulação a implementação da detecção de falhas é bastante semelhante. Cada entidade ativa tem um conjunto de monitores e uma referência para um oráculo detector de falhas, que implementa a interface **FailureDetector**. O trecho de código 3.4 ilustra, nas primeira linhas, invocações dos métodos *addOnRecoveryEvent* e *addOnFailureEvent*, que têm o objetivo de cadastrar eventos de notificação, e, nas linhas seguintes, é ilustrada a

Trecho de código 3.4: Cadastro de eventos de notificação e registro de interesse.

```
1 peer.addOnRecoveryEvent(Worker.class, PeerEvents.WORKER_AVAILABLE);
2 peer.addOnFailureEvent(Worker.class, PeerEvents.WORKER_FAILED);
3
4 for (String workerId : peer.getWorkersIds()) {
5     MonitorUtil.registerMonitored(ourSim, getTime(),
6         peer.getId(), workerId);
7 }
```

criação de interesse em uma entidade, por meio do método *MonitorUtil.registerMonitored*.

Tal qual a modelagem de rede, a detecção de falhas pode ser desativada. Assim a percepção de recuperação e de falha será consequência instantânea do ativamento e desativamento dos componentes.

A principal diferença na implementação do simulador é a adição de um evento de **liveness check**, que é escalonado de tempos em tempos e verifica, junto ao FailureDetector, se os objetos monitorados falharam. Isso substitui as várias *Threads* da solução do OurGrid.

3.3.6 Coletores de traços e geração de números aleatórios

Para dar suporte às execuções, este simulador provê uma interface para coleta de traços (**TraceCollector**). Atualmente esta interface deixa ganchos para o término das entidades executáveis (replica, task e job). Sua implementação padrão escreve em um arquivo CSV (*comma separated values*) os dados de finalização destas entidades. Quando da necessidade do pesquisador, essa interface pode ser estendida e novas implementações podem ser criadas.

Além disso, a fachada do simulador provê uma instância da classe *java.util.Random*, que fornece métodos para a geração de números aleatórios. Essa instância pode ser inicializada com diferentes valores de semente.

3.3.7 Criando um *framework* de testes

Para garantir os requisitos de um software testável (controlabilidade, observabilidade, isolabilidade e automatibilidade), foi criado um *framework* de testes para este simulador.

Trecho de código 3.5: Exemplo da utilização do framework de testes.

```
1  addEvent(new EventSpec(BrokerEvents.BROKER_UP, 0, brokerId));
2  addEvent(new EventSpec(PeerEvents.PEER_UP, 1, peerId));
3
4  List<Event> secondary = addEvent(new EventSpec(
5          BrokerEvents.ADD_JOB, 2, brokerId + " " + jsonJob1));
6
7  Assert.assertTrue(EventRecorderUtils.hasEvent(secondary,
8          PeerEvents.REQUEST_WORKERS));
9  Assert.assertEquals(1, broker.getJobs().size());
10 Assert.assertEquals(1, broker.getRequests().size());
```

Este *framework* permite que o testador:

- **Execute a simulação passo-a-passo**: O testador pode adicionar eventos primários à fila e executá-los passo a passo, o próprio evento e os eventos secundários consequentes são processados e a simulação retorna. Isso possibilita a verificação do estado do modelo em momentos específicos da simulação.

- **Pare o simulador na ocorrência de um evento**: O testador pode definir tipos de eventos secundários cujas ocorrências fazem a simulação parar.

- **Recuperar eventos secundários e verificar um comportamento esperado**: Um passo de simulação pode retornar todos os eventos secundários criados por uma determinado evento adicionado. De posse do evento, classes utilitárias deste *framework* provêm métodos para verificação de comportamento, como *hasEvent* e *hasEventSequence*. Esse tipo de verificação de comportamento faz analogia à tecnologia de *Mocks* [44].

O trecho de código 3.5 é retirado de uma das classes de teste utilizando o framework em questão, e mostra a aplicação dos conceitos citados acima.

Capítulo 4

Validação dos requisitos

Neste capítulo são apresentadas as técnicas de validação e são validados os requisitos funcionais e não-funcionais para um simulador do OurGrid discreto baseado em eventos apresentados no Capítulo 3.

4.1 Completude e coerência

Como antecipado no Capítulo 3, o modelo deste simulador deve ser completo e coerente. Para avaliar sua completude, recorreu-se aos testes de aceitação do OurGrid. Usando o *framework* de testes desenvolvido, os próprios testes do OurGrid foram convertidos em testes deste modelo. Entende-se que essa abordagem dará completude ao modelo uma vez que a implementação do OurGrid foi guiada por estes testes. Já a coerência foi avaliada por meio de validação externa, comparando resultados de experimentos no OurGrid em cenário de produção com resultados de simulações deste modelo.

4.1.1 Validação contra os cenários de aceitação

Para garantir a completude deste modelo, utilizou-se a técnica TDD (*Test Driven Development*) partindo dos testes de aceitação [11] do próprio OurGrid, que foram convertidos em testes deste simulador, aliado a uma ferramenta de cobertura de código, o Clover [2]. Com todos os testes de aceitação implementados, o modelo deve ter pelo menos 80% de seu código coberto pelos testes de aceitação, como recomendado por Cornett [7].

Partindo dos testes de integração do OurGrid, 164 testes de unidade foram implementados sob este modelo. Estes testes utilizaram o *framework* descrito na seção 3.3.7. A verificação de *Mocks* foi substituída pela verificação dos eventos secundários produzidos pelos eventos adicionados nos testes, viabilizada pela técnica de simulação passo-a-passo.

Com a instrumentação do código-fonte do simulador aliada à execução do Clover, notou-se uma cobertura de 94,1% do código fonte, atendendo ao requisito de completude levantado no Capítulo 3.

O Anexo A apresenta o relatório completo de pacotes reportado pelo Clover.

4.1.2 Validação contra os resultados do OurGrid em ambiente de produção

A precisão do modelo deve ser garantida por meio de uma validação externa contra o próprio OurGrid, executando em ambiente de produção. O *makespan* das tarefas, principal métrica de interesse na avaliação de grades P2P, foi medido em diversos cenários que contemplam utilização de recursos locais e remotos, preempção, replicação e influência da rede de favores. A hipótese de que o simulador não apresenta valores aceitáveis de *makespan* quando comparado com o OurGrid é refutada por análise gráfica das tendências dos resultados.

Para realizar os experimentos de validação externa, foram separadas algumas máquinas do ambiente de produção real do OurGrid. Todas essas máquinas executam numa mesma LAN, cujo atraso total de rede, incluindo o *overhead* da pilha de comunicação do Commune, é em média 5ms. Todas as tarefas destes experimentos executam um *sleep* de 10 segundos. Cada cenário foi executado 5 vezes, usando sementes distintas para geração de números aleatórios, e o intervalo de confiança apresentado nos gráficos de barras que seguem é de 95%. Tanto o OurGrid quanto o simulador seguem suas configurações padrão, que são iguais. Nas simulações, tanto o modelo de rede quanto a detecção de falhas estão ativados.

A métrica de interesse nos experimentos que seguem é o *makespan* - o tempo necessário para todas as *tasks* executarem. A topologia, a quantidade de *tasks* de cada Broker e o atraso de submissão das tarefas variam de acordo com cada cenário.

Execução local, único Broker

A topologia deste cenário é composta por um Peer, um Broker local e W Workers locais, sendo W = 1, 2, 3 e 4. O Broker submete K tasks, onde K = 1, 2, 4, 20 e 100, valores escolhidos por serem divisores ou multiplos do número máximo de workers, definindo os seguintes fatores de contenção: 0,25, 0,5, 1, 5 e 25. Dois conjuntos de experimentos foram executados: no primeiro fixou-se o K = 100 e o valor de W foi variado; e no segundo fixou-se o W = 4 e o valor de K foi variado.

Nesse cenário pretende-se validar as funcionalidades básicas do modelo, como o atendimento, a persistência e repetição de requisições locais no *Peer*, a execução de tarefas no *Worker* e o escalonamento de tarefas no *Broker*. A Figura 4.1 ilustra a topologia deste cenário, tal que P é o Peer local, B é o Broker local e R_i é um Worker local, tendo $i = 1..W$.

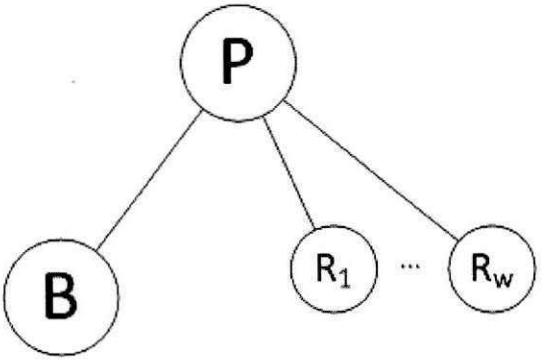

Figura 4.1: Execução local, único Broker - Topologia do cenário

O Gráfico 4.2 compara a variação do *makespan* entre as simulações e os experimentos em ambiente de produção para os dois conjuntos de experimentos.

Execução local, concorrência entre Brokers

Em comparação ao cenário anterior, este cenário adiciona mais um Broker local, e introduz mais uma variável, o atraso de submissão das tarefas pelo segundo Broker T, sendo T = 0, 150, 300 e 500 segundos. Esses valores de atraso marcam o disparo dos *jobs* do segundo *Broker* no mesmo momento, durante, e depois da execução dos *jobs* do primeiro *Broker*. Nesse cenário, três conjuntos de experimentos foram executados: no primeiro fixou-se o K = 100, T = 0 e o valor de W foi variado; no segundo fixou-se o W = 4, T = 0 e o valor de K

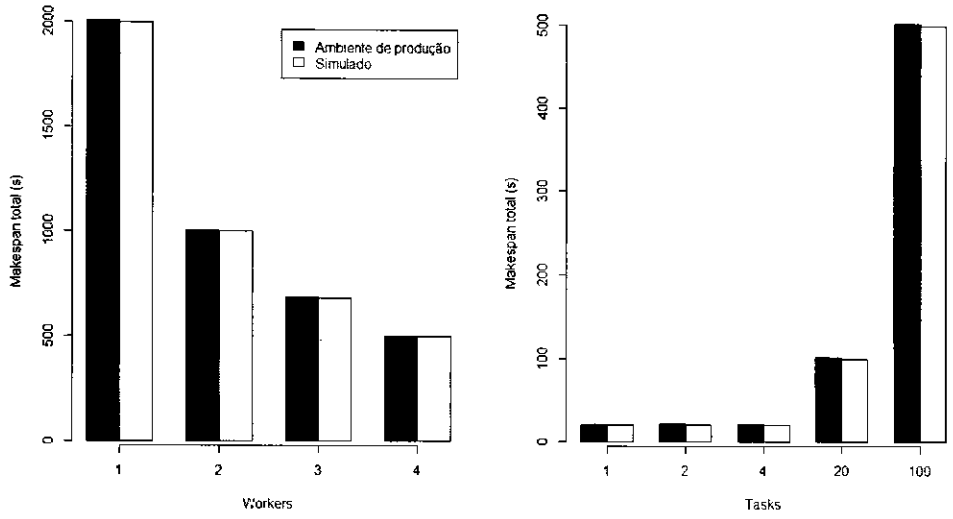

Figura 4.2: Validação externa: Execução local, único Broker

foi variado e no terceiro fixou-se o $W = 4$, $K = 100$ e o valor de T foi variado.

O objetivo deste cenário é avaliar a distribuição uniforme de recursos entre os *Brokers* locais de um *Peer*. A Figura 4.3 ilustra a topologia deste cenário, tal que P é o Peer local, B_A e B_B são os Brokers locais e $R)i$ é um Worker local, tendo $i = 1..W$.

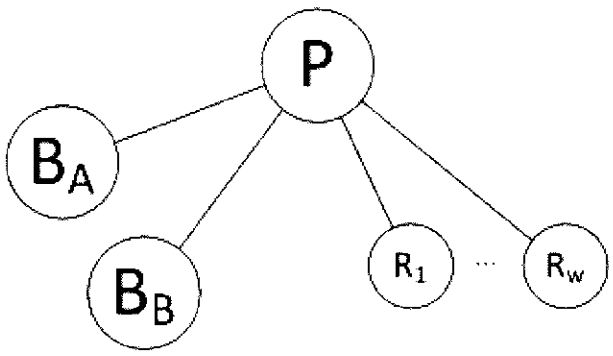

Figura 4.3: Execução local, concorrência entre Brokers - Topologia do cenário

O Gráfico 4.4 compara a variação do *makespan* para os três conjuntos de experimentos.

Execução local e remota, concorrência entre Brokers

Neste cenário o Broker local adicional deixa de existir e um *site* simplesmente consumidor é adicionado à topologia: um Peer remoto e um Broker remoto. O cenário é semelhante ao

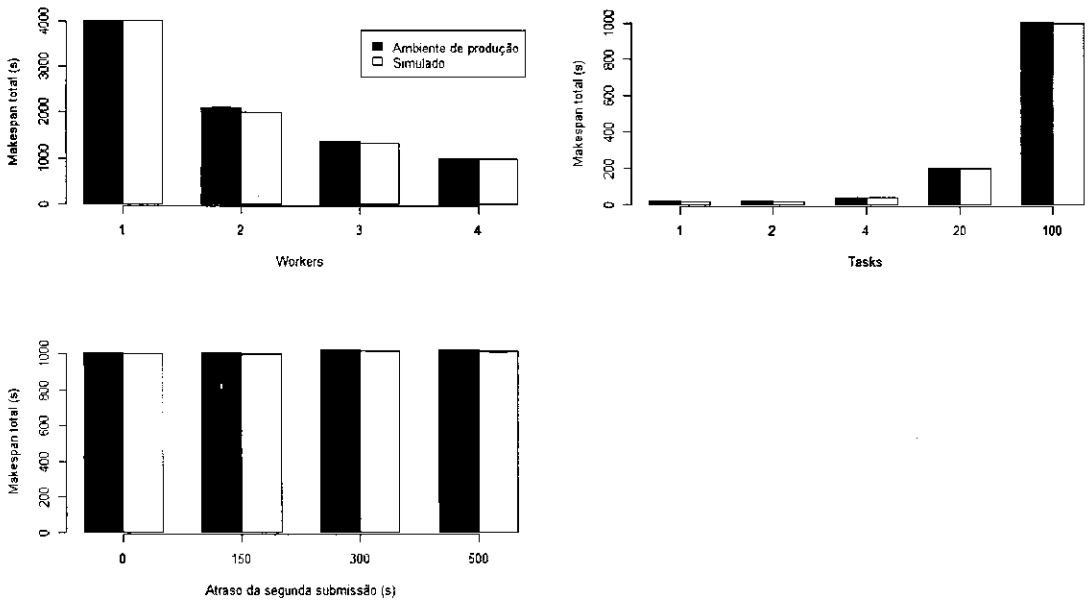

Figura 4.4: Validação externa: Execução local, concorrência entre Brokers

anterior quanto às variáveis, porém *T*, que agora representa o atraso de submissão do Broker remoto, assume os valores 0 e 500 segundos.

O objetivo deste cenário é avaliar a distribuição de recursos entre *Brokers* locais, que devem ter prioridade absoluta, e remotos. A Figura 4.5 ilustra a topologia deste cenário, tal que DS é o Discovery Service, P_L é o Peer local, P_R é o Peer remoto, B_L é o Broker local, B_R é o Broker remoto e $R)i$ é um Worker local, tendo $i = 1..W$.

O Gráfico 4.6 compara a variação do *makespan* para os três conjuntos de experimentos.

Execução remota, concorrência entre Brokers

Neste cenário não há Broker local, apenas Workers, e mais um site remoto simplesmente consumidor é adicionado, ou seja, há dois sites remotos disputando recursos locais. Os valores assumidos pelas variáveis são os mesmos do cenário de execução local, com concorrência entre Brokers.

