Universidade Federal do Ceará

Centro de Ciências

Mestrado e Doutorado em Ciência da Computação

Um Ambiente de Execução *Peer-to-peer* para Escalonamento Dinâmico de *Workflows* em Grades Computacionais

Dissertação de Mestrado

João Marcelo Uchôa de Alencar

Orientadora: Rossana Maria de Castro Andrade, PhD.

Co-Orientador: Windson Viana de Carvalho, DSc.

Fortaleza, Ceará

2010

João Marcelo Uchôa de Alencar

Um Ambiente de Execução *Peer-to-peer* para Escalonamento Dinâmico de *Workflows* em Grades Computacionais

Dissertação submetida a coordenação do programa de Mestrado e Doutorado em Ciência da Computação, da Universidade Federal do Ceará como requisito parcial para a obtenção do grau de Mestre em Ciências da Computação.

Orientadora: Profa. Dra. Rossana Maria de Castro Andrade

Co-Orientador: Prof. Dr. Windson Viana de Carvalho

Fortaleza, Ceará

2010

João Marcelo Uchôa de Alencar

Um Ambiente de Execução *Peer-to-peer* para Escalonamento Dinâmico de *Workflows* em Grades Computacionais

Dissertação submetida a coordenação do programa de Mestrado e Doutorado em Ciência da Computação, da Universidade Federal do Ceará como requisito parcial para a obtenção do grau de Mestre em Ciências da Computação.

Aprovada em: __/__/__

BANCA EXAMINADORA

Profa. Dra. Rossana Maria de Castro Andrade
(Orientadora)
Universidade Federal do Ceará - UFC

Prof. Dr. Windson Viana de Carvalho
(Co-Orientador)
Universidade Federal do Ceará - UFC

Prof. Dr. Bruno Richard Schulze
Laboratório Nacional de Computação Científica - LNCC

Prof. Dr. José Neuman de Souza
Universidade Federal do Ceará - UFC

Prof. Dr. Danielo Gonçalves Gomes
Universidade Federal do Ceará - UFC

Dedico esta dissertação
Aos meus pais João Alencar de Oliveira e
Josefa Francisca Uchôa de Alencar.

Agradecimentos

Não há como começar qualquer agradecimentos sem reconhecer o papel crucial que meus pais desempenharam nas minhas conquistas. Apesar de ter saído de casa cedo, seus conselhos e orientações foram essenciais para o meu sucesso. O pragmatismo de meu pai, João Alencar, sempre me guiou para longe dos caminhos mais árduos. A paciência de minha mãe, Josefa Francisca, manteve-se como uma lembrança que as recompensas de esforços legítimos podem demorar, mas sempre chegam. Quanto aos meus irmãos, espero que o reflexo do meu sucesso sirva de incentivo para que atinjam objetivos cada vez mais ambiciosos, assim como as conquistas deles já me incentivaram neste trabalho.

Outra pessoa que não pode ser esquecida é minha namorada, Germana. Desde o período final da graduação, ela tem me acompanhado durante todos os percalços da minha vida acadêmica. Sua paciência e compreensão ajudou em muito a superar os finais de semanas de dedicação aos estudos e as noites mal dormidas. Nestes anos atingimos muitas metas e tenho certeza que com ela ao meu lado conseguiremos muito mais.

Agradeço à minha orientadora Rossana e ao meu co-orientador Windson. A professora Rossana confiou em mim quando era pouco mais do que um menino, sendo que mal posso mensurar o quanto amadureci durante sua tutela. Sua força de vontade foi um farol importante na conclusão deste trabalho. Já o Windson trouxe um fôlego contagiante para a conclusão da minha jornada no mestrado, seus conselhos e dicas contribuíram em muito para a qualidade deste trabalho e para futuros projetos da minha carreira.

Agradeço também aos professores do MDCC, em especial àqueles que ministraram as disciplinas que participei: Ana Teresa, Bernadete, Carlos e Francisco Heron. Não poderia esquecer do secretário José Orley, que desde o primeiro dia me ajudou em todas as pendências relativas ao curso.

Um agradecimento especial aos professores Neuman, Danielo e Bruno por terem aceitado participar da minha banca.

Aos colegas da família GREat, só posso agradecer pelo companheirismo e paciência. Sei que consigo ser insuportável algumas vezes, então além de agradecer, também peço perdão. É difícil fazer uma lista, mas se esquecer de alguém não é por má vontade. Em ordem alfabética,

gostaria de lembrar desses caros amigos: Bosco, Bringel, Bruno Góis, Bruno Sabóia, Cláudio, Christiane, Danilo, Davi, Darilu, Diana, Fabiana, Fabrício, Flávio, Felipe, João Borges, José Adir, Levi, Liliane, Lincoln, Luana, Márcio, Marcos Dantas, Michel, Mirko, Paulo Henrique, Reinaldo, Rute, Suzana e Valéria.

Por último, agradeço ao CNPq. Desde que virei bolsista, mesmo antes do mestrado, sempre recebi em dia. E quando precisei entrar em contato com o órgão em Brasília, sempre fui muito bem atendido. Espero poder um dia completar o ciclo e de orientando passar a orientador de bolsistas do CNPq.

Epígrafe

”For small creatures such as we the vastness is bearable only through love. ”
Carl Sagan

Resumo

Grades computacionais são infraestruturas que integram em larga escala recursos distribuídos e heterogêneos. A fim de suportar experimentos científicos complexos, os serviços disponíveis em uma grade precisam ser orquestrados de maneira eficiente. A utilização de *workflows* para representar orquestrações de serviços em grades computacionais proporciona ao usuário uma visão geral da sua aplicação, compreendendo o fluxo de controle e a troca de dados entre os serviços. O escalonamento adequado de *workflows* é crucial para a manutenção do sistema e a satisfação do usuário, sendo importante considerar métricas de qualidade de serviço (QoS) definidas pelo usuário. As soluções existentes para o escalonamento de *workflows* de serviços em grades computacionais não estabelecem arquiteturas descentralizadas que considerem diversas métricas de QoS. Desta forma, os sistemas atuais são suscetíveis a gargalos de desempenho. Nesta dissertação propomos uma solução para a execução *peer-to-peer* de *workflows* baseada na invocação de serviços de grades e que permite a definição de prioridades entre métricas de QoS. A proposta considera tanto informações do contexto do ambiente de execução da grade quanto os requisitos de qualidade de serviços que são informados pelo usuário. A meta é permitir a adaptação do escalonamento do *workflow*. Para a avaliação da proposta, duas comparações são feitas. Primeiro, o sistema é analisado em relação a uma solução descentralizada existente, a qual não considera métricas de QoS. Segundo, o sistema é comparado com o estado da arte em escalonamento centralizado, que considera métricas de QoS. Nesse caso, o objetivo de averiguar o impacto da descentralização na qualidade da execução de *workflows*.

Palavras-chave: Sistemas Distribuídos; Grades Computacionais; Redes *peer-to-peer*; *Workflows*.

Abstract

Grid computing is a suitable infrastructure for integrating large scale, distributed and heterogeneous resources. In order to support complex scientific experiments, services available in a grid need to be orchestrated effectively. The use of workflows to represent orchestrations of services in grid computing gives users an overview of their applications, including control flow and data exchange between services. The proper scheduling of workflows is crucial for the maintenance of the system and user satisfaction, being important to consider quality of service (QoS) metrics defined by the user. Existing solutions for the deployment of service workflows in computational grids do not establish decentralized architectures that consider several QoS metrics. Thus, current systems are susceptible to performance bottlenecks. In this work, we propose a solution for the peer-to-peer execution of workflows based on the invocation of grid services that allows the definition of priorities among QoS metrics. The proposal considers both the context of the grid execution environment and the requirements of quality of services informed by the user. The goal is allow adaptation of the workflow scheduling. For the proposal evaluation, two comparisons are made. First, the system is analyzed in relation to an existing decentralized solution that does not consider QoS metrics. Second, the system is compared with the state of art of centralized scheduling, which considers QoS metrics. In this case, the goal is to investigate the impact of decentralization on the quality of execution of workflows.