O objetivo deste cenário é avaliar o efeito da NoF na distribuição de recursos entre *Brokers* remotos de sites simples consumidores. A Figura 4.7 ilustra a topologia deste cenário, tal que DS é o Discovery Service, P_L é o Peer local, P_A e P_B são os Peers remotos, B_A e

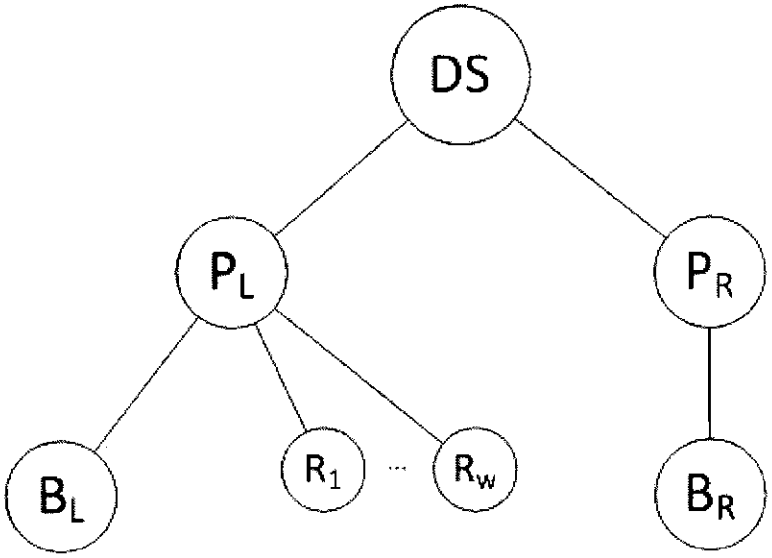

Figura 4.5: Execução local e remota, concorrência entre Brokers - Topologia do cenário

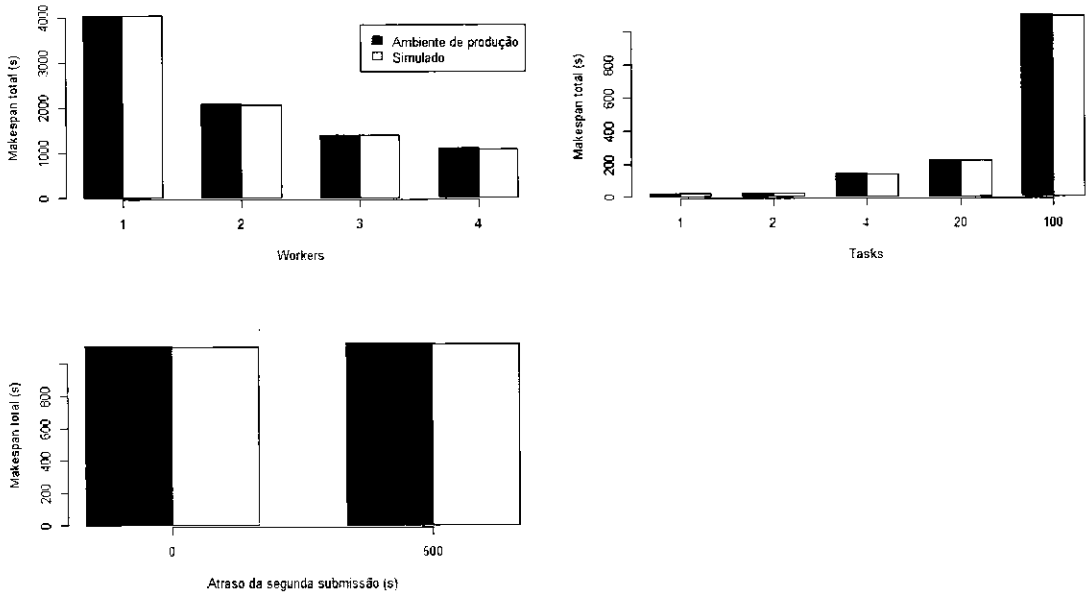

Figura 4.6: Validação externa: Execução local e remota, concorrência entre Brokers

B_B são os Brokers remotos e $R)i$ é um Worker local, tendo $i = 1..W$.

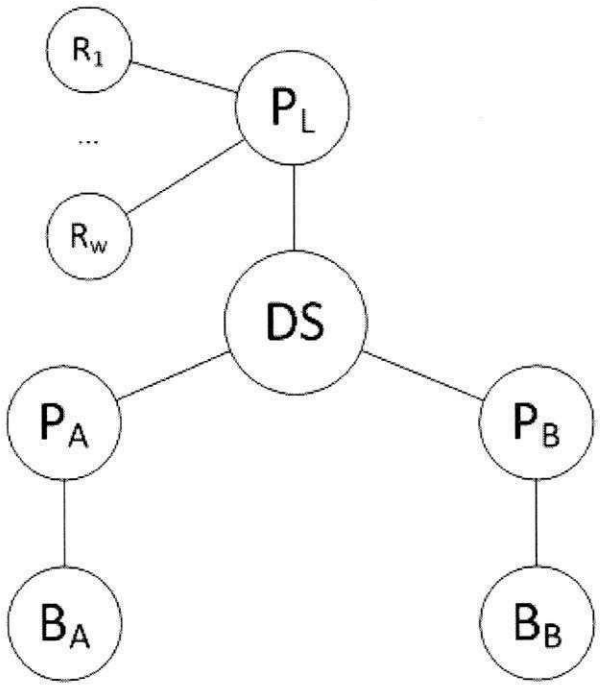

Figura 4.7: Execução remota, concorrência entre Brokers - Topologia do cenário

O Gráfico 4.8 compara a variação do *makespan* para os três conjuntos de experimentos.

Para todos os cenários é possível notar, por análise gráfica, a proximidade entre os valores de *makespan* dos cenários executados em ambiente de produção no OurGrid e dos cenários simulados. Sendo assim, fica claro o atendimento ao requisito funcional de precisão.

4.2 Escalabilidade e velocidade de simulação

Para analisar esta métrica, foram executados sete cenários de simulação, cada um com fator de replicação 5. A quantidade de *sites* foi definida com os valores 1, 5, 10, 50, 100, 500 e 1000. Em cada *site* foram adicionados 50 Workers, e em um deles foi adicionado um Broker, que submetia 50*S tarefas de 10 segundos cada, onde S é a quantidade de sites. A quantidade de *Workers* por *site* reflete a realidade atual da comunidade OurGrid, e a quantidade de tarefas submetidas forçam um cenário de contenção, ou seja, utilização máxima dos recursos. Esse cenários contemplavam modelagem de rede, com um atraso médio de 5ms (LAN), e detecção de falhas, com os valores padrão do OurGrid.

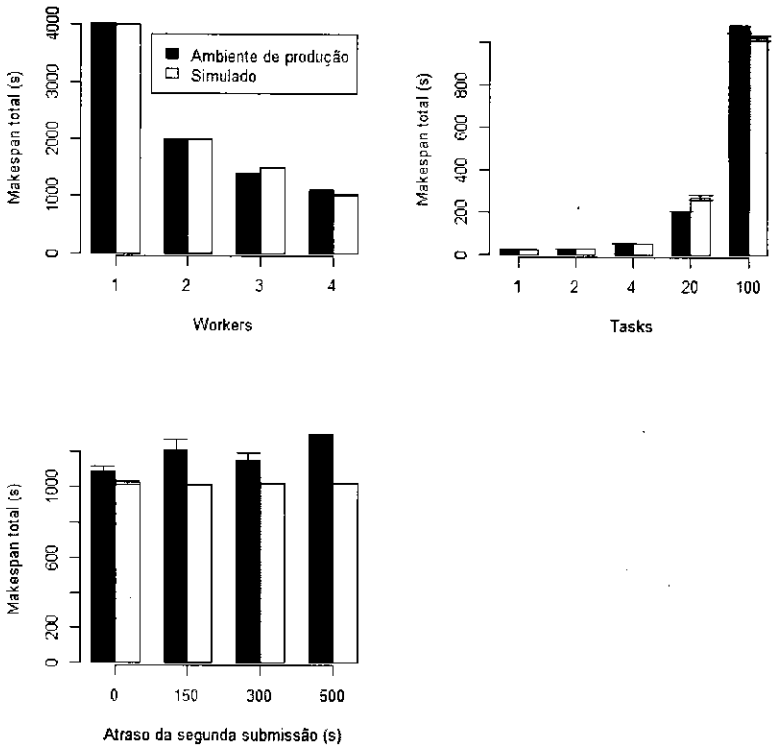

Figura 4.8: Validação externa: Execução remota, concorrência entre Brokers

Essas simulações foram executadas em uma instância extragrande de CPU elevada da Amazon EC2. Esta máquina tem 7 GB de memória e 20 unidades computacionais EC2 (8 núcleos virtuais com 2,5 unidades computacionais EC2 cada). Cada unidade computacional EC2 é equivalente a uma CPU 1.0-1.2 GHz 2007 Opteron.

A Figura 4.9, com os eixos em escala logarítmica e intervalo de confiança em 95%, ilustra os tempos de execução para cada tamanho de cenário. Nota-se que o tempo de execução real aumenta de forma exponencial com o aumento da quantidade de *sites*, o que é explicado pelo aumento da quantidade de eventos de requisição entre *Peers* remotos e pela execução em contenção. A quantidade de eventos processada no maior dos cenários chega à ordem de centenas de milhões.

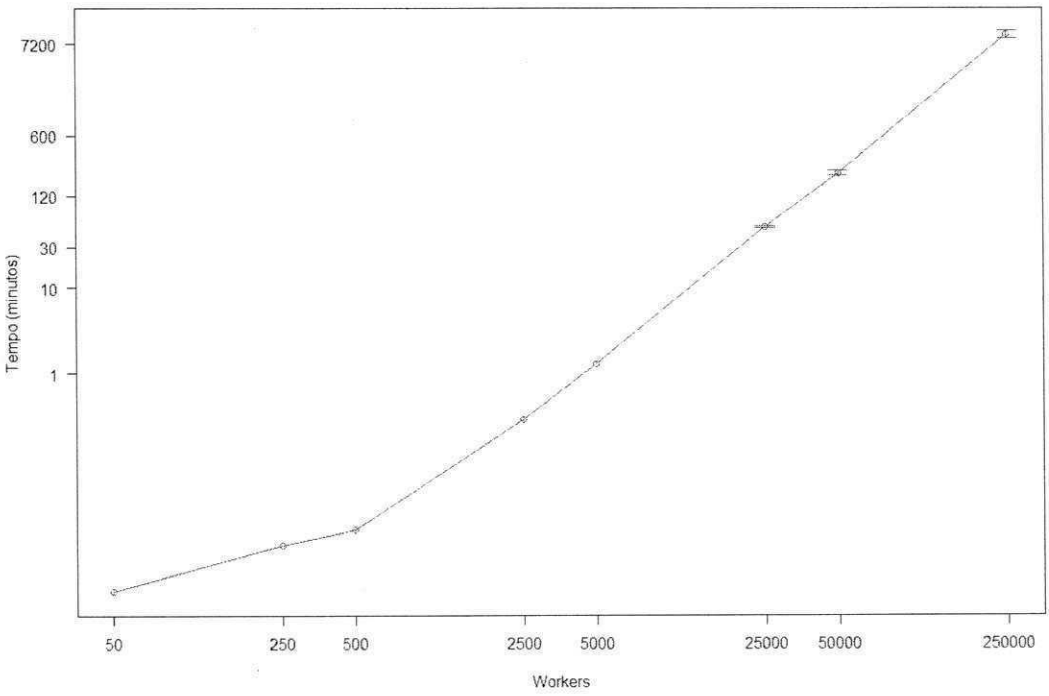

Figura 4.9: Resultados dos testes de escalabilidade

Com esses resultados os requisitos de escalabilidade e velocidade são atendidos, uma vez que o cenário estimado pelo requisito de escala (50000 workers) executou com sucesso e em torno de 3 horas. Além disso, a suposição de que o simulador tem um baixo registro de memória foi confirmada, já que este cenário utilizou, em média, 11% da memória disponível

na máquina.

Além deste conjunto de experimentos, avaliou-se o comportamento da métrica tempo de execução real quando outros fatores, além do número de sites, são variados. Primeiro, o mesmo conjunto de cenário é executado com a detecção de falhas e a camada de rede foram desligados. O Gráfico 4.10 ilustra a evolução do tempo de execução com o aumento do número de *sites*. Nota-se que a adoção da implementação de rede e de detecção de falhas não interfere de forma significativa no tempo de execução dos experimentos em cenários de contenção.

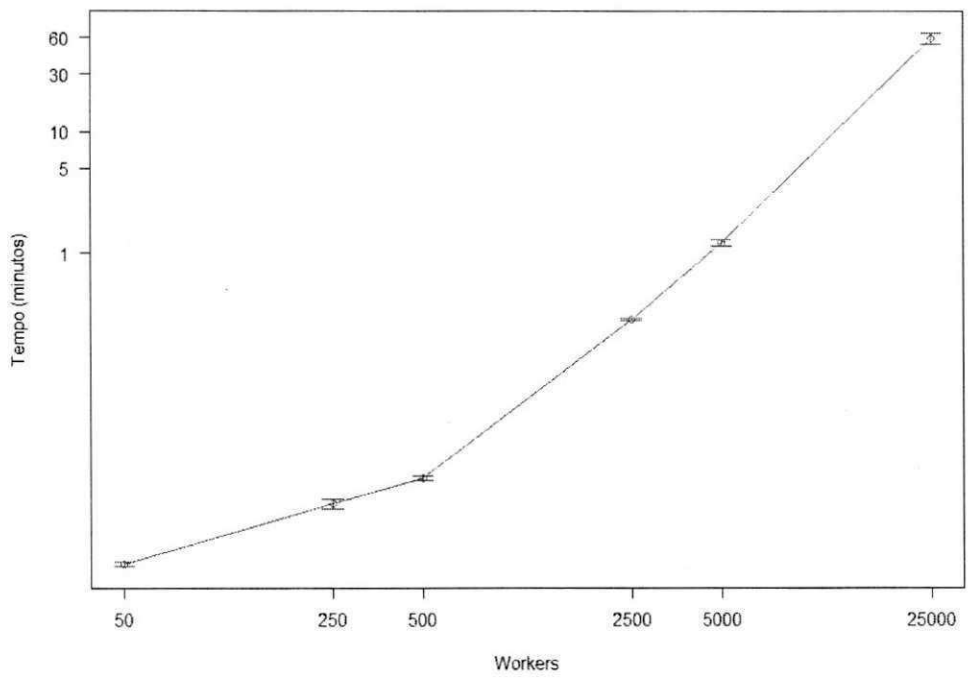

Figura 4.10: Resultados dos testes de escalabilidade: Sem rede e detecção de falhas

Segundo, aumentou-se a quantidade de *sites* enquanto a quantidade de *Workers* foi mantido. Nestes experimentos cada *Peer* tem 25 *Workers*, e, comparado ao primeiro conjunto de experimentos, há o dobro de *Peers* em cada cenário. O Gráfico 4.11 ilustra a evolução do tempo de execução com o aumento do número de *sites*. É possível notar um aumento no tempo de execução médio quando comparado com o primeiro cenário. Esse fato fortalece a hipótese de que a quantidade de *sites* influencia diretamente no tempo de execução dos

experimentos.

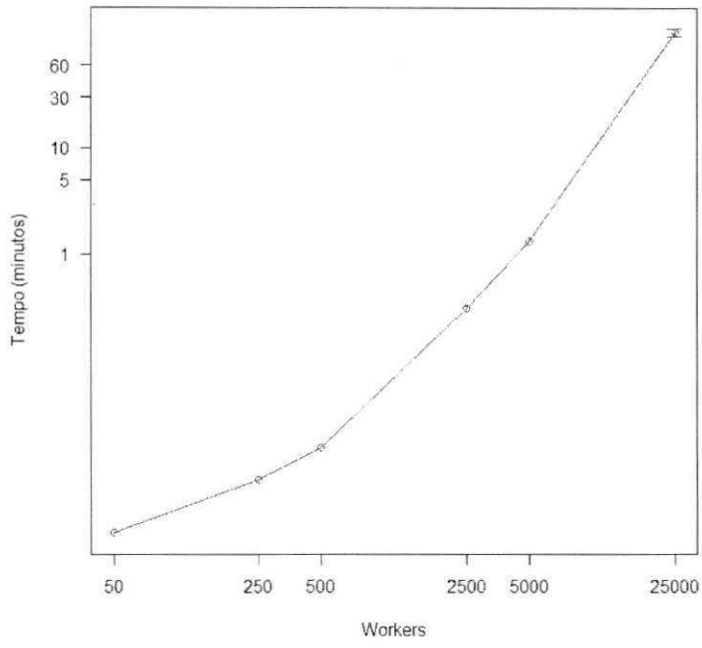

Figura 4.11: Resultados dos testes de escalabilidade: Mais Peers, 25 Workers por site

Por último, variou-se o fator de contenção dos cenários, o que determina a quantidade de tarefas a serem executadas. Fixou-se a quantidade de *sites* em 100, com 50 *Workers* em cada. Foram executados cenários com 1250, 2500, 5000, 10000 e 20000. O Gráfico 4.12, em escala linear, ilustra a evolução do tempo de execução com o aumento do número de tarefas. Nota-se que o tempo de execução cresce praticamente de forma linear com o aumento da quantidade de tarefas.

4.3 Extensibilidade

Nesta seção serão avaliadas as métricas de Engenharia de Software apresentadas na seção 3.1.3, que darão uma indicação sobre o quanto o código é extensível [45], o que compreende a primeira frente de validação de extensibilidade. A segunda frente é contemplada no Capítulo 5, que descreve casos de uso de extensão do simulador.

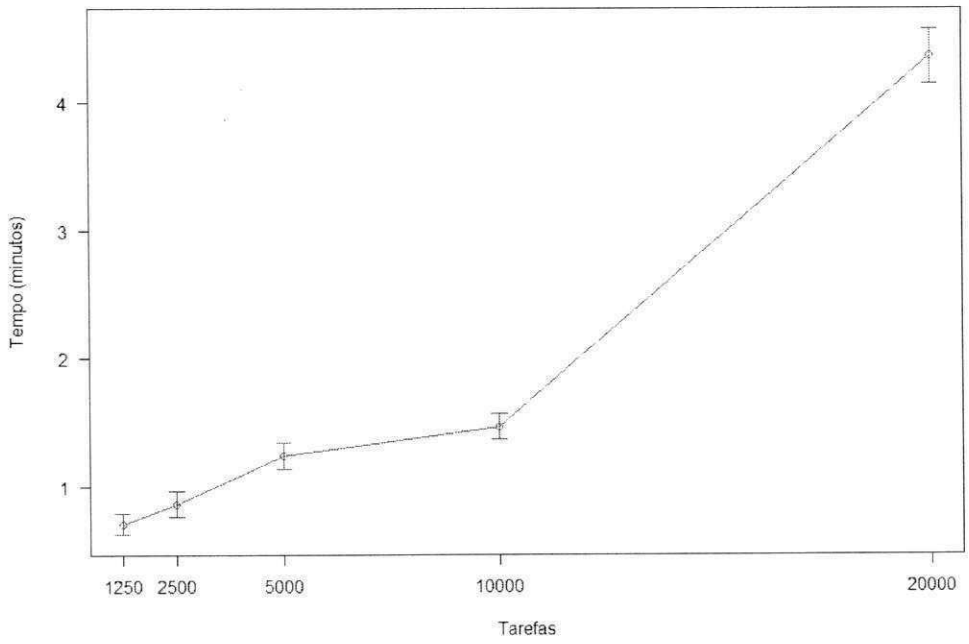

Figura 4.12: Resultados dos testes de escalabilidade: Variação da quantidade de tarefas

O Anexo B apresenta um relatório completo gerado pela ferramenta JDepend das métricas de qualidade para o código deste simulador. Na Tabela 4.1 é apresentado um sumário das métricas em questão, que são detalhadas nas subseções seguintes.

Tabela 4.1: Sumário das métricas de qualidade de código

Métrica	Máximo		Mínimo		**Média**
	Valor	**Pacote**	**Valor**	**Pacote**	
Acoplamento aferente (Aa)	11	entities.grid	1	events.broker	4.1
Acoplamento eferente (Ae)	18	factories	2	entities.accounting	6.23
Abstração (A)	0.8	events	0.1	entities	0.115
Instabilidade (I)	0.93	events.broker	0.17	events	0.63
Distância da sequência principal (DSP)	0.71	entities.accounting	0.03	events	0.27

4.3.1 Acoplamento aferente (Aa) e eferente (Ae)

A métrica acoplamento aferente define, de forma absoluta, quantos pacotes fazem referência ao pacote em questão. Dessa forma, espera-se que pacotes com classes destinadas a reuso tenha um *Aa* alto, quando comparado com a média entre os pacotes. Já o acoplamento eferente indica quantos pacotes o pacote em questão faz referência. Pacotes com classes concretas, ou, no caso deste simulador, implementações de eventos, devem ter um *Ae* alto, quando comparado com a média entre os pacotes. E, de fato, é o que se percebe nas métricas do código fonte deste simulador.

Pacotes como **br.edu.ufcg.lsd.oursim.entities** e **br.edu.ufcg.lsd.oursim.entities.grid**, que possuem componentes a serem reusados, apresentam um acoplamento aferente alto. Já pacotes como **br.edu.ufcg.lsd.oursim.events.broker** e **br.edu.ufcg.lsd.oursim.events.peer**, que possuem implementações concretas de eventos, apresentam um acoplamento eferente alto.

4.3.2 Abstração (A)

Um valor alto da métrica de abstração para os pacotes, que, supostamente devem ter componentes reusáveis, é um bom indicativo de extensibilidade. Esta indica a razão entre classes abstratas e interfaces e a quantidade total de classes em um pacote.

Um exemplo de pacote destinado a reuso é o pacote **br.edu.ufcg.lsd.oursim.events**, que, como esperado, tem um valor de abstração alto de $0,8$.

4.3.3 Instabilidade (I)

Representa a capacidade de um pacote passar por mudanças [45], representado pela razão entre o Acoplamento Eferente e a soma dos Acoplamentos Aferente e Eferente.

Alguns dos pacotes de eventos apresentaram um valor de instabilidade alto, como o pacote **br.edu.ufcg.lsd.oursim.events.broker**, que obteve I=$0,93$. Este valor alto é consequência da decisão de projeto de tornar o evento um elemento independente, que recebe uma referência para a fachada do simulador, tendo acesso a toda a topologia e a fila de eventos; e dinâmico, criado em tempo de execução.

4.3.4 Distância da Sequência Principal (DSP)

Representa o compromisso entre estabilidade e abstração. Quanto mais perto o valor desta métrica estiver de 0 maior é o indicativo de que o pacote em questão é extensível e estável [45].

Em média, os pacotes deste simulador têm um DSP de $0,27$. Um dos pacotes com menor DSP é o **br.edu.ufcg.lsd.oursim.events**, com $0,03$.

4.3.5 Comparação com outros simuladores do OurGrid

Também é interessante comparar os valores obtidos para as métricas supracitadas neste simulador com os valores obtidos por seus predecessores. O OGSim [9] tem uma média de DSP baixa, de 0,18, o que indica alto nível de abstração. Porém alguns módulos, como **entities.util** e **policy**, que deveriam ter uma baixa instabilidade por definição, tem valores acima de 0,7. O SRSSim [13] tem um valor baixo de DSP de 0,15, porém todos os seus pacotes tem instabilidade superior a 0,8, indício de alta sensibilidade a mudanças. O Efficient Grid Simulator [4] tem um valor de DSP de 0,12, porém sua instabilidade média é de 0,86. Pode se concluir que, apesar de apresentarem valores de DSP baixos, mostrando um bom compromisso entre abstração e implementação concreta, os simuladores anteriores falham por

apresentar poucos ganchos de extensão e uma instabilidade média alta quando comparados ao simulador apresentado.

4.4 Testabilidade

Pode-se dizer que um software é testável quando as qualidades apontadas na Seção 3.1.4 são alcançadas. Sendo assim, um componente testável deve ser controlável, observável, isolável e seu teste deve ser passível de automatização.

A implementação do *framework* de testes descrito na Seção 3.3.7, somado à própria arquitetura e projeto do simulador, conseguem contemplar os requisitos levantados anteriormente, uma vez que o testador consegue: i) controlar o andamento da simulação, passo-a-passo e por meio de ganchos quando eventos específicos são processados; ii) visualizar o estado das entidades ativas e executáveis, além de gravar e verificar comportamentos secundários; iii) executar o processamento isolado de eventos específicos, que por si só são unidades lógicas atômicas; e iv) automatizar os testes, já que o desenvolvedor pode, de forma programática, criar a topologia e as especificações de evento, configurar os detalhes da simulação e executá-la.

Capítulo 5

Relatos de extensão

Neste capítulo são analisados três casos de extensão do OurGrid, escolhidos por abordarem níveis de esforço de modelagem distintos. Dois deles são reproduções de trabalhos sobre o OurGrid, um deles relacionado à implementação de novos escalonadores e outro sobre a avaliação de uma nova Rede de Favores, e o terceiro é uma avaliação da adoção de mecanismos adaptativos de detecção de falhas.

Nota-se que, apesar de explorarem diferentes níveis de modelagem, os casos de extensão em questão demandam um esforço de implementação semelhante, que normalmente consiste em generalizar (criação de interfaces) e especializar (criar implementações destas interfaces). Os passos da experiência do programador são relatados e relacionados às técnicas descritas no Capítulo 3. Além disso, os resultados dos trabalhos e suas respectivas reproduções são comparados.

A contribuição geral do relato dos casos de extensão é fortalecer a ideia da utilização do simulador em cenários reais de pesquisa, enaltacendo a sua extensibilidade pra diferentes níveis de abstração e necessidades do pesquisador.

5.1 Metodologia

A observação e o relato de um caso de extensão envolve, primeiramente, a definição do objetivo do desenvolvedor ao estender o código. Em seguida é necessário definir a contribuição que existe para esta pesquisa dado o relato desse caso de extensão. Terceiro, deve-se observar os passos necessários para a implementação da extensão e analisar o quão complexo e o

quanto de esforço de desenvolvimento foi necessário para estender o código. Por fim, relatamos o exercício do simulador extendido implementado pelo desenvolvedor, fazendo uma análise dos resultados obtidos.

5.2 Implementação de novos escalonadores

5.2.1 Objetivo da extensão

O objetivo desse caso é a reprodução do trabalho de Paranhos et al. [57]:*Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids*. Neste trabalho o autor propõe a política WQR (*WorkQueue with Replication*) e a compara com diferentes heurísticas de escalonamento, com o objetivo de mostrar que é possível atingir um bom desempenho utilizando uma abordagem de escalonamento que não utiliza informação do sistema.

Para essa comparação, são escolhidas as políticas: **Dynamic FPLTF** (*Fastest Processor to Largest Task First*), que tenta minimizar o efeito da heterogeneidade dos recursos; a **Sufferage**, que tenta estimar o quanto uma tarefa "sofreria" caso um recurso não fosse alocado para a sua execução. As duas políticas necessitam das informações de tamanho da tarefa, velocidade do recurso e carga do recurso para realizar o escalonamento; e a WorkQueue, que escolhe tarefas de forma arbitrária para serem enviadas aos recursos, sem assumir nada sobre o sistema.

A heurística WQR, proposta neste trabalho, é uma extensão da WorkQueue, uma vez que também escolhe tarefas de forma arbitrária. A principal diferença dessa heurística é a adoção da replicação, que implica na requisição de N recursos para a mesma tarefa. No momento da alocação, o escalonador cria réplicas da tarefa e as executa simultaneamente, até que uma delas termine. Nesse momento as outras réplicas são abortadas e os recursos devolvidos. O objetivo da replicação é lidar com a heterogeneidade da grade e a eventual indisponibilidade dos recursos. A heurística WQR de replicação N é chamada de $WQRN$, sendo assim, a heurística WorkQueue também pode ser chamada de $WQR1$.

Para a comparação com o Dynamic FPLTF e com o Sufferage, foram escolhidos os níveis de replicação 1, 2, 3 e 4 para o WQR.

5.2.2 Contribuição para a pesquisa

Ao observar esta extensão foi possível perceber o potencial da utilização deste modelo na reprodução de trabalhos que modelam o OurGrid em um nível de detalhe igual ou semelhante. Dessa forma, é possível a comparação de resultados e validação do trabalho anterior.

5.2.3 Passos de implementação

O primeiro passo na implementação dos diferentes escalonadores neste modelo de simulação foi abstrair o conceito de heurística de escalonamento. O evento de escalonamento (BrokerEvents.SCHEDULE) em princípio não isola a lógica da heurística WQR, que é implementada por padrão no Broker, assim foi necessário identificar o que era funcionalidade WQR e criar uma interface (SchedulerHeuristic).

Essa interface possui um único método, chamado *schedule*, que recebe como parâmetros o *job* a ser escalonado, a referência para a entidade Broker, a referência para fachada deste simulador e o tempo discreto em que o escalonamento é realizado.

Feito isso, três implementações desta interface foram criadas, cada uma representando uma das heurísticas estudadas. Além disso, para tornar a execução configurável, uma fábrica de heurísticas foi criada, a **SchedulerHeuristicFactory**, que é parametrizada por meio da propriedade BROKER_SCHEDULER_HEURISTIC. Dessa forma, a heurística de escalonamento é criada em tempo de execução.

Resumindo, esta modificação demandou a criação de cinco novas classes relacionadas a heurísticas de escalonamento: SchedulerHeuristic, SchedulerHeuristicDFPLTF, SchedulerHeuristicSufferage, SchedulerHeuristicWQR e SchedulerHeuristicFactory; e demandou a modificação do evento de escalonamento, o ScheduleEvent.

5.2.4 Exercício da extensão

Os experimentos realizados por Paranhos et al. foram baseados em simulação e fizeram uso do SimGrid. O autor fixou o poder computacional da grade em 1.000 e variou o poder computacional de cada Worker seguindo uma distribuição uniforme. 20 diferentes tipos de aplicação de granularidades diferentes foram utilizados, e os tempos de cada tarefa também variam de acordo com uma distribuição uniforme, cuja soma deve ser sempre 3.600.000

segundos.

A principal diferença de projeto entre as simulações de Paranhos et al. e as realizadas neste simulador é a consideração da variação de carga nos Workers. Como os traços usados pelos autores não estão disponíveis, essa variação de carga foi ignorada.

A Figura 5.1 compara os resultados dos tempos médios de execução simulados neste modelo (acima) com os tempos médios de execução do trabalho original de Paranhos et al. (abaixo), para os diferentes grupos de aplicação.

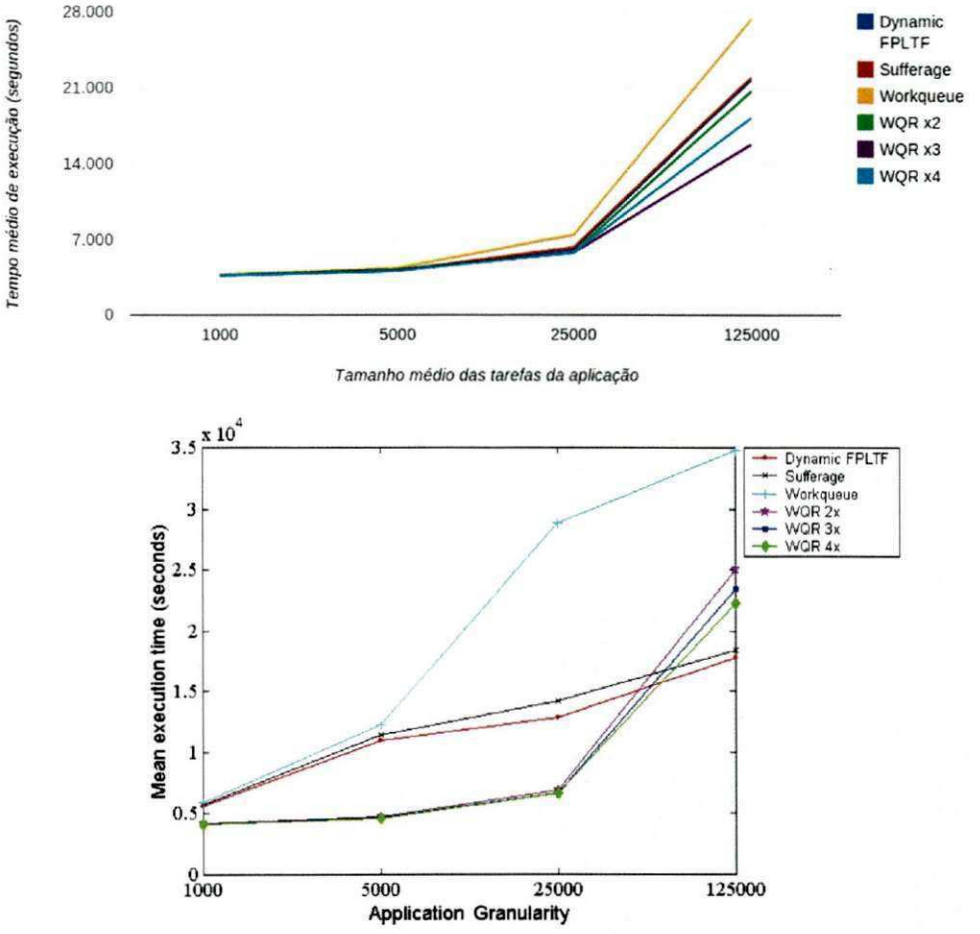

Figura 5.1: Paranhos et al.: Desempenho por tipo de aplicação

Já a Figura 5.2 compara os resultados dos tempos médios de execução simulados neste modelo (acima) com os tempos médios de execução do trabalho original de Paranhos et al.(abaixo), para os diferentes níveis de heterogeneidade das máquinas.