Keywords: Distributed Computing; Grid Computing; Peer-to-peer Computing; *Workflows*.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

SOA	Service-Oriented Architecture
OGSA	Open Grid Services Architecture
QoS	Quality Of Service
ESG	*Earth System Grid*
OGSA	*The Open Grid Services Architecture*
BPEL	*Business Process Execution Language*
YAWL	*Yet Another Workflow Language*
SETI	Search for Extraterrestrial Intelligence
DHT	Distributed Hash Table
SwinDeW	*Swinburne Decentralized Workflow*
GAT	*Grid Application Toolkit*
UML	*Unified Modeling Language*
AGWL	*Abstract Grid Workflow Language*
URI	Universal Resource Identifier
WSDL	Web Services Description Language
DAG	*Directed Acyclic Graph*
JGAP	*Java Genetic Algorithms Package*

Sumário

1 Introdução

Esta dissertação propõe um ambiente para execução dinâmica *peer-to-peer* de *workflows* para aplicações científicas em grades computacionais. Na seção 1.1 é discutido o contexto das grades computacionais e a execução de *workflows*. A seção 1.2 apresenta a motivação e problemática do trabalho. As contribuições e os objetivos são detalhados na seção 1.3, e por fim, a seção 1.4 explica como o restante da dissertação está organizada.

1.1 Contextualização

Grades computacionais são coleções de *clusters* ou supercomputadores geograficamente distantes, porém interconectados com o objetivo de compartilhar recursos computacionais entre várias instituições de forma a maximizar o uso dos recursos e permitir a construção de uma coleção de máquinas capaz de resolver problemas que não podem ser solucionados apenas com os recursos das instituições isoladas (KON; GOLDMAN, 2008). Essas coleções são cada vez mais utilizadas em lugar de supercomputadores ou *clusters* independentes para resolução de problemas que exigem alto poder de processamento, visto que o compartilhamento de recursos entre instituições diminui o custo financeiro e incentiva a colaboração entre as mesmas. Supercomputadores ou *clusters* dedicados a apenas um grupo restrito de usuários costumam apresentar períodos de ociosidade consideráveis, sendo natural que a computação em grade possa estabelecer uma melhor política de utilização, na qual ciclos ociosos em determinados períodos possam ser negociados por outros tipos de recursos ausentes, tal como armazenamento. Grades têm obtido sucesso na indústria e na academia. Entretanto, há diferenças em como a tecnologia é utilizada nos dois meios, relacionadas a seguir.

Nas grades empresariais, os acordos de formação refletem contratos formais já existentes entre as empresas, sendo que o cenário de implementação da grade possui regras rígidas. As aplicações em execução pertencem a um conjunto restrito ao domínio de atuação da empresa, existindo casos nos quais a grade é montada para a execução de uma aplicação exclusiva (ORACLE, 2009). Nestes casos, os usuários finais não interagem de maneira direta com os serviços

da grade, lidando apenas com a aplicação em execução sobre a infraestrutura. As tarefas ou invocações de serviços submetidas à grade não apresentam variação, já que são originadas de um conjunto limitado de aplicações.

Já as grades científicas são formadas a partir de acordo espontâneos entre instituições de pesquisa, existindo casos nos quais a filiação é voluntária (CIRNE et al., 2006). As aplicações são variadas, sendo que os usuários são pesquisadores que interagem diretamente com os serviços da grade para submissão de tarefas e invocação de serviços. Para aumentar a utilidade da grade visando o usuário final, ambientes de acesso intuitivos e abstrações são cruciais (ROURE; JENNINGS; SHADBOLT, 2005). Por se tratar de um ambiente no qual as requisições são heterogêneas, não há possibilidade de realizar otimizações específicas para uma dada aplicação, como no caso das grades empresariais. O sistema deve ser capaz de realizar o escalonamento adequado das requisições distintas para os recursos disponíveis, evitando a sobrecarga de recursos e oferecendo tempo de resposta de acordo com as necessidades do usuários.