É possível notar um discrepância entre os valores do trabalho de Paranhos et al. e os

valores observados neste trabalho, principalmente no que diz respeito aos valores obtidos pela heurística WorkQueue. Pode-se dizer que a não utilização dos traços de carga descritos por Paranhos et al. teve impacto na diferença dos resultados, uma vez que esta heurística, em particular, tem uma degradação considerável quando é utilizada em um cenário de recursos muito heterogêneos. Inferir a real causa da diferença dos resultados requer uma análise mais detalhada do projeto experimental e dos dois modelos.

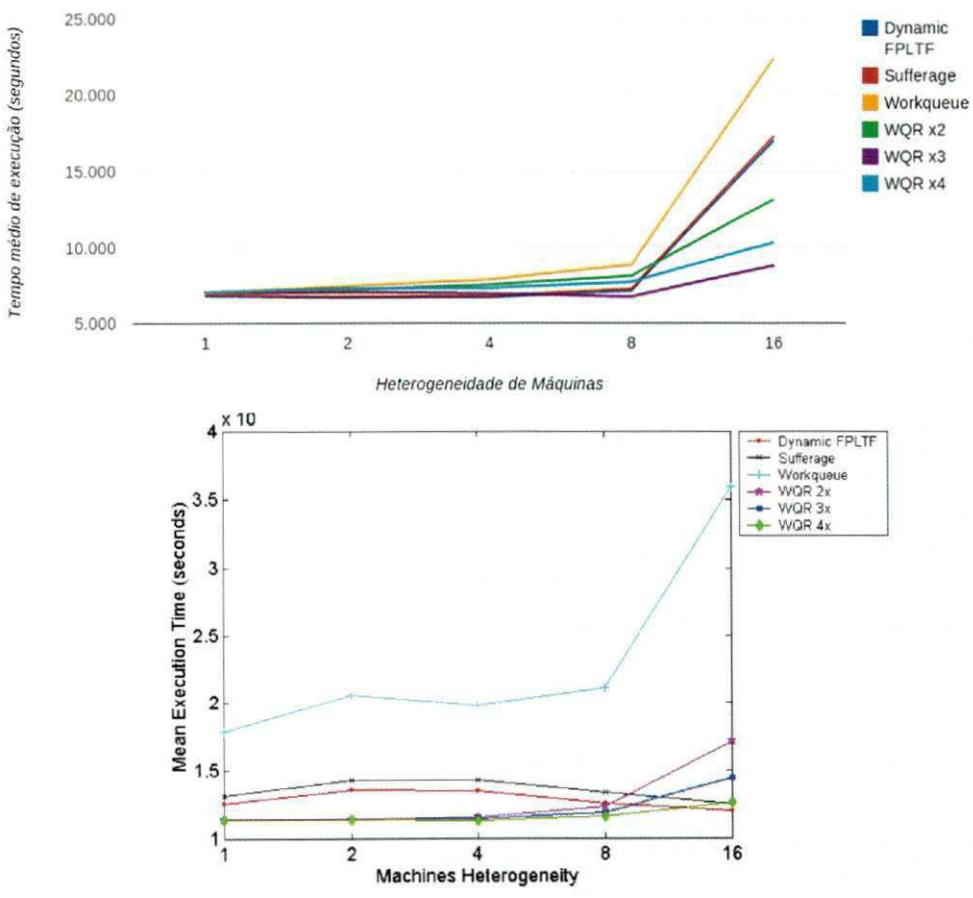

Figura 5.2: Paranhos et al.: Desempenho por nível de heterogeneidade das máquinas

5.3 Implementação de detectores de falha adaptativos

5.3.1 Objetivo da extensão

O objetivo final desta extensão é avaliar o impacto da adoção de diferentes mecanismos de detecção de falha adaptativos no OurGrid, comparado ao detector estático existente no *middleware*. Esta trabalho baseia-se no trabalho de Hayashibara et al. [40], porém tem objetivo de verificar se há impacto no *makespan* de *jobs* e tarefas com a adoção de tais detectores em ambientes de alta latência.

Três detectores adaptativos foram escolhidos para avaliação, propostos por: Chen et al. [28], Bertier et al. [20] e Hayashibara et al. [40].

Chen Chen et al. [28] propuseram um detector de falhas adaptativo que estima os tempos de chegada das mensagens de controle (*heartbeat*) baseando-se em uma janela de amostragem. Nesse mecanismo, sempre que um *heartbeat* é recebido, o tempo de sua chegada é armazenado na janela de amostragem, e o tempo de chegada estimado para o próximo *heartbeat* (*EA*) é atualizado. Essa estimativa é uma média normalizada dos tempos de recepção de *heartbeats* armazenados na janela de amostragem.

Baseado nessa estimativa é calculado o ponto de expiração para o objeto monitorado. O ponto de expiração é dado pela soma da estimativa *EA* com uma margem de segurança α, que é um parâmetro de entrada desse mecanismo. Assim, para determinar se um objeto monitorado falhou, o detector de falhas verifica se o ponto de expiração atual foi ultrapassado.

Bertier O mecanismo de detecção de falhas proposto por Bertier et al. [20] funciona de forma similar ao mecanismo de Chen. A principal diferença está na computação da margem de segurança α, que no mecanismo proposto por Bertier é dinâmica, e é atualizada de acordo com a qualidade da estimativa *EA*.

A computação do α é baseada na estimativa de Jacobson [47], que leva em consideração o erro entre a última estimativa e o tempo real de chegada de um *heartbeat*. Assim, a estimativa do α envolve três parâmetros: γ, que representa a importância da nova estimativa com relação às anteriores; e os parâmetros β e ϕ, que permitem ponderar a estimativa de acordo com a variância do erro.

φ **Accrual** Detectores de falha *accrual* [35,40] descrevem uma abstração diferente: ao invés de entregarem a aplicação um valor booleano (falho/operacional), eles produzem um valor de suspeição numa escala contínua. Assim, esse paradigma permite a separação dos requisitos da aplicação do mecanismo de detecção de falhas.

O detector φ Accrual [40] também armazena *heartbeats* numa janela de amostragem. Essa informação histórica é utilizada para determinar a distribuição dos tempos entre chegadas. Então, essa distribuição é usada para computar o valor corrente de φ, que representa o nível de suspeição para um objeto monitorado.

5.3.2 Contribuição para a pesquisa

Ao observar esta extensão foi possível perceber o potencial da utilização deste modelo na avaliação do impacto da implementação de novas funcionalidades no OurGrid em uma métrica de interesse. Dessa forma, avaliamos os passos de uma extensão não baseada em uma implementação prévia.

5.3.3 Passos de implementação

A implementação desta modificação é bastante semelhante à anterior. Requeriu ainda menos esforço de implementação, uma vez que a interface **FailureDetector** já estava bem definida, como mostrado na Seção 3.3.5.

Assim, foram criadas mais 3 implementações da interface FailureDetector: BertierFailureDetector, ChenFailureDetector e PhiAccrualFailureDetector, além de uma classe abstrata, a AbstractFailureDetector.

Mais dois objetos de modelo foram adicionados para dar suporte às implementações: InterArrivalSamplingWindow e MessageType. De forma análoga à modificação da seção anterior, uma fábrica de detectores, a FailureDetectorFactory, foi criada. Para injetar os detectores nas entidades, as classe FailureDetectorOptInjector e FailureDetectorOptParser foram criadas, e a fábrica JSONGridFactory foi alterada para utilizar este injetor.

Resumindo, 10 classes foram criadas e uma modificada, suprindo as demandas de novas implementações, criação e configuração dinâmica.

5.3.4 Exercício da extensão

Os experimentos deste trabalho têm o objetivo de mostrar se há alguma diferença significante entre o *makespan* de tarefas na utilização dos mecanismos de detecção adaptativos e o mecanismo estático implementado no OurGrid.

O *workload* utilizado nos experimentos foi gerado a partir de traços reais de submissão de tarefas em grades P2P, utilizando o modelo de geração de carga sintética descrito em [25], que descreve o comportamento dos usuários da grade pelo intervalo entre submissões, e as características das aplicações pelo tempo de execução de tarefas e tempo de execução de *jobs*.

Este *workload* representa a execução de *jobs bag-of-task* durante uma semana, em uma comunidade de 50 *peers*, cada *site* com 50 *workers*, 210 usuários submetendo 1860 *jobs* com tamanho médio de 248 *tasks* e desvio padrão de 269 *tasks*. Cada *task* dura em média 1949,03 s, com desvio padrão de 1677,67 s.

Emulou-se uma rede com atraso médio de 500ms, os valores de configuração dos detectores de falha foram definidos de acordo com trabalhos anteriores descritos pelos autores e a configuração do OurGrid segue os valores padrão do *middleware*.

Quatro cenários foram executados, um para cada mecanismo de detecção de falha. O mecanismo de *timeout* foi configurado com tempo de detecção de 300 segundos, que é o valor padrão do OurGrid; o Phi-Accrual foi configurado com *threshold* igual a 0.5; o mecanismo proposto por Chen foi configurado com α=5s; e o detector proposto por Bertier foi configurado com β=1, γ=0.1, ϕ=4 e passo de moderação de 5s. Esses valores foram escolhidos com base nos experimentos descritos em [18,20,40] e nas condições dos cenários desta simulação.

Dado que se pretende verificar se existe diferença no *makespan* de tarefas entre as abordagens adaptativas e o detector de *timeout* estático, foi utilizado um teste-t pareado, com o valor de 0.95 de coeficiente de confiança, para realizar a comparação entre os valores de *makespan* de todas as tarefas.

Com base no teste-t foi possível refutar as hipóteses nulas de que o uso dos detectores adaptativos provocou uma diminuição estatisticamente significativa no *makespan* das tarefas.

Esse resultado se dá principalmente por dois motivos: i) a replicação simultânea implementada no modelo OurGrid amortiza os efeitos da indisponibilidade no *makespan* das

tarefas; e ii) enquanto detectores adaptativos conseguem diminuir o tempo de detecção das falhas, eles também são mais suscetíveis a falsas suspeições, o que pode acarretar o aumento no *makespan* das tarefas

Apesar de não reduzir significamente o *makespan* das tarefas, a adoção dos detectores adaptativos teve um impacto positivo no desperdício de recursos da grade. A Figura 5.3 ilustra o desperdício de recursos em horas para cada um dos mecanismos de detecção. Esse desperdício é calculado como sendo o somatório do tempo de todas as réplicas falhas ou abortadas.

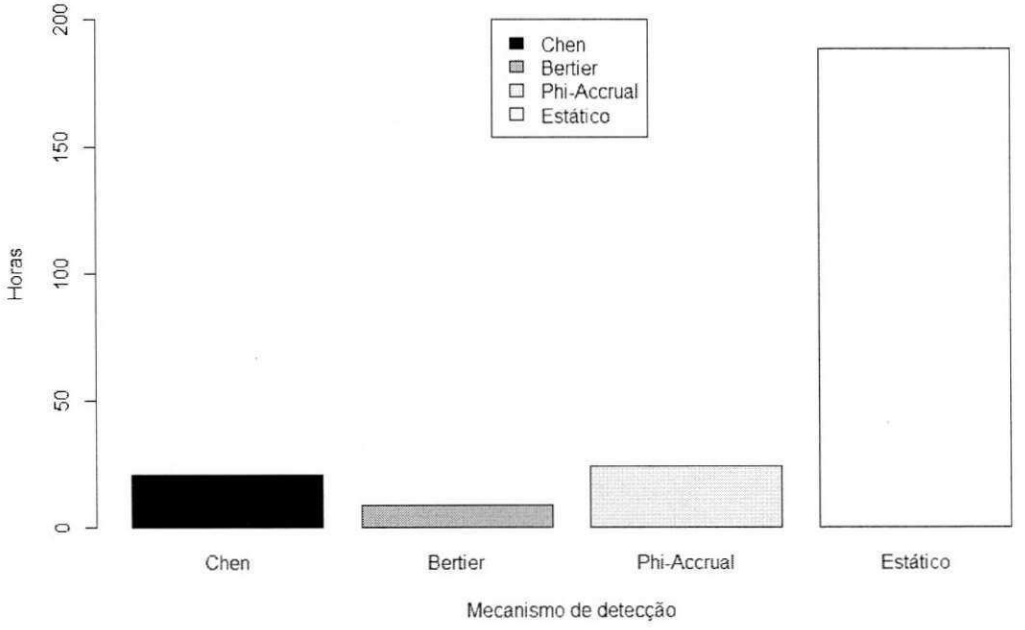

Figura 5.3: Desperdício dos recursos da grade por mecanismo de detecção

5.4 Implementação de uma nova política de rede de favores

5.4.1 Objetivo da extensão

O objetivo desta extensão é a reprodução do trabalho de Gaudêncio e Brasileiro [39]:*Uma Nova Política de Incentivos para Grades Computacionais Entre-Pares*. Neste trabalho os

autores propõem a NNoF, uma nova rede de favores baseada na NoF, política implementada atualmente no OurGrid. Esta nova política se propõe a melhor diferenciar os nós caroneiros (que consomem sem retribuir) dos nós que consomem mais do que doam à grade, com o objetivo de aumentar a quantidade de recursos recebidos pelos nós doadores.

A principal diferença da NNoF em relação à NoF é a manutenção nos Peers dos valores de *doadoPara(P)* e *recebidoDe(P)* para cada outro Peer *P*. Enquanto a NoF representa o valor de crédito com apenas uma variável *conta(P)*.

Quando um nó recebe recursos de um outro nó, ele incrementa o valor de *recebidoDe(P)* com a quantidade de recursos recebidos. Quando um nó doa recursos para outro nó, ele atualiza o valor de *doadoPara(P)* incrementando a quantidade de recursos doados. A variável *conta(P)* ainda existe, mas é calculada a partir desses dois valores.

Para diferenciar os nós simplesmente caroneiros daqueles que consomem mais do que doam, os autores propõem o algoritmo da Figura 5.4. Quando o numero de recursos doados por um consumidor é maior do que o número de recursos que aquele consumidor já recebeu, utiliza-se o saldo $recebidoDe(P) - doadoPara(P)$. Do contrario, se ele já recebeu algum recurso, utiliza-se a taxa $doadoPara(P)/recebidoDe(P)$ para representar a atual conta. Por fim, se o nó nunca recebeu recursos do consumidor, a conta é 0.

if $Recebido\_de(P) \geq Doado\_para(P)$ **then**
  $Conta(P) = Recebido\_de(P) - Doado\_para(P)$
else if $Recebido\_de(P) > 0$ **then**
  $Conta(P) = \frac{Doado\_para(P)}{Recebido\_de(P)}$
else
  $Conta(P) = 0$
end if

Figura 5.4: NNoF: Calculando a conta do nó

5.4.2 Contribuição para a pesquisa

Ao observar esta extensão, nota-se uma contribuição no sentido da adaptação do modelo para reproduzir um trabalho que trata de um nível de detalhe distinto ao que é implementado. Assim, percebe-se que é possível obter resultados da solução aplicada em um modelo mais completo sem grande esforço de implementação.

5.4.3 Passos de implementação

Implementar a NNoF neste novo modelo de simulação foi bastante simples. Os momentos em que os valores de conta de Peer são atualizados, chamados no código por *balances*, ocorrem nos eventos *PeerEvents.FINISH_REQUEST* e *PeerEvents.REPORT_WORK_ACCOUNTING*. Assim, foi necessária a reescrita desses eventos no que diz respeito à atualização dos *balances*, e a adição das estruturas que mantêm os valores de *doadoPara(P)* e *recebidoDe(P)* na entidade Peer.

Dessa forma, nenhuma nova classe foi criada e 3 classes foram alteradas, incluindo a fábrica de topologia, que inicializa as estruturas da NNoF.

5.4.4 Exercício da extensão

Para validar sua solução, Gaudencio e Brasileiro propuseram a avaliação da métrica de satisfação S, dada pela razão entre os recursos recebidos e os recursos requisitados. O autor fez uso de uma modelagem em turnos de trocas de favores, fixando a quantidade de nós em 200 e a quantidade de turnos em 6000. Variou a porcentagem de nós caroneiros em cada cenário em 0%, 25%, 50% e 75%, e definiu três famílias de nós não-caroneiros de acordo com a possibilidade de este nó ser consumidor, chamada de frequência de consumo.

A Figura 5.5 mostra os resultados do cenário com 75% de caroneiros. Na figura cada nó é representado por um tom-de-cinza proporcional ao seu fator de consumo. É notável o aumento na satisfação para os nós doadores, mesmo para aqueles com um alta taxa de consumo. A consequência disso é o escanteamento dos nós caroneiros.

Para reproduzir os cenários de simulações, criou-se uma tupla de Broker e Peer para cada nó simulado. Para cada turno, verificou-se se o Peer é caroneiro, consumidor ou doador, para disparar o comportamento de: i) Derrubar todos os Workers de seu *site* e adicionar um *job* com 10 tarefas de replicação um; ou ii) Levantar todos os Workers de seu *site* e não adicionar nenhuma tarefa. Cada turno tinha a duração da tarefa a ser executada mais um. Nestes cenários, 1000 turnos foram executados.

Foram simuladas duas versões do modelo OurGrid: uma versão simplificada, que tenta refletir o modelo usado no trabalho do autor, e demandou algumas alterações, como a desabilitação da repetição de requisições e redistribuição de *Workers*, e a versão completa do

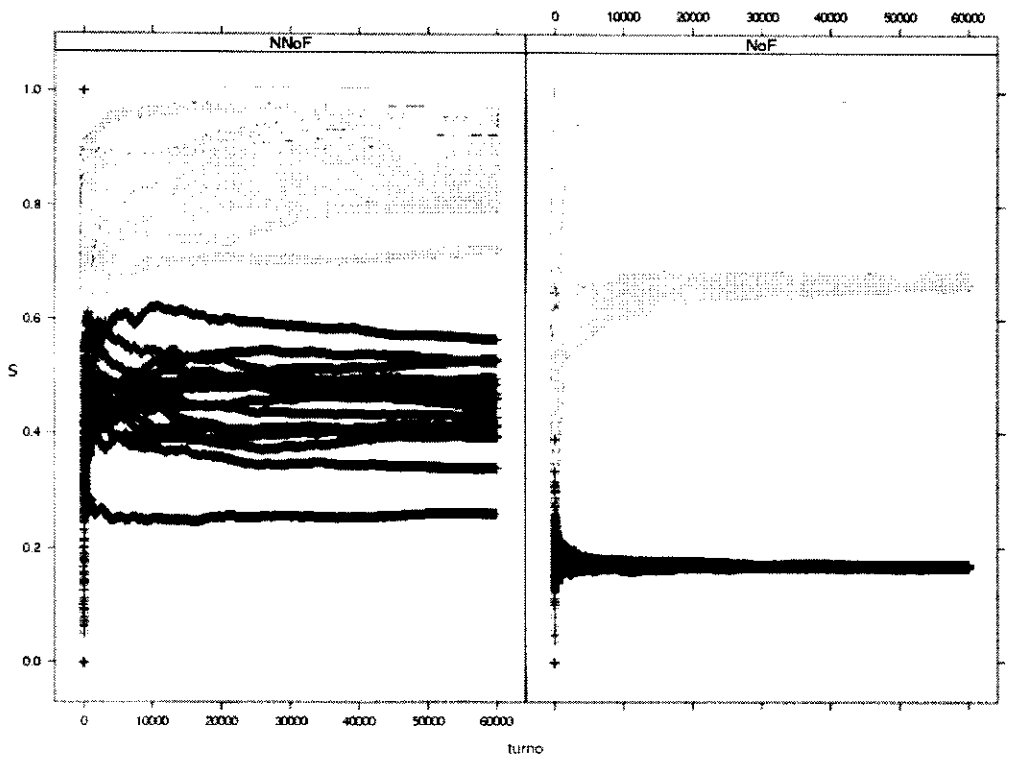

Figura 5.5: Satisfação dos nós doadores ao longo do tempo: NNoF x NoF

modelo.

As Figuras 5.6 e 5.7 mostram os resultados obtidos com o modelo simplificado e com o modelo real, respectivamente. No modelo simplificado, a NNoF consegue distinguir melhor as classes de nós, enquando que no modelo completo, praticamente não há distinção entre as classes, o que deve acontecer pela repetição de requisições e redistribuição de Workers do modelo OurGrid. O impacto da aplicação da NNoF no OurGrid provavelmente seria melhor avaliado com a métrica de *makespan* de tarefas.

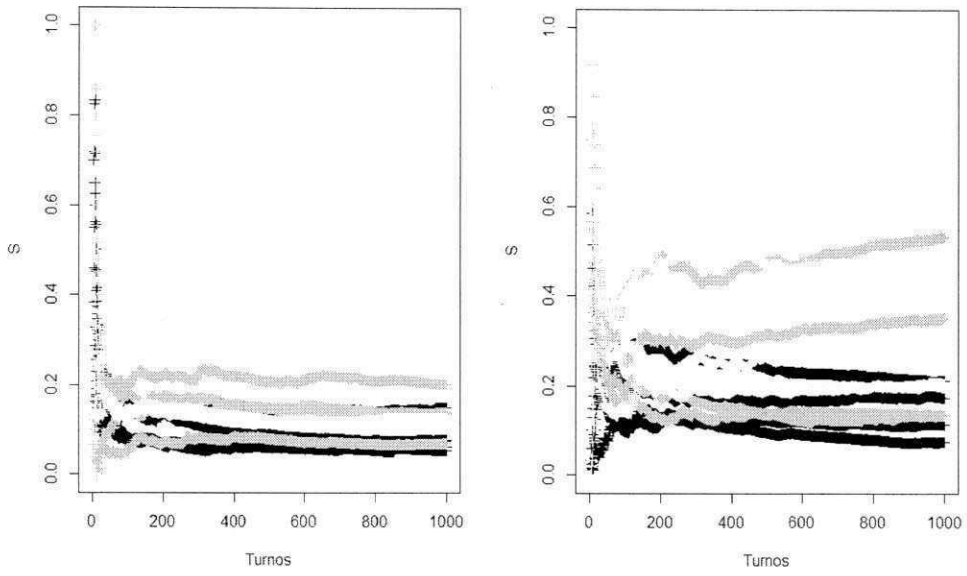

Figura 5.6: Satisfação dos nós doadores ao longo do tempo no modelo simplificado: NoF x NNoF

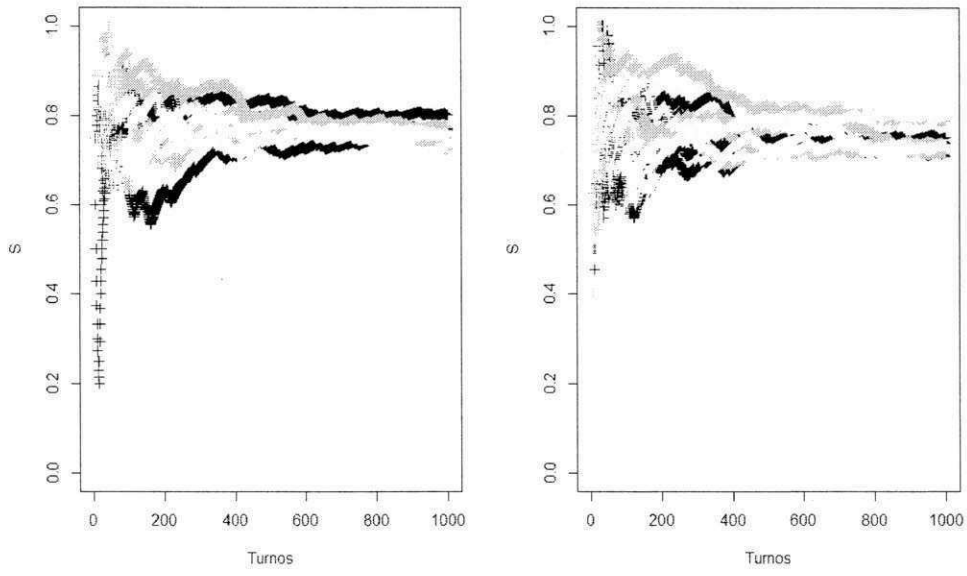

Figura 5.7: Satisfação dos nós doadores ao longo do tempo no modelo completo: NoF x NNoF

Capítulo 6

Conclusões e Trabalhos Futuros

Este capítulo contém considerações, apresentadas na Seção 6.1, acerca dos resultados descritos ao longo desta dissertação, além de desdobramentos destes resultados em termos de trabalhos futuros que são discutidos na Seção 6.2.

6.1 Conclusões

Nesta dissertação mostrou-se que é possível criar um modelo de simulação para uma grade computacional P2P, o OurGrid, respeitando requisitos de completude, precisão, escala, velocidade, extensibilidade e testabilidade. O principal objetivo desse modelo é viabilizar seu reuso entre os trabalhos de pesquisa que tratem deste *middleware*, contribuindo para o amadurecimento de uma cultura e metodologia de pesquisa que eleve o empacotamento dos trabalhos e a reprodutibilidade dos resultados a um novo patamar.

Como prova de conceito, foi implementando um simulador discreto baseado em eventos para contemplar este modelo. Seu código está disponível na página de acompanhamento do projeto OurSim no GitHub [15].

Partindo de técnicas de implementação e de validação vindas da Engenharia de Software, foi possível atender e validar os requisitos citados, dadas as métricas e valores determinados em suas especificações. Além disso, foram mostrados casos de uso que exemplificam a facilidade de adaptação do simulador, além de resultados reproduzidos por meio deste modelo.

6.2 Trabalhos Futuros

6.2.1 Extensão de outros aspectos do modelo

Apesar de se mostrar como um modelo completo em nível de aplicação, o modelo apresentado neste trabalho ainda carece de certas funcionalidades relacionadas ao ambiente simulado. Por exemplo, é necessário avaliar qual o impacto da implementação de uma rede sensível ao contexto, ou seja, que representa de forma adaptativa a conexão modelada. Além disso, deve ser possível modelar a rede de forma mais detalhada, contemplando uma pilha TCP, por exemplo.

É óbvio que estes adendos ao modelo não devem deixar de lado a flexibilidade do modelo. Toda nova funcionalidade deve ser facilmente desligável e configurável, mantendo o compromisso assumido entre escala e velocidade.

6.2.2 Reprodução de outros trabalhos e comparação dos resultados

A avaliação deste modelo deve ser contínua e iterativa. A reprodução de outros trabalhos sobre o OurGrid, além de enriquecer e atualizar os dados de pesquisa sobre o *middleware*, propiciam o amadurecimento do modelo no sentido da adaptação para diferentes contextos de pesquisa.

6.2.3 Criação de um esquema de empacotamento de resultados

A ideia de criar um esquema padrão de empacotamento de resultados, documentação precisa dos passos de pesquisa, e reprodução fácil do ambiente de desenvolvimento do pesquisador, apesar de não ser nova na Ciência de um modo geral, engatinha quando se fala em Ciência da Computação.

Ter padrões de ferramenta durante a pesquisa, além de diminuir os riscos de validação do trabalho, é um primeiro passo na busca pela reprodutibilidade. Assim, arquivos de entrada e saída usados em uma pesquisa, quando publicados, podem ser reusados sem maiores esforços de tradução.

A criação de um repositório padronizado de trabalhos de pesquisa no OurGrid, no qual os pesquisadores possam documentar todos os seus passos, armazenar arquivos de entrada e

saída de simulações, guardar *patches* de mudanças do simulador, etc., seria de grande valia para a mudança cultural metodológica levantada neste trabalho.

Bibliografia

[1] Apache commons. http://commons.apache.org/, 2011.

[2] Clover. http://www.atlassian.com/software/clover/overview, 2011.

[3] douglascrockford / json-java - github. https://github.com/douglascrockford/JSON-java, 2011.

[4] Efficient grid simulator. https://svn.lsd.ufcg.edu.br/repos/ourgrid/projects/efficient-grid-simulator/, 2011.

[5] Jdepend. http://clarkware.com/software/JDepend.html, 2011.

[6] Junit.org resources for test driven development. http://www.junit.org/, 2011.

[7] Minimum acceptable code coverage. http://www.bullseye.com/minimum.html, 2011.

[8] The network simulator (ns2). http://isi.edu/nsnam/ns/, 2011.

[9] Ogsim - an opportunistic peer-to-peer desktop grid simulator. http://redmine.lsd.ufcg.edu.br/projects/ogsim/, 2011.

[10] Oracle technology network for java developers. http://www.oracle.com/technetwork/java/index.html, 2011.

[11] Ourgrid acceptance tests. http://redmine.lsd.ufcg.edu.br/projects/ourgrid/wiki/Development-acceptancetests, 2011.

[12] Peersim: A peer-to-peer simulator. http://peersim.sourceforge.net/, 2011.

[13] Srssim. https://svn.lsd.ufcg.edu.br/repos/ourgrid/projects/srssim/, 2011.

[14] Ssj: Simulation stochastique en java. http://www.iro.umontreal.ca/ simardr/ssj/, 2011.

[15] abmargb / oursim-ext - github. https://github.com/abmargb/oursim-ext, 2012.

[16] Osman Balci. Principles and techniques of simulation validation, verification, and testing. In *Proceedings of the 27th conference on Winter simulation*, WSC '95, pages 147–154, Washington, DC, USA, 1995. IEEE Computer Society.

[17] Jerry Banks. *Principles of Simulation*, pages 1–30. John Wiley Sons, Inc., 2007.

[18] Abmar Barros. Zookeeper failure detector model. http://wiki.apache.org/hadoop/ZooKeeper/GSoCFailureDetector, 2010.

[19] William H. Bell, David G. Cameron, Luigi Capozza, A. Paul Millar, Kurt Stockinger, and Floriano Zini. Optorsim - a grid simulator for studying dynamic data replication strategies. *International Journal of High Performance Computing Applications*, page 2003, 2003.

[20] M. Bertier, O. Marin, and P. Sens. Implementation and performance evaluation of an adaptable failure detector. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 354 – 363, 2002.

[21] Robert V. Binder. *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[22] F. Brasileiro. Laboratórios do mundo uni-vos!!! Seminários da Física, 2009.

[23] Rajkumar Buyya and Manzur Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE (CCPE*, 14(13):1175–1220, 2002.

[24] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, and O. Richard. Grid'5000: A large scale and highly reconfigurable grid experimental testbed. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, GRID '05, pages 99–106, Washington, DC, USA, 2005. IEEE Computer Society.

[25] Marcus Williams Aquino Carvalho. Predição da Qualidade de Serviço em Grades Computacionais P2P. Master's thesis, Universidade Federal de Campina Grande, Campina Grande, Paraíba, Brasil, 2010.

[26] Henri Casanova, Arnaud Legrand, and Martin Quinson. Simgrid: A generic framework for large-scale distributed experiments. In *Proceedings of the Tenth International Conference on Computer Modeling and Simulation*, pages 126–131, Washington, DC, USA, 2008. IEEE Computer Society.

[27] Henri Casanova, Arnaud Legrand, and Martin Quinson. Simgrid: A generic framework for large-scale distributed experiments. In *Proceedings of the Tenth International Conference on Computer Modeling and Simulation*, pages 126–131, Washington, DC, USA, 2008. IEEE Computer Society.

[28] W. Chen, S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. In *International Conference on Dependable Systems and Networks (DSN'2000)*, pages 191–200, New York, USA, Jun 2000. IEEE Computer Society.

[29] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33:3–12, July 2003.

[30] Walfredo Cirne, Francisco Brasileiro, Nazareno Andrade, Lauro Costa, Alisson Andrade, Reynaldo Novaes, and Miranda Mowbray. Labs of the World, Unite!!! *Journal of Grid Computing*, 4(3):225–246, 2006.

[31] Walfredo Cirne, Francisco Brasileiro, Daniel Paranhos, Luís Fabrício W. Góes, and William Voorsluys. On the efficacy, efficiency and emergent behavior of task replication in large distributed systems. *Parallel Computing*, 33(3):213 – 234, 2007.

[32] Walfredo Cirne, Francisco Brasileiro, Jacques Sauvé, Nazareno Andrade, Daniel Paranhos, Elizeu Santos-neto, Raissa Medeiros, and Federal Campina Gr. Grid computing for bag of tasks applications. In *In Proc. of the 3rd IFIP Conference on E-Commerce, E-Business and EGovernment*, 2003.