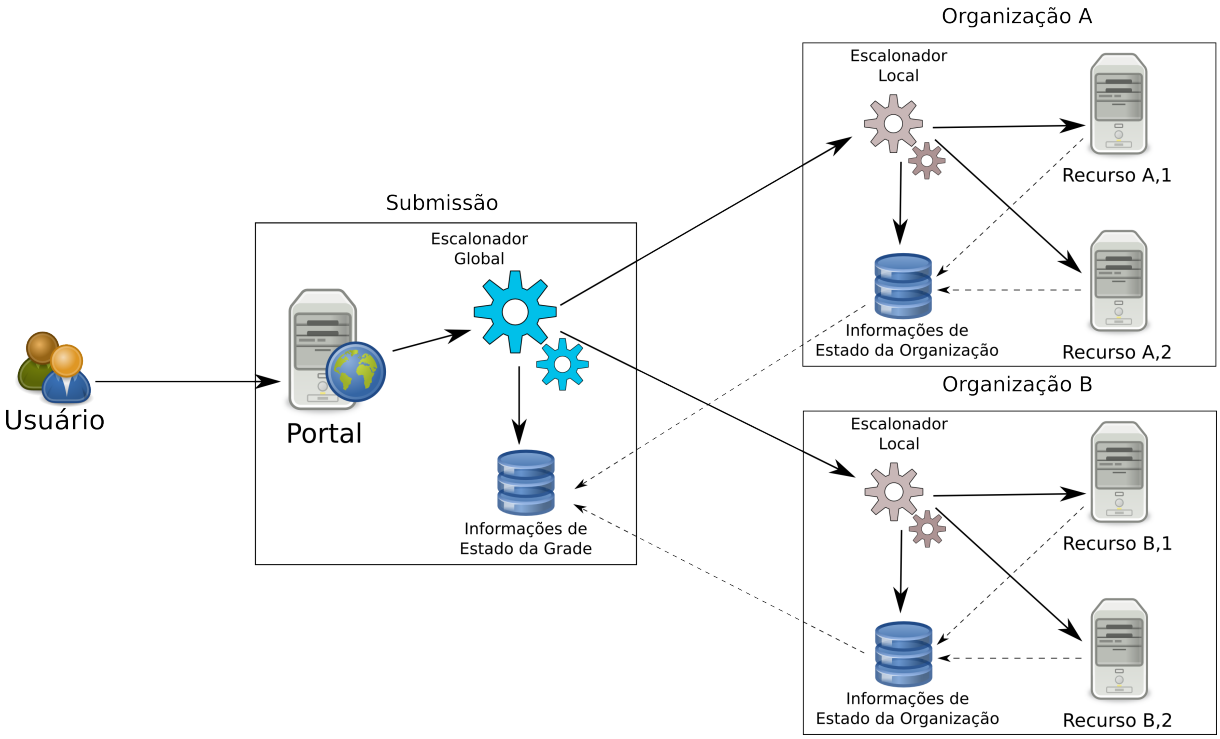

Figura 1.1: Modelo de funcionamento de uma grade para submissão de tarefas.

A figura 1.1 apresenta um modelo tradicional de arquitetura presente em vários projetos de grades computacionais acadêmicas (SINAPAD, 2010) (VCG, 2010). Aos usuários pesquisadores é fornecido um portal *web* no qual tarefas podem ser submetidas para a grade. As tarefas submetidas consistem de um arquivo executável e seus dados de entrada. O portal encaminha as requisições para a fila de um escalonador global da grade que tem ao seu dispor um repositório com informações sobre recursos de todas as organizações integrantes. Ele de-

cide para qual organização encaminhar a requisição, adicionando a tarefa na fila do escalonador local escolhido. Utilizando informações sobre os recursos locais, o escalonador local decide então qual recurso executa a tarefa.

Apesar das grades facilitarem a administração de recursos computacionais, o paradigma de submissão de tarefas apresenta problemas para os pesquisadores que desejam utilizar a grade para execução de aplicações científicas, o que é o foco deste trabalho. Por exemplo, como a criação de tarefas envolve a definição de arquivos executáveis, o usuário precisa considerar a compatibilidade entre o ambiente no qual a aplicação foi compilada e o sistema no qual será executada. Nas grades computacionais baseadas na submissão de tarefas temos a abstração de localização do recurso, do processo de escalonamento e da maioria das etapas de autenticação e autorização. Entretanto, o usuário pesquisador ainda precisa conhecer detalhes das configurações de pelo menos parte dos recursos para poder executar suas tarefas.