[33] James H. Cowie, David M. Nicol, and Andy T. Ogielski. Modeling the global internet. *Computing in Science and Engg.*, 1:42–50, January 1999.

[34] Leonardo de Assis, Nelson Nóbrega-Júnior, Francisco Brasileiro, and Walfredo Cirne. Uma heurística de particionamento de carga divisível para grids computacionais. In *Anais do 24o Simpósio Brasileiro de Redes de Computadores, 2006*, 2006.

[35] X. Défago, P. Urbán, N. Hayashibara, and T. Katayama. Definition and specification of accrual failure detectors. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'2005)*, pages 206–215, Yokohama, Japan, June 2005. IEEE Computer Society.

[36] XMPP Standard Foundations. About - xmpp standard foundations. http://xmpp.org/about-xmpp/, 2011.

[37] Kayo Fujiwara and Henri Casanova. Speed and accuracy of network simulation in the simgrid framework. In *Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, ValueTools '07, pages 12:1–12:10, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[38] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[39] Matheus Gaudêncio and Francisco Brasileiro. Uma nova política de incentivos para grades computacionais entre-pares. In *Anais do 29o Simpósio Brasileiro de Redes de Computadores, 2011*, 2011.

[40] N. Hayashibara, X. Défago, R. Yared, and T. Katayama. The φ accrual failure detector. In *Proceedings of the 23rd IEEE Symposium on Reliable Distributed Systems*, pages 66–78, Florianópolis, Brazil, September 2004. IEEE Computer Society.

[41] D. Hughes, G. Coulson, and J. Walkerdine. Free riding on gnutella revisited: the bell tolls? *Distributed Systems Online, IEEE*, 6(6), june 2005.

[42] Jeffrey A. Joines and Stephen D. Roberts. *Object-Oriented Simulation*, pages 395–427. John Wiley Sons, Inc., 2007.

[43] Pedro García López, Carles Pairot, Rubén Mondéjar, Jordi Pujol Ahulló, Helio Tejedor, and Robert Rallo. Planetsim: A new overlay network simulation framework. In Thomas Gschwind and Cecilia Mascolo, editors, *SEM*, volume 3437 of *Lecture Notes in Computer Science*, pages 123–136. Springer, 2004.

[44] Tim Mackinnon, Steve Freeman, and Philip Craig. Endo-testing : Unit testing with mock objects. *Extreme programming examined*, pages 287–301, 2001.

[45] Robert C. Martin. Object oriented design quality metrics: An analysis of dependencies. *ROAD*, 2, Sep-Oct 1995.

[46] Robert C Martin. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002.

[47] V. Paxson. Computing tcp's retransmission timer. http://www.faqs.org/rfcs/rfc2988.html, 2000.

[48] John R. Platt. Strong Inference. *Science*, 146(3642):347–353, October 1964.

[49] K. R. Popper. *The Logic of Scientific Discovery*. Hutchinson, London, 1934.

[50] A. Alan B. Pritsker. *Principles of Simulation Modeling*, pages 31–51. John Wiley Sons, Inc., 2007.

[51] M. Quinson. Ultra scalable simulation with simgrid, January 2012.

[52] Kavitha Ranganathan and Ian Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, HPDC '02, pages 352–, Washington, DC, USA, 2002. IEEE Computer Society.

[53] George F. Riley. The georgia tech network simulator. In *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, MoMeTools '03, pages 5–12, New York, NY, USA, 2003. ACM.

[54] M Ripeanu and I Foster. Mapping the gnutella network: Macroscopic properties of large-scale peer-to-peer systems. *PeerToPeer Systems First International Workshop IPTPS 2002 Cambridge MA USA March 78 2002 Revised Papers*, 2429:85–93, 2002.

[55] César A. F. De Rose, Tiago Ferreto, Rodrigo N. Calheiros, Walfredo Cirne, Lauro B. Costa, and Daniel Fireman. Allocation strategies for utilization of space-shared resources in bag of tasks grids. *Future Generation Computer Systems*, 24(5):331–341, 2008.

[56] Robson Santos, Alisson Andrade, Walfredo Cirne, Francisco Brasileiro, and Nazareno Andrade. Relative autonomous accounting for peer-to-peer grids: Research articles. *Concurr. Comput. : Pract. Exper.*, 19(14):1937–1954, 2007.

[57] Daniel Paranhos Da Silva, Walfredo Cirne, Francisco Vilar Brasileiro, and Campina Grande. Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. In *Applications on Computational Grids, in Proc of Euro-Par 2003*, pages 169–180, 2003.

[58] J.A. Smith and S.K. Shrivastava. A system for fault-tolerant execution of data and compute intensive programs over a network of workstations, 1996.

[59] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostić, Jeff Chase, and David Becker. Scalability and accuracy in a large-scale network emulator. *SIGOPS Oper. Syst. Rev.*, 36:271–284, December 2002.

[60] Rodrigo Vilar. Commune - a communication platform for asynchronous distributed objects. http://redmine.lsd.ufcg.edu.br/projects/commune/, 2010.

[61] Huaxia Xia, Holly Dail, Henri Casanova, and Andrew A. Chien. The microgrid: Using online simulation to predict application performance in diverse grid network environments. In *Proceedings of the 2nd International Workshop on Challenges of Large Applications in Distributed Environments*, pages 52–, Washington, DC, USA, 2004. IEEE Computer Society.

Apêndice A

Relatório de cobertura

Clover Coverage Report -
Coverage timestamp: Qui Nov 17 2011 17:38:19 GMT-03:00

App Test Results Clouds
Overview Package File
FRAMES NO FRAMES SHOW HELP

Statistics for project Clover database Qui Nov 17 2011 17:37:25 GMT-03:00:

Stmts:	2.064	**LOC:**	7.962	**Total cmp:**	921
Branches:	562	**NCLOC:**	5.464	**Cmp density:**	0,45
Methods:	581	**Files:**	138	**Avg method cmp:**	1,59
Classes:	143	**Packages:**	23		

Stmts/Method:	3,55
Methods/Class:	4,06
Classes/Pkg:	6,22

This report was generated with an evaluation server license. Purchase Clover or configure your license.

	Packages	Average Method Complexity	TOTAL Coverage
Clover database Qui Nov 17 2011 17:37:25 GMT-03:00	23	1,59	94,1%

Package	Files	Average Method Complexity	TOTAL Coverage
br.edu.ufcg.lsd.oursim.entities.allocation	2	1,12	74,6%
br.edu.ufcg.lsd.oursim.queue	4	1,78	86,5%
br.edu.ufcg.lsd.oursim.util	3	2	89,3%
br.edu.ufcg.lsd.oursim.fd	12	1,66	90%
br.edu.ufcg.lsd.oursim.events.peer.allocation	4	2,92	90%
br.edu.ufcg.lsd.oursim	2	1,38	93,5%
br.edu.ufcg.lsd.oursim.events.fd	9	2,14	93,7%
br.edu.ufcg.lsd.oursim.entities.accounting	2	1	94,3%
br.edu.ufcg.lsd.oursim.entities	4	1,11	95,4%
br.edu.ufcg.lsd.oursim.factories	3	2,18	95,5%
br.edu.ufcg.lsd.oursim.trace	2	1,17	95,7%
br.edu.ufcg.lsd.oursim.events.worker	11	1,65	96,3%
br.edu.ufcg.lsd.oursim.events.ds	4	1,29	96,6%
br.edu.ufcg.lsd.oursim.events.broker	15	2,3	97%
br.edu.ufcg.lsd.oursim.acceptance	5	1,24	98%
br.edu.ufcg.lsd.oursim.events.peer	29	1,43	98,1%
br.edu.ufcg.lsd.oursim.events.peer.accounting	5	1,82	98,2%
br.edu.ufcg.lsd.oursim.entities.grid	5	1,05	100%
br.edu.ufcg.lsd.oursim.entities.job	4	1	100%
br.edu.ufcg.lsd.oursim.entities.request	3	1	100%
br.edu.ufcg.lsd.oursim.events	5	1,08	100%
br.edu.ufcg.lsd.oursim.events.global	2	1,25	100%
br.edu.ufcg.lsd.oursim.network	3	1	100%

Report generated by Clover Code Coverage v3.1.0 *Qui Nov 17 2011 17:38:20 GMT-03:00.*

Clover: Evaluation License registered to Abmar Barros. You have 12 day(s) before your license expires.