Por outro lado, com o advento de arquiteturas SOA[1] para aplicações distribuídas, as comunidades de serviços *web* e grades computacionais iniciaram um processo de convergência que resultou na criação da OGSA (*Open Grid Services Architecture*) (FOSTER et al., 2005). A OGSA descreve os principais serviços que as grades computacionais devem possuir e quais funcionalidades devem apresentar. Apesar das grades adotarem uma arquitetura orientada a serviços na sua infraestrutura, nem todas as aplicações em execução necessitam adotar a mesma arquitetura. A organização das grades em uma arquitetura SOA não significa o abandono do paradigma de submissão de tarefas na criação de aplicações. A necessidade de melhor manutenção e extensão do *middleware* é suficiente para justificar a adoção desse novo tipo de arquitetura (FOSTER, 2006). Na figura 1.1, por exemplo, os escalonadores podem ser representados como serviços, o que facilita modelar sua interação em termos de orquestração de serviços. Apesar disso, não se deve desconsiderar que a mesma tecnologia que serviu para reestruturar as grades possa ser utilizada para reformular as aplicações em execução nela.

Em vez de modelar as aplicações como um conjunto de tarefas submetidas à grade, pesquisadores podem definir uma orquestração de serviços que cumpre o mesmo papel. Orquestração de serviços é uma coordenação automática de serviços que atuam em conjunto para prover uma funcionalidade completa (PREIST, 2007). Os serviços utilizados para compor tal orquestração são disponibilizados pelas organizações que formam a grade, sendo que os usuários não precisam conhecer detalhes de sua implementação real. Configurações necessárias na submissão de tarefas, como versões de sistemas operacionais e compiladores, passam a ser

[1] *Service-oriented architecture*(SOA), pode ser traduzido como arquitetura orientada a serviços, e é um estilo de arquitetura de software cujo princípio fundamental prega que as funcionalidades implementadas pelas aplicações devem ser disponibilizadas na forma de serviços (OPEN GROUP, 2009).

desnecessárias, aumentando a abstração do sistema.

A utilização de *workflows* para representar orquestrações de serviços em grades computacionais proporciona ao usuário uma visão geral da sua aplicação, compreendendo o fluxo de controle e a troca de dados entre os serviços (GOBLE et al., 2003) (OINN et al., 2004) . *Workflows* científicos surgiram como um paradigma para descrever, gerenciar e compartilhar análises científicas complexas. Eles representam de maneira declarativa os componentes que precisam ser executados em uma aplicação científica complexa, assim como as dependências de dados entre esses componentes (GIL, 2008). *Workflows* também facilitam o compartilhamento de informações sobre processos científicos, já que seu desenvolvimento pode ser feito de maneira colaborativa por vários pesquisadores, aspecto que se encaixa no conceito de grades como entidades baseadas na colaboração.

Apesar das vantagens da utilização de *workflows* em grades, ainda persistem desafios que precisam ser contornados para atingir todo o potencial de abstração dos detalhes técnicos do sistema (GIL et al., 2007) (OINN et al., 2005). Por exemplo, uma questão relevante a ser tratada é o mapeamento do modelo de *workflow* criado pelo usuário para uma instância real em execução, visto que é preciso definir qual serviço concreto será invocado para cada componente do *workflow*, pois podem existir várias versões de um mesmo serviço implantadas na grade. O escalonamento adequado de *workflows* é crucial para a manutenção do sistema e a satisfação do usuário, sendo importante levar em consideração métricas de QoS[2] definidas pelo contrato de formação da grade e pelos usuários que submetem aplicações.