Apêndice B

Relatório de métricas de qualidade

```
------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim
------------------------------------------------

Stats:
    Total Classes: 2
    Concrete Classes: 2
    Abstract Classes: 0

    Ca: 10
    Ce: 10

    A: 0
    I: 0,5
    D: 0,5

Abstract Classes:

Concrete Classes:
    br.edu.ufcg.lsd.oursim.Main
    br.edu.ufcg.lsd.oursim.OurSim

Depends Upon:
    br.edu.ufcg.lsd.oursim.entities.grid
    br.edu.ufcg.lsd.oursim.events
    br.edu.ufcg.lsd.oursim.factories
    br.edu.ufcg.lsd.oursim.network
    br.edu.ufcg.lsd.oursim.queue
    br.edu.ufcg.lsd.oursim.trace
    br.edu.ufcg.lsd.oursim.util
    java.io
    java.lang
```

```
    java.util

Used By:
    br.edu.ufcg.lsd.oursim.events
    br.edu.ufcg.lsd.oursim.events.broker
    br.edu.ufcg.lsd.oursim.events.ds
    br.edu.ufcg.lsd.oursim.events.fd
    br.edu.ufcg.lsd.oursim.events.global
    br.edu.ufcg.lsd.oursim.events.peer
    br.edu.ufcg.lsd.oursim.events.peer.allocation
    br.edu.ufcg.lsd.oursim.events.worker
    br.edu.ufcg.lsd.oursim.experiments
    br.edu.ufcg.lsd.oursim.queue

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.entities
-------------------------------------------------

Stats:
    Total Classes: 4
    Concrete Classes: 4
    Abstract Classes: 0

    Ca: 9
    Ce: 4

    A: 0
    I: 0,31
    D: 0,69

Abstract Classes:

Concrete Classes:
    br.edu.ufcg.lsd.oursim.entities.ActiveEntity
    br.edu.ufcg.lsd.oursim.entities.Entity
    br.edu.ufcg.lsd.oursim.entities.ExecutableEntity
    br.edu.ufcg.lsd.oursim.entities.Monitor

Depends Upon:
    br.edu.ufcg.lsd.oursim.entities.job
    br.edu.ufcg.lsd.oursim.fd
    java.lang
    java.util

Used By:
    br.edu.ufcg.lsd.oursim.entities.grid
    br.edu.ufcg.lsd.oursim.entities.job
```

```
    br.edu.ufcg.lsd.oursim.entity
    br.edu.ufcg.lsd.oursim.events.broker
    br.edu.ufcg.lsd.oursim.events.fd
    br.edu.ufcg.lsd.oursim.events.peer
    br.edu.ufcg.lsd.oursim.experiments
    br.edu.ufcg.lsd.oursim.factories
    br.edu.ufcg.lsd.oursim.fd

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.entities.accounting
-------------------------------------------------

Stats:
    Total Classes: 2
    Concrete Classes: 2
    Abstract Classes: 0

    Ca: 5
    Ce: 2

    A: 0
    I: 0,29
    D: 0,71

Abstract Classes:

Concrete Classes:
    br.edu.ufcg.lsd.oursim.entities.accounting.ReplicaAccounting
    br.edu.ufcg.lsd.oursim.entities.accounting.WorkAccounting

Depends Upon:
    br.edu.ufcg.lsd.oursim.entities.job
    java.lang

Used By:
    br.edu.ufcg.lsd.oursim.entities.grid
    br.edu.ufcg.lsd.oursim.events.broker
    br.edu.ufcg.lsd.oursim.events.peer
    br.edu.ufcg.lsd.oursim.events.peer.accounting
    br.edu.ufcg.lsd.oursim.events.worker

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.entities.allocation
-------------------------------------------------

Stats:
    Total Classes: 2
```

```
    Concrete Classes: 2
    Abstract Classes: 0

    Ca: 3
    Ce: 3

    A: 0
    I: 0,5
    D: 0,5

Abstract Classes:

Concrete Classes:
    br.edu.ufcg.lsd.oursim.entities.allocation.Allocation
    br.edu.ufcg.lsd.oursim.entities.allocation.AllocationInfo

Depends Upon:
    br.edu.ufcg.lsd.oursim.entities.request
    java.lang
    java.util

Used By:
    br.edu.ufcg.lsd.oursim.entities.grid
    br.edu.ufcg.lsd.oursim.events.peer
    br.edu.ufcg.lsd.oursim.events.peer.allocation

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.entities.grid
-------------------------------------------------

Stats:
    Total Classes: 5
    Concrete Classes: 5
    Abstract Classes: 0

    Ca: 11
    Ce: 8

    A: 0
    I: 0,42
    D: 0,58

Abstract Classes:

Concrete Classes:
    br.edu.ufcg.lsd.oursim.entities.grid.Broker
    br.edu.ufcg.lsd.oursim.entities.grid.DiscoveryService
```

```
    br.edu.ufcg.lsd.oursim.entities.grid.Grid
    br.edu.ufcg.lsd.oursim.entities.grid.Peer
    br.edu.ufcg.lsd.oursim.entities.grid.Worker

Depends Upon:
    br.edu.ufcg.lsd.oursim.entities
    br.edu.ufcg.lsd.oursim.entities.accounting
    br.edu.ufcg.lsd.oursim.entities.allocation
    br.edu.ufcg.lsd.oursim.entities.job
    br.edu.ufcg.lsd.oursim.entities.request
    br.edu.ufcg.lsd.oursim.events.peer
    java.lang
    java.util

Used By:
    br.edu.ufcg.lsd.oursim
    br.edu.ufcg.lsd.oursim.events.broker
    br.edu.ufcg.lsd.oursim.events.ds
    br.edu.ufcg.lsd.oursim.events.fd
    br.edu.ufcg.lsd.oursim.events.peer
    br.edu.ufcg.lsd.oursim.events.peer.accounting
    br.edu.ufcg.lsd.oursim.events.peer.allocation
    br.edu.ufcg.lsd.oursim.events.worker
    br.edu.ufcg.lsd.oursim.experiments
    br.edu.ufcg.lsd.oursim.factories
    br.edu.ufcg.lsd.oursim.fd

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.entities.job
-------------------------------------------------

Stats:
    Total Classes: 4
    Concrete Classes: 4
    Abstract Classes: 0

    Ca: 9
    Ce: 4

    A: 0
    I: 0,31
    D: 0,69

Abstract Classes:

Concrete Classes:
    br.edu.ufcg.lsd.oursim.entities.job.ExecutionState
```

```
    br.edu.ufcg.lsd.oursim.entities.job.Job
    br.edu.ufcg.lsd.oursim.entities.job.Replica
    br.edu.ufcg.lsd.oursim.entities.job.Task

Depends Upon:
    br.edu.ufcg.lsd.oursim.entities
    br.edu.ufcg.lsd.oursim.entities.request
    java.lang
    java.util

Used By:
    br.edu.ufcg.lsd.oursim.entities
    br.edu.ufcg.lsd.oursim.entities.accounting
    br.edu.ufcg.lsd.oursim.entities.grid
    br.edu.ufcg.lsd.oursim.entities.request
    br.edu.ufcg.lsd.oursim.events.broker
    br.edu.ufcg.lsd.oursim.events.peer.accounting
    br.edu.ufcg.lsd.oursim.events.worker
    br.edu.ufcg.lsd.oursim.experiments
    br.edu.ufcg.lsd.oursim.trace

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.entities.request
-------------------------------------------------

Stats:
    Total Classes: 3
    Concrete Classes: 3
    Abstract Classes: 0

    Ca: 7
    Ce: 3

    A: 0
    I: 0,3
    D: 0,7

Abstract Classes:

Concrete Classes:
    br.edu.ufcg.lsd.oursim.entities.request.BrokerRequest
    br.edu.ufcg.lsd.oursim.entities.request.PeerRequest
    br.edu.ufcg.lsd.oursim.entities.request.RequestSpec

Depends Upon:
    br.edu.ufcg.lsd.oursim.entities.job
    java.lang
```

```
    java.util

Used By:
    br.edu.ufcg.lsd.oursim.entities.allocation
    br.edu.ufcg.lsd.oursim.entities.grid
    br.edu.ufcg.lsd.oursim.entities.job
    br.edu.ufcg.lsd.oursim.events.broker
    br.edu.ufcg.lsd.oursim.events.peer
    br.edu.ufcg.lsd.oursim.events.peer.allocation
    br.edu.ufcg.lsd.oursim.events.worker

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.entity
-------------------------------------------------

Stats:
    Total Classes: 1
    Concrete Classes: 1
    Abstract Classes: 0

    Ca: 0
    Ce: 3

    A: 0
    I: 1
    D: 0

Abstract Classes:

Concrete Classes:
    br.edu.ufcg.lsd.oursim.entity.EntityTest

Depends Upon:
    br.edu.ufcg.lsd.oursim.entities
    java.lang
    junit.framework

Used By:
    Not used by any packages.

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.events
-------------------------------------------------

Stats:
    Total Classes: 5
    Concrete Classes: 1
```

```
    Abstract Classes: 4

    Ca: 10
    Ce: 2

    A: 0,8
    I: 0,17
    D: 0,03

Abstract Classes:
    br.edu.ufcg.lsd.oursim.events.AbstractEvent
    br.edu.ufcg.lsd.oursim.events.Event
    br.edu.ufcg.lsd.oursim.events.EventListener
    br.edu.ufcg.lsd.oursim.events.PrimaryEvent

Concrete Classes:
    br.edu.ufcg.lsd.oursim.events.EventSpec

Depends Upon:
    br.edu.ufcg.lsd.oursim
    java.lang

Used By:
    br.edu.ufcg.lsd.oursim
    br.edu.ufcg.lsd.oursim.events.broker
    br.edu.ufcg.lsd.oursim.events.ds
    br.edu.ufcg.lsd.oursim.events.fd
    br.edu.ufcg.lsd.oursim.events.global
    br.edu.ufcg.lsd.oursim.events.peer
    br.edu.ufcg.lsd.oursim.events.worker
    br.edu.ufcg.lsd.oursim.experiments
    br.edu.ufcg.lsd.oursim.factories
    br.edu.ufcg.lsd.oursim.queue

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.events.broker
-------------------------------------------------

Stats:
    Total Classes: 16
    Concrete Classes: 16
    Abstract Classes: 0

    Ca: 1
    Ce: 13

    A: 0
```

```
    I: 0,93
    D: 0,07

Abstract Classes:

Concrete Classes:
    br.edu.ufcg.lsd.oursim.events.broker.AddJobEvent
    br.edu.ufcg.lsd.oursim.events.broker.BrokerDownEvent
    br.edu.ufcg.lsd.oursim.events.broker.BrokerEvents
    br.edu.ufcg.lsd.oursim.events.broker.BrokerLoggedEvent
    br.edu.ufcg.lsd.oursim.events.broker.BrokerUpEvent
    br.edu.ufcg.lsd.oursim.events.broker.CancelJobEvent
    br.edu.ufcg.lsd.oursim.events.broker.HereIsExecutionResultEvent
    br.edu.ufcg.lsd.oursim.events.broker.HereIsWorkerEvent
    br.edu.ufcg.lsd.oursim.events.broker.PeerAvailableEvent
    br.edu.ufcg.lsd.oursim.events.broker.PeerFailedEvent
    br.edu.ufcg.lsd.oursim.events.broker.ScheduleEvent
    br.edu.ufcg.lsd.oursim.events.broker.SchedulerHelper
    br.edu.ufcg.lsd.oursim.events.broker.SetGridEvent
    br.edu.ufcg.lsd.oursim.events.broker.WorkerAvailableEvent
    br.edu.ufcg.lsd.oursim.events.broker.WorkerFailedEvent
    br.edu.ufcg.lsd.oursim.events.broker.WorkerPreemptedEvent

Depends Upon:
    br.edu.ufcg.lsd.oursim
    br.edu.ufcg.lsd.oursim.entities
    br.edu.ufcg.lsd.oursim.entities.accounting
    br.edu.ufcg.lsd.oursim.entities.grid
    br.edu.ufcg.lsd.oursim.entities.job
    br.edu.ufcg.lsd.oursim.entities.request
    br.edu.ufcg.lsd.oursim.events
    br.edu.ufcg.lsd.oursim.events.fd
    br.edu.ufcg.lsd.oursim.trace
    br.edu.ufcg.lsd.oursim.util
    java.lang
    java.util
    org.json

Used By:
    br.edu.ufcg.lsd.oursim.factories

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.events.ds
-------------------------------------------------

Stats:
    Total Classes: 4
```

```
    Concrete Classes: 4
    Abstract Classes: 0

    Ca: 1
    Ce: 6

    A: 0
    I: 0,86
    D: 0,14

Abstract Classes:

Concrete Classes:
    br.edu.ufcg.lsd.oursim.events.ds.DiscoveryServiceDownEvent
    br.edu.ufcg.lsd.oursim.events.ds.DiscoveryServiceEvents
    br.edu.ufcg.lsd.oursim.events.ds.DiscoveryServiceUpEvent
    br.edu.ufcg.lsd.oursim.events.ds.GetWorkerProvidersEvent

Depends Upon:
    br.edu.ufcg.lsd.oursim
    br.edu.ufcg.lsd.oursim.entities.grid
    br.edu.ufcg.lsd.oursim.events
    br.edu.ufcg.lsd.oursim.events.fd
    java.lang
    java.util

Used By:
    br.edu.ufcg.lsd.oursim.factories

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.events.fd
-------------------------------------------------

Stats:
    Total Classes: 9
    Concrete Classes: 7
    Abstract Classes: 2

    Ca: 5
    Ce: 6

    A: 0,22
    I: 0,55
    D: 0,23

Abstract Classes:
    br.edu.ufcg.lsd.oursim.events.fd.ActiveEntityDownEvent
```

```
    br.edu.ufcg.lsd.oursim.events.fd.ActiveEntityUpEvent

Concrete Classes:
    br.edu.ufcg.lsd.oursim.events.fd.FailureDetectionEvents
    br.edu.ufcg.lsd.oursim.events.fd.IsItAliveReceivedEvent
    br.edu.ufcg.lsd.oursim.events.fd.IsItAliveSentEvent
    br.edu.ufcg.lsd.oursim.events.fd.LivenessCheckEvent
    br.edu.ufcg.lsd.oursim.events.fd.MonitorUtil
    br.edu.ufcg.lsd.oursim.events.fd.ReleaseMonitoredEvent
    br.edu.ufcg.lsd.oursim.events.fd.UpdateStatusAvailableEvent

Depends Upon:
    br.edu.ufcg.lsd.oursim
    br.edu.ufcg.lsd.oursim.entities
    br.edu.ufcg.lsd.oursim.entities.grid
    br.edu.ufcg.lsd.oursim.events
    java.lang
    java.util

Used By:
    br.edu.ufcg.lsd.oursim.events.broker
    br.edu.ufcg.lsd.oursim.events.ds
    br.edu.ufcg.lsd.oursim.events.peer
    br.edu.ufcg.lsd.oursim.events.worker
    br.edu.ufcg.lsd.oursim.factories

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.events.global
-------------------------------------------------

Stats:
    Total Classes: 2
    Concrete Classes: 1
    Abstract Classes: 1

    Ca: 2
    Ce: 3

    A: 0,5
    I: 0,6
    D: 0,1

Abstract Classes:
    br.edu.ufcg.lsd.oursim.events.global.HaltCondition

Concrete Classes:
    br.edu.ufcg.lsd.oursim.events.global.HaltEvent
```

```
Depends Upon:
    br.edu.ufcg.lsd.oursim
    br.edu.ufcg.lsd.oursim.events
    java.lang

Used By:
    br.edu.ufcg.lsd.oursim.experiments
    br.edu.ufcg.lsd.oursim.factories

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.events.peer
-------------------------------------------------

Stats:
    Total Classes: 29
    Concrete Classes: 29
    Abstract Classes: 0

    Ca: 3
    Ce: 12

    A: 0
    I: 0,8
    D: 0,2

Abstract Classes:

Concrete Classes:
    br.edu.ufcg.lsd.oursim.events.peer.BrokerAvailableEvent
    br.edu.ufcg.lsd.oursim.events.peer.BrokerFailedEvent
    br.edu.ufcg.lsd.oursim.events.peer.BrokerLoginEvent
    br.edu.ufcg.lsd.oursim.events.peer.DiscoveryServiceFailedEvent
    br.edu.ufcg.lsd.oursim.events.peer.DisposeRemoteWorkerEvent
    br.edu.ufcg.lsd.oursim.events.peer.DisposeWorkerEvent
    br.edu.ufcg.lsd.oursim.events.peer.FinishRequestEvent
    br.edu.ufcg.lsd.oursim.events.peer.HereAreWorkerProvidersEvent
    br.edu.ufcg.lsd.oursim.events.peer.HereIsRemoteWorkerEvent
    br.edu.ufcg.lsd.oursim.events.peer.LocalWorkerAvailableEvent
    br.edu.ufcg.lsd.oursim.events.peer.LocalWorkerFailedEvent
    br.edu.ufcg.lsd.oursim.events.peer.PauseRequestEvent
    br.edu.ufcg.lsd.oursim.events.peer.PeerDownEvent
    br.edu.ufcg.lsd.oursim.events.peer.PeerEvents
    br.edu.ufcg.lsd.oursim.events.peer.PeerUpEvent
    br.edu.ufcg.lsd.oursim.events.peer.RemoteRequestWorkersEvent
    br.edu.ufcg.lsd.oursim.events.peer.RemoteWorkerAvailableEvent
    br.edu.ufcg.lsd.oursim.events.peer.RemoteWorkerFailedEvent
```

```
    br.edu.ufcg.lsd.oursim.events.peer.RepeatGetWorkerProvidersEvent
    br.edu.ufcg.lsd.oursim.events.peer.ReportReplicaAccountingEvent
    br.edu.ufcg.lsd.oursim.events.peer.ReportWorkAccountingEvent
    br.edu.ufcg.lsd.oursim.events.peer.RequestWorkersEvent
    br.edu.ufcg.lsd.oursim.events.peer.ResumeRequestEvent
    br.edu.ufcg.lsd.oursim.events.peer.WorkerAvailableEvent
    br.edu.ufcg.lsd.oursim.events.peer.WorkerDonatedEvent
    br.edu.ufcg.lsd.oursim.events.peer.WorkerFailedEvent
    br.edu.ufcg.lsd.oursim.events.peer.WorkerIdleEvent
    br.edu.ufcg.lsd.oursim.events.peer.WorkerInUseEvent
    br.edu.ufcg.lsd.oursim.events.peer.WorkerState

Depends Upon:
    br.edu.ufcg.lsd.oursim
    br.edu.ufcg.lsd.oursim.entities
    br.edu.ufcg.lsd.oursim.entities.accounting
    br.edu.ufcg.lsd.oursim.entities.allocation
    br.edu.ufcg.lsd.oursim.entities.grid
    br.edu.ufcg.lsd.oursim.entities.request
    br.edu.ufcg.lsd.oursim.events
    br.edu.ufcg.lsd.oursim.events.fd
    br.edu.ufcg.lsd.oursim.events.peer.accounting
    br.edu.ufcg.lsd.oursim.events.peer.allocation
    java.lang
    java.util

Used By:
    br.edu.ufcg.lsd.oursim.entities.grid
    br.edu.ufcg.lsd.oursim.events.peer.allocation
    br.edu.ufcg.lsd.oursim.factories

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.events.peer.accounting
-------------------------------------------------

Stats:
    Total Classes: 5
    Concrete Classes: 4
    Abstract Classes: 1

    Ca: 1
    Ce: 5

    A: 0,2
    I: 0,83
    D: 0,03
```

```
Abstract Classes:
    br.edu.ufcg.lsd.oursim.events.peer.accounting.AccountingStrategy

Concrete Classes:
    br.edu.ufcg.lsd.oursim.events.peer.accounting.AccountingEvaluator
    br.edu.ufcg.lsd.oursim.events.peer.accounting.AccountingHelper
    br.edu.ufcg.lsd.oursim.events.peer.accounting.RelativeCPUAccountingStrategy
    br.edu.ufcg.lsd.oursim.events.peer.accounting.RemoteCPUAccountingStrategy

Depends Upon:
    br.edu.ufcg.lsd.oursim.entities.accounting
    br.edu.ufcg.lsd.oursim.entities.grid
    br.edu.ufcg.lsd.oursim.entities.job
    java.lang
    java.util

Used By:
    br.edu.ufcg.lsd.oursim.events.peer

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.events.peer.allocation