Outro problema relacionado com a execução de *workflows* diz respeito a inadequação da arquitetura cliente-servidor para essa finalidade (YAN; YANG; RAIKUNDALIA, 2006). Em geral, nessa arquitetura o esforço computacional não é balanceado, pois a carga destinada ao servidor é maior do que a atribuída ao cliente. Tal desproporção pode se tornar crítica caso não exista replicação suficiente de servidores, tornando os processos existentes gargalos do sistema. A escalabilidade do sistema também é afetada pelo desequilíbrio entre clientes e servidores. E por se tratar de um sistema que executa em grades computacionais, a sobrecarga de um determinado recurso para desempenhar o papel de servidor vai contra os fundamentos da arquitetura (FOSTER; KESSELMAN, 1999).

A execução *peer-to-peer* de *workflows* elimina a necessidade do escalonador central, distribui a tomada de decisões e evita o problema da existência de gargalos. Como nas redes *peer-to-peer* todo hospedeiro requisita ou fornece serviços aos pares, existe também a possibi-

[2] A qualidade do serviço (QoS) refere-se as características não funcionais pelas quais um serviço é avaliado por clientes ou outros interessados (MANI; NAGARAJAN, 2002).

lidade de implementar esquemas de distribuição de carga entre os hospedeiros que integram a rede. Redes *peer-to-peer* são infraestruturas de sistemas distribuídos caracterizadas pela descentralização das funções na rede, na qual cada hospedeiro realiza tanto funções de servidor quanto de cliente (TANENBAUM; STEEN, 2007). Redes P2P adotam a distribuição horizontal como paradigma arquitetural. Neste tipo de distribuição, um cliente ou servidor pode ser separado fisicamente em partes lógicas equivalentes, mas cada parte atua na sua própria partição do conjunto de dados, desta forma balanceado a carga (FOSTER; IAMNITCHI, 2003).

Soluções existentes como o *Ourgrid* (CIRNE; BRASILEIRO; PARANHOS, 2003) comprovam que grades e computação *peer-to-peer* têm muito em comum, sendo que o aspecto colaborativo de ambos os sistemas serve como motivação para a criação de infraestruturas robustas baseadas nos dois conceitos. Adicionando a essa combinação a abstração fornecida pelos *workflows*, o resultado são ambientes que podem ser capazes de atender as necessidades dos usuários com menor esforço administrativo e custo operacional.

1.2 Motivação

Na última década, diversos trabalhos emergiram para a execução *peer-to-peer* de *workflows* em grades computacionais acadêmicas, conforme a taxonomia apresentada em (KRAUTER; BUYYA; MAHESWARAN, 2002). No entanto, eles apresentam as limitações mencionadas a seguir que as impedem de atingir todo o potencial possível pelas tecnologias envolvidas.

A primeira limitação encontrada nas soluções existentes é a persistência de características centralizadas baseadas no paradigma cliente-servidor. O Triana (MATTHEW et al., 2003) é um ambiente de execução de *workflows* que fornece a opção de utilizar uma rede *peer-to-peer* para execução de etapas de um *workflow*. Por exemplo, em um *workflow* Triana, uma etapa pode consistir da invocação de um serviço em uma grade construída com o *Globus* (FOSTER, 2006), enquanto a próxima etapa é formada pela submissão de várias tarefas a uma grade *Ourgrid* (CIRNE; BRASILEIRO; PARANHOS, 2003). É possível utilizar várias tecnologias de grade diferentes, sendo essa sua principal vantagem. Entretanto, todo o processo de escalonamento continua centralizado em um servidor. Persistem assim os problemas relacionados a existência de gargalos e à falta de colaboração no planejamento da execução.

A segunda limitação encontrada é o suporte exclusivo ao paradigma de submissão de tarefas. A solução apresentada em (RANJAN; RAHMAN; BUYYA, 2008) é baseada em uma rede *peer-to-peer* estruturada e leva em consideração os requisitos de QoS do usuário na submissão de tarefas. A limitação ocorre porque tais requisitos são descritos em termos de

quantidade de memória, armazenamento em disco e ciclos de processamento necessários para completar uma tarefa. Por exemplo, um usuário pode descrever uma etapa de *workflow* como uma tarefa que necessita de 2 *gigabytes* de memória para execução, 40 *gigabytes* em disco e um processador de 3 *gigahertz* para execução. Para um usuário ter noção desses valores necessários com precisão, é necessário realizar uma análise de complexidade e estatística do código submetido na tarefa, algo que não é trivial. Nesse caso, a submissão de tarefas se torna menos transparente, o que dificulta a utilização da grade e diminui a abstração.