-------------------------------------------------

Stats:
    Total Classes: 8
    Concrete Classes: 8
    Abstract Classes: 0

    Ca: 1
    Ce: 7

    A: 0
    I: 0,88
    D: 0,12

Abstract Classes:

Concrete Classes:
    br.edu.ufcg.lsd.oursim.events.peer.allocation.AllocationHelper
    br.edu.ufcg.lsd.oursim.events.peer.allocation.AllocationHelper$1
    br.edu.ufcg.lsd.oursim.events.peer.allocation.AllocationHelper$2
    br.edu.ufcg.lsd.oursim.events.peer.allocation.AllocationHelper$3
    br.edu.ufcg.lsd.oursim.events.peer.allocation.AllocationHelper$Priority
    br.edu.ufcg.lsd.oursim.events.peer.allocation.LowerPriorityAllocationHelper
    br.edu.ufcg.lsd.oursim.events.peer.allocation.SamePriorityAllocationHelper
    br.edu.ufcg.lsd.oursim.events.peer.allocation.WorkerDistributionHelper
```

```
Depends Upon:
    br.edu.ufcg.lsd.oursim
    br.edu.ufcg.lsd.oursim.entities.allocation
    br.edu.ufcg.lsd.oursim.entities.grid
    br.edu.ufcg.lsd.oursim.entities.request
    br.edu.ufcg.lsd.oursim.events.peer
    java.lang
    java.util

Used By:
    br.edu.ufcg.lsd.oursim.events.peer

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.events.worker
-------------------------------------------------

Stats:
    Total Classes: 11
    Concrete Classes: 11
    Abstract Classes: 0

    Ca: 1
    Ce: 9

    A: 0
    I: 0,9
    D: 0,1

Abstract Classes:

Concrete Classes:
    br.edu.ufcg.lsd.oursim.events.worker.CleanWorkerHelper
    br.edu.ufcg.lsd.oursim.events.worker.SendHereIsExecutionEvent
    br.edu.ufcg.lsd.oursim.events.worker.SendReportWorkAccountingEvent
    br.edu.ufcg.lsd.oursim.events.worker.SetPeerEvent
    br.edu.ufcg.lsd.oursim.events.worker.StartWorkEvent
    br.edu.ufcg.lsd.oursim.events.worker.StopWorkingEvent
    br.edu.ufcg.lsd.oursim.events.worker.WorkForBrokerEvent
    br.edu.ufcg.lsd.oursim.events.worker.WorkForPeerEvent
    br.edu.ufcg.lsd.oursim.events.worker.WorkerDownEvent
    br.edu.ufcg.lsd.oursim.events.worker.WorkerEvents
    br.edu.ufcg.lsd.oursim.events.worker.WorkerUpEvent

Depends Upon:
    br.edu.ufcg.lsd.oursim
    br.edu.ufcg.lsd.oursim.entities.accounting
    br.edu.ufcg.lsd.oursim.entities.grid
```

```
    br.edu.ufcg.lsd.oursim.entities.job
    br.edu.ufcg.lsd.oursim.entities.request
    br.edu.ufcg.lsd.oursim.events
    br.edu.ufcg.lsd.oursim.events.fd
    java.lang
    java.util

Used By:
    br.edu.ufcg.lsd.oursim.factories

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.experiments
-------------------------------------------------

Stats:
    Total Classes: 12
    Concrete Classes: 12
    Abstract Classes: 0

    Ca: 0
    Ce: 18

    A: 0
    I: 1
    D: 0

Abstract Classes:

Concrete Classes:
    br.edu.ufcg.lsd.oursim.experiments.JDependRunner
    br.edu.ufcg.lsd.oursim.experiments.LocalConcurrentScenarioGenerator
    br.edu.ufcg.lsd.oursim.experiments.LocalConcurrentScenarioGenerator$JobFinishedCondition
    br.edu.ufcg.lsd.oursim.experiments.LocalScenarioGenerator
    br.edu.ufcg.lsd.oursim.experiments.LocalScenarioGenerator$JobFinishedCondition
    br.edu.ufcg.lsd.oursim.experiments.RemoteAndLocalConcurrentScenarioGenerator
    br.edu.ufcg.lsd.oursim.experiments.RemoteAndLocalConcurrentScenarioGenerator$JobFinishedCondition
    br.edu.ufcg.lsd.oursim.experiments.RemoteNoFScenarioGenerator
    br.edu.ufcg.lsd.oursim.experiments.RemoteNoFScenarioGenerator$JobFinishedCondition
    br.edu.ufcg.lsd.oursim.experiments.ScalabilityScenarioGenerator
    br.edu.ufcg.lsd.oursim.experiments.ScalabilityScenarioGenerator$JobFinishedCondition
    br.edu.ufcg.lsd.oursim.experiments.ScenarioEvaluator

Depends Upon:
    br.edu.ufcg.lsd.oursim
    br.edu.ufcg.lsd.oursim.entities
    br.edu.ufcg.lsd.oursim.entities.grid
    br.edu.ufcg.lsd.oursim.entities.job
```

```
    br.edu.ufcg.lsd.oursim.events
    br.edu.ufcg.lsd.oursim.events.global
    br.edu.ufcg.lsd.oursim.fd
    br.edu.ufcg.lsd.oursim.network
    br.edu.ufcg.lsd.oursim.queue
    br.edu.ufcg.lsd.oursim.trace
    br.edu.ufcg.lsd.oursim.util
    java.io
    java.lang
    java.util
    jdepend.framework
    jdepend.textui
    org.apache.commons.io
    org.json

Used By:
    Not used by any packages.

-----------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.factories
-----------------------------------------------------

Stats:
    Total Classes: 4
    Concrete Classes: 3
    Abstract Classes: 1

    Ca: 2
    Ce: 18

    A: 0,25
    I: 0,9
    D: 0,15

Abstract Classes:
    br.edu.ufcg.lsd.oursim.factories.GridFactory

Concrete Classes:
    br.edu.ufcg.lsd.oursim.factories.EventFactory
    br.edu.ufcg.lsd.oursim.factories.JsonGridFactory
    br.edu.ufcg.lsd.oursim.factories.JsonGridFactoryTest

Depends Upon:
    br.edu.ufcg.lsd.oursim.entities
    br.edu.ufcg.lsd.oursim.entities.grid
    br.edu.ufcg.lsd.oursim.events
    br.edu.ufcg.lsd.oursim.events.broker
```

```
    br.edu.ufcg.lsd.oursim.events.ds
    br.edu.ufcg.lsd.oursim.events.fd
    br.edu.ufcg.lsd.oursim.events.global
    br.edu.ufcg.lsd.oursim.events.peer
    br.edu.ufcg.lsd.oursim.events.worker
    br.edu.ufcg.lsd.oursim.fd
    br.edu.ufcg.lsd.oursim.util
    java.io
    java.lang
    java.lang.reflect
    java.util
    junit.framework
    org.apache.commons.io
    org.json

Used By:
    br.edu.ufcg.lsd.oursim
    br.edu.ufcg.lsd.oursim.queue

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.fd
-------------------------------------------------

Stats:
    Total Classes: 27
    Concrete Classes: 25
    Abstract Classes: 2

    Ca: 3
    Ce: 6

    A: 0,07
    I: 0,67
    D: 0,26

Abstract Classes:
    br.edu.ufcg.lsd.oursim.fd.AbstractFailureDetector
    br.edu.ufcg.lsd.oursim.fd.FailureDetector

Concrete Classes:
    br.edu.ufcg.lsd.oursim.fd.AbstractFDTest
    br.edu.ufcg.lsd.oursim.fd.AbstractFDTest$SimpleFailureDetector
    br.edu.ufcg.lsd.oursim.fd.BertierFDTest
    br.edu.ufcg.lsd.oursim.fd.BertierFailureDetector
    br.edu.ufcg.lsd.oursim.fd.BertierFailureDetector$BertierMonitored
    br.edu.ufcg.lsd.oursim.fd.ChenFDTest
    br.edu.ufcg.lsd.oursim.fd.ChenFailureDetector
```

```
    br.edu.ufcg.lsd.oursim.fd.ChenFailureDetector$ChenMonitored
    br.edu.ufcg.lsd.oursim.fd.FailureDetectorFactory
    br.edu.ufcg.lsd.oursim.fd.FailureDetectorFactoryTest
    br.edu.ufcg.lsd.oursim.fd.FailureDetectorOptInjector
    br.edu.ufcg.lsd.oursim.fd.FailureDetectorOptParser
    br.edu.ufcg.lsd.oursim.fd.FailureDetectorOptParserTest
    br.edu.ufcg.lsd.oursim.fd.FixedHeartbeatFDTest
    br.edu.ufcg.lsd.oursim.fd.FixedPingFailureDetector
    br.edu.ufcg.lsd.oursim.fd.InterArrivalSamplingWindow
    br.edu.ufcg.lsd.oursim.fd.InterArrivalSamplingWindowTest
    br.edu.ufcg.lsd.oursim.fd.MessageType
    br.edu.ufcg.lsd.oursim.fd.Monitored
    br.edu.ufcg.lsd.oursim.fd.PhiAccrualFDTest
    br.edu.ufcg.lsd.oursim.fd.PhiAccrualFailureDetector
    br.edu.ufcg.lsd.oursim.fd.PhiAccrualFailureDetector$PhiAccrualMonitored
    br.edu.ufcg.lsd.oursim.fd.SlicedHeartbeatFDTest
    br.edu.ufcg.lsd.oursim.fd.SlicedPingFailureDetector
    br.edu.ufcg.lsd.oursim.fd.SlicedPingFailureDetector$SlicedMonitored

Depends Upon:
    br.edu.ufcg.lsd.oursim.entities
    br.edu.ufcg.lsd.oursim.entities.grid
    java.lang
    java.lang.reflect
    java.util
    junit.framework

Used By:
    br.edu.ufcg.lsd.oursim.entities
    br.edu.ufcg.lsd.oursim.experiments
    br.edu.ufcg.lsd.oursim.factories

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.network
-------------------------------------------------

Stats:
    Total Classes: 4
    Concrete Classes: 3
    Abstract Classes: 1

    Ca: 2
    Ce: 4

    A: 0,25
    I: 0,67
    D: 0,08
```

```
Abstract Classes:
    br.edu.ufcg.lsd.oursim.network.Network

Concrete Classes:
    br.edu.ufcg.lsd.oursim.network.BlankNetwork
    br.edu.ufcg.lsd.oursim.network.DefaultWANNetwork
    br.edu.ufcg.lsd.oursim.network.DefaultWANNetworkTest

Depends Upon:
    java.lang
    org.junit
    umontreal.iro.lecuyer.randvar
    umontreal.iro.lecuyer.rng

Used By:
    br.edu.ufcg.lsd.oursim
    br.edu.ufcg.lsd.oursim.experiments

-------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.queue
-------------------------------------------------

Stats:
    Total Classes: 7
    Concrete Classes: 6
    Abstract Classes: 1

    Ca: 2
    Ce: 9

    A: 0,14
    I: 0,82
    D: 0,04

Abstract Classes:
    br.edu.ufcg.lsd.oursim.queue.EventProxy

Concrete Classes:
    br.edu.ufcg.lsd.oursim.queue.BlankEvent
    br.edu.ufcg.lsd.oursim.queue.EventQueue
    br.edu.ufcg.lsd.oursim.queue.EventQueueTest
    br.edu.ufcg.lsd.oursim.queue.FileEventProxy
    br.edu.ufcg.lsd.oursim.queue.FileEventProxyTest
    br.edu.ufcg.lsd.oursim.queue.ListEventProxy

Depends Upon:
```

```
    br.edu.ufcg.lsd.oursim
    br.edu.ufcg.lsd.oursim.events
    br.edu.ufcg.lsd.oursim.factories
    br.edu.ufcg.lsd.oursim.util
    java.io
    java.lang
    java.util
    junit.framework
    org.junit

Used By:
    br.edu.ufcg.lsd.oursim
    br.edu.ufcg.lsd.oursim.experiments

------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.trace
------------------------------------------------

Stats:
    Total Classes: 3
    Concrete Classes: 2
    Abstract Classes: 1

    Ca: 3
    Ce: 4

    A: 0,33
    I: 0,57
    D: 0,1

Abstract Classes:
    br.edu.ufcg.lsd.oursim.trace.TraceCollector

Concrete Classes:
    br.edu.ufcg.lsd.oursim.trace.DefaultTraceCollector
    br.edu.ufcg.lsd.oursim.trace.DefaultTraceCollectorTest

Depends Upon:
    br.edu.ufcg.lsd.oursim.entities.job
    java.io
    java.lang
    junit.framework

Used By:
    br.edu.ufcg.lsd.oursim
    br.edu.ufcg.lsd.oursim.events.broker
    br.edu.ufcg.lsd.oursim.experiments
```

```
------------------------------------------------
- Package: br.edu.ufcg.lsd.oursim.util
------------------------------------------------

Stats:
    Total Classes: 3
    Concrete Classes: 3
    Abstract Classes: 0

    Ca: 5
    Ce: 3

    A: 0
    I: 0,38
    D: 0,62

Abstract Classes:

Concrete Classes:
    br.edu.ufcg.lsd.oursim.util.Configuration
    br.edu.ufcg.lsd.oursim.util.JSONUtils
    br.edu.ufcg.lsd.oursim.util.LineParser

Depends Upon:
    java.lang
    java.util
    org.json

Used By:
    br.edu.ufcg.lsd.oursim
    br.edu.ufcg.lsd.oursim.events.broker
    br.edu.ufcg.lsd.oursim.experiments
    br.edu.ufcg.lsd.oursim.factories
    br.edu.ufcg.lsd.oursim.queue

------------------------------------------------
- Package Dependency Cycles:
------------------------------------------------

br.edu.ufcg.lsd.oursim
    |
    |   br.edu.ufcg.lsd.oursim.trace
    |   br.edu.ufcg.lsd.oursim.entities.job
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities
```

```
br.edu.ufcg.lsd.oursim.entities
    |
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.entities.accounting
    |
    |   br.edu.ufcg.lsd.oursim.entities.job
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.entities.allocation
    |
    |   br.edu.ufcg.lsd.oursim.entities.request
    |   br.edu.ufcg.lsd.oursim.entities.job
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.entities.grid
    |
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.entities.job
    |
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.entities.request
    |
    |   br.edu.ufcg.lsd.oursim.entities.job
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.entity
    |
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.events
```

```
    |
    |   br.edu.ufcg.lsd.oursim
    |   br.edu.ufcg.lsd.oursim.trace
    |   br.edu.ufcg.lsd.oursim.entities.job
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.events.broker
    |
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.events.ds
    |
    |   br.edu.ufcg.lsd.oursim
    |   br.edu.ufcg.lsd.oursim.trace
    |   br.edu.ufcg.lsd.oursim.entities.job
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.events.fd
    |
    |   br.edu.ufcg.lsd.oursim.events
    |   br.edu.ufcg.lsd.oursim
    |   br.edu.ufcg.lsd.oursim.trace
    |   br.edu.ufcg.lsd.oursim.entities.job
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.events.global
    |
    |   br.edu.ufcg.lsd.oursim.events
    |   br.edu.ufcg.lsd.oursim
    |   br.edu.ufcg.lsd.oursim.trace
    |   br.edu.ufcg.lsd.oursim.entities.job
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.events.peer
    |
    |   br.edu.ufcg.lsd.oursim.events
    |   br.edu.ufcg.lsd.oursim
```

```
    |   br.edu.ufcg.lsd.oursim.trace
    |   br.edu.ufcg.lsd.oursim.entities.job
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.events.peer.accounting
    |
    |   br.edu.ufcg.lsd.oursim.entities.accounting
    |   br.edu.ufcg.lsd.oursim.entities.job
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.events.peer.allocation
    |
    |   br.edu.ufcg.lsd.oursim.entities.allocation
    |   br.edu.ufcg.lsd.oursim.entities.request
    |   br.edu.ufcg.lsd.oursim.entities.job
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.events.worker
    |
    |   br.edu.ufcg.lsd.oursim.entities.accounting
    |   br.edu.ufcg.lsd.oursim.entities.job
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.experiments
    |
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.factories
    |
    |   br.edu.ufcg.lsd.oursim.events.worker
    |   br.edu.ufcg.lsd.oursim.entities.accounting
    |   br.edu.ufcg.lsd.oursim.entities.job
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.fd
```

```
    |
    |   br.edu.ufcg.lsd.oursim.entities
    |-> br.edu.ufcg.lsd.oursim.fd

br.edu.ufcg.lsd.oursim.queue
    |
    |   br.edu.ufcg.lsd.oursim.events
    |   br.edu.ufcg.lsd.oursim
    |   br.edu.ufcg.lsd.oursim.trace
    |   br.edu.ufcg.lsd.oursim.entities.job
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities

br.edu.ufcg.lsd.oursim.trace
    |
    |   br.edu.ufcg.lsd.oursim.entities.job
    |-> br.edu.ufcg.lsd.oursim.entities
    |   br.edu.ufcg.lsd.oursim.fd
    |-> br.edu.ufcg.lsd.oursim.entities


-------------------------------------------------
- Summary:
-------------------------------------------------

Name, Class Count, Abstract Class Count, Ca, Ce, A, I, D, V:

br.edu.ufcg.lsd.oursim,2,0,10,10,0,0,5,0,5,1
br.edu.ufcg.lsd.oursim.entities,4,0,9,4,0,0,31,0,69,1
br.edu.ufcg.lsd.oursim.entities.accounting,2,0,5,2,0,0,29,0,71,1
br.edu.ufcg.lsd.oursim.entities.allocation,2,0,3,3,0,0,5,0,5,1
br.edu.ufcg.lsd.oursim.entities.grid,5,0,11,8,0,0,42,0,58,1
br.edu.ufcg.lsd.oursim.entities.job,4,0,9,4,0,0,31,0,69,1
br.edu.ufcg.lsd.oursim.entities.request,3,0,7,3,0,0,3,0,7,1
br.edu.ufcg.lsd.oursim.entity,1,0,0,3,0,1,0,1
br.edu.ufcg.lsd.oursim.events,5,4,10,2,0,8,0,17,0,03,1
br.edu.ufcg.lsd.oursim.events.broker,16,0,1,13,0,0,93,0,07,1
br.edu.ufcg.lsd.oursim.events.ds,4,0,1,6,0,0,86,0,14,1
br.edu.ufcg.lsd.oursim.events.fd,9,2,5,6,0,22,0,55,0,23,1
br.edu.ufcg.lsd.oursim.events.global,2,1,2,3,0,5,0,6,0,1,1
br.edu.ufcg.lsd.oursim.events.peer,29,0,3,12,0,0,8,0,2,1
br.edu.ufcg.lsd.oursim.events.peer.accounting,5,1,1,5,0,2,0,83,0,03,1
br.edu.ufcg.lsd.oursim.events.peer.allocation,8,0,1,7,0,0,88,0,12,1
br.edu.ufcg.lsd.oursim.events.worker,11,0,1,9,0,0,9,0,1,1
br.edu.ufcg.lsd.oursim.experiments,12,0,0,18,0,1,0,1
br.edu.ufcg.lsd.oursim.factories,4,1,2,18,0,25,0,9,0,15,1
```

```
br.edu.ufcg.lsd.oursim.fd,27,2,3,6,0,07,0,67,0,26,1
br.edu.ufcg.lsd.oursim.network,4,1,2,4,0,25,0,67,0,08,1
br.edu.ufcg.lsd.oursim.queue,7,1,2,9,0,14,0,82,0,04,1
br.edu.ufcg.lsd.oursim.trace,3,1,3,4,0,33,0,57,0,1,1
br.edu.ufcg.lsd.oursim.util,3,0,5,3,0,0,38,0,62,1
java.io,0,0,5,0,0,0,1,1
java.lang,0,0,24,0,0,0,1,1
java.lang.reflect,0,0,2,0,0,0,1,1
java.util,0,0,18,0,0,0,1,1
jdepend.framework,0,0,1,0,0,0,1,1
jdepend.textui,0,0,1,0,0,0,1,1
junit.framework,0,0,5,0,0,0,1,1
org.apache.commons.io,0,0,2,0,0,0,1,1
org.json,0,0,4,0,0,0,1,1
org.junit,0,0,2,0,0,0,1,1
umontreal.iro.lecuyer.randvar,0,0,1,0,0,0,1,1
umontreal.iro.lecuyer.rng,0,0,1,0,0,0,1,1
```