A terceira limitação comum é a ausência de avaliação dos requisitos de QoS informados pelos usuários no escalonamento de *workflows*. O SwinDew (YANG et al., 2007) utiliza uma rede *peer-to-peer* não estruturada para a execução descentralizada de *workflows*. O fator que guia a execução é o balanceamento de carga entre os hospedeiros, na verdade, um requisito desejável para a grade como um todo. Entretanto, o SwinDew não deixa margem para o usuário definir critérios em relação a requisitos tais como tempo de execução. A experiência final do usuário tende a ser insatisfatória, pois o mesmo não tem controle nenhum sobre a execução, mesmo quando as escolhas de hospedeiros diferentes do escolhido por ter a menor carga não prejudicariam o balanceamento de carga.

A quarta limitação é que mesmo as soluções descentralizadas como o SwinDew não apresentam o mesmo funcionamento adaptativo presente em outras redes *peer-to-peer* (LUA et al., 2005). Apesar de ser um processo de decisão distribuído, o *peer* que recebe a requisição do usuário considera informações provenientes de outros *peers* que são verdadeiras no momento da submissão. Por exemplo, podem ser escolhidos os *peers* como menor carga no momento da submissão. Entretanto, não há garantias que no decorrer da execução de um *workflow* o estado da grade não mudará, invalidando as decisões tomadas no momento da submissão.

As limitações citadas são empecilhos para a utilização eficiente das grades computacionais por pesquisadores que não pertencen à área de ciência da computação ou de sistemas distribuídos. Em grades computacionais nas quais o número de usuários é maior do que o de recursos disponíveis, um servidor centralizado sobrecarregado pode apresentar longos intervalos de resposta mesmo que os recursos da grade sejam máquinas com alto poder de processamento. Ambientes que não forneçam ao usuário as abstrações necessárias para a descrição das aplicações e exijam conhecimento detalhado sobre os recursos não são de fácil utilização. Além disso, usuários estão acostumados a redes *peer-to-peer* que possuem funcionalidades de configuração e manutenção automática. Nessas redes, após a submissão de requisições há garantias que a rede irá tentar oferecer a melhor qualidade de serviço revisando decisões já tomadas no momento da submissão, buscando adaptar-se ao ambiente em constante evoluçãom sem a ne-

cessidade de intervenção do usuário.

1.3 Objetivos e Contribuições

O objetivo desta dissertação é propor uma solução que permita aos administradores de grades computacionais científicas estabelecerem um ambiente para execução *peer-to-peer* de *workflows*. Por ambiente de execução entende-se uma aplicação que recebe do usuário uma estrutura de dados representando o *workflow* e se responsabiliza por gerenciar a execução das etapas do *workflow* na grade computacional. O ambiente proposto deve apresentar um alto nível de descentralização, fornecendo ao usuário a abstração da orquestração de serviços em *workflows* e possibilitar ao mesmo influenciar no escalonamento dos serviços ao informar prioridades entre as métricas de QoS. O foco desta dissertação são as aplicações científicas em execução nas grades computacionais.

As seguintes metas relacionadas a execução de *workflows* em grades computacionais fazem parte desta proposta:

- A definição de uma estratégia *peer-to-peer* que considere parâmetros de QoS para o escalonamento de *workflow* em grades computacionais. Essa estratégia se beneficia da arquitetura de escalonamento com alto nível de descentralização e realiza uma tomada de decisão local a cada etapa do *workflow*. O objetivo é maximizar o cumprimento dos requisitos de QoS considerando informações recentes do estado da grade na avaliação de cada etapa.

- O projeto e implementação de uma arquitetura para a execução *peer-to-peer* de *workflows* baseados na orquestração de serviços desenvolvida utilizando os paradigmas do *Globus Toolkit* e do WSRF (*Web Service Resource Framework*) (FOSTER, 2006). A implementação inclui um critério para formação de grupos de *peers* além de algoritmos para manutenção dos mesmos.

- A análise comparativa da proposta em relação a soluções já existentes. Para efeito de comparação, na análise foi escolhida uma técnica de escalonamento centralizado tradicional baseada em algoritmos genéticos e outra heurística *peer-to-peer* que apresenta alto nível de descentralização e foi implementada pelos desenvolvedores do SwinDew.

A contribuição principal desta dissertação é um ambiente de execução adequado as necessidades da pesquisa científica moderna que faz uso de computação de alto desempenho. Este

ambiente inclui aprimoramentos, os quais são comprovados pela análise, em relação à soluções existentes nos quesitos de descentralização e requisitos de QoS dos usuários pesquisadores.

1.4 Estrutura

O restante deste trabalho se organiza da seguinte maneira:

- no capítulo 2, a fundamentação teórica sobre grades computacionais é descrita;
- o capítulo 3 apresenta a problemática da execução *peer-to-peer* de *workflows* em grades científicas e a análise de soluções existentes;
- no capítulo 4, a solução proposta é descrita em detalhes;
- o capítulo 5 traz uma análise comparativa da proposta em relação a outras soluções existentes e,
- o capítulo 6 detalha as conclusões do trabalho realizado, assim como trabalhos futuros importantes.

2 Grades Computacionais

Neste capítulo são apresentados os conceitos teóricos relacionados às grades computacionais. Uma visão geral das grades é discutida e em seguida o estudo é aprofundado para a arquitetura e tecnologias adotadas como referências nesta dissertação.

A seção 2.1 apresenta os componentes presentes nas grades computacionais. Os tipos de grades computacionais são discutidos na seção 2.2, na qual duas classificações são apresentadas, a primeira baseada no tipo de aplicação e a segunda no tipo de serviço disponibilizado. As métricas de qualidade de serviços para grades computacionais são enumeradas na seção 2.3, como foco nas grades de serviços. Dois exemplos com características distintas de grades computacionais são tema da seção 2.4. A arquitetura para grades de serviços OGSA é explicada na seção 2.5, com ênfase no paradigma de programação adotado pela arquitetura, o *Web Service Resource Framework* (WSRF).

2.1 Arquitetura Geral das Grades

Grades computacionais são definidas como um conjunto de aplicações e políticas que permitem o compartilhamento coordenado de recursos computacionais entre organizações (FOSTER; KESSELMAN, 1999). O grupo de entidades participantes de uma grade é chamado de Organização Virtual (FOSTER; KESSELMAN, 1999), pois é formado por integrantes pertencentes a organizações distintas que firmam acordos, em geral, momentâneos, visando a resolução de um determinado problema comum de forma rápida e menos onerosa. A grade é, portanto, um ambiente que se estende através de vários domínios administrativos, cada um contendo suas próprias políticas.

A definição de recurso computacional é ampla o suficiente para englobar um vasto conjunto de dispositivos. Qualquer instrumento que possua uma interface de controle conectada a um computador em rede pode ser considerado um recurso computacional em uma grade e até mesmo software entra nessa classificação. Tal definição, apesar de parecer pouco restritiva, é

interessante pelas possibilidades que são abertas. Supercomputadores, câmeras digitais, satélites e estruturas de armazenamento são apenas alguns exemplos de recursos. Como nas relações existentes entre organizações reais, as interações das organizações virtuais possuem uma ordem econômica. Aquele que cede acesso à um recurso espera receber algo em troca. A recompensa não precisa ser em valor monetário, é comum em grades a existência de outras moedas de troca, tais como tempo de acesso liberado à outros recursos.

Existe uma clara distinção entre os acordos firmados que definem uma grade e a tecnologia que serve para implementação. Por exemplo, uma grade pode ser implementada por aplicativos e serviços que utilizam apenas *sockets* TCP/IP para programação. Entretanto, essa alternativa não é a mais correta, pois um sistema complexo como uma grade não é ideal para programação de tão baixo nível, sendo que o código resultante pode apresentar diversas características indesejáveis como a dificuldade de manutenção. Existem atualmente *middlewares* e bibliotecas de alto nível que facilitam o trabalho do desenvolvedor que encara o desafio de implantar uma grade (FOSTER, 2006) (CIRNE et al., 2006).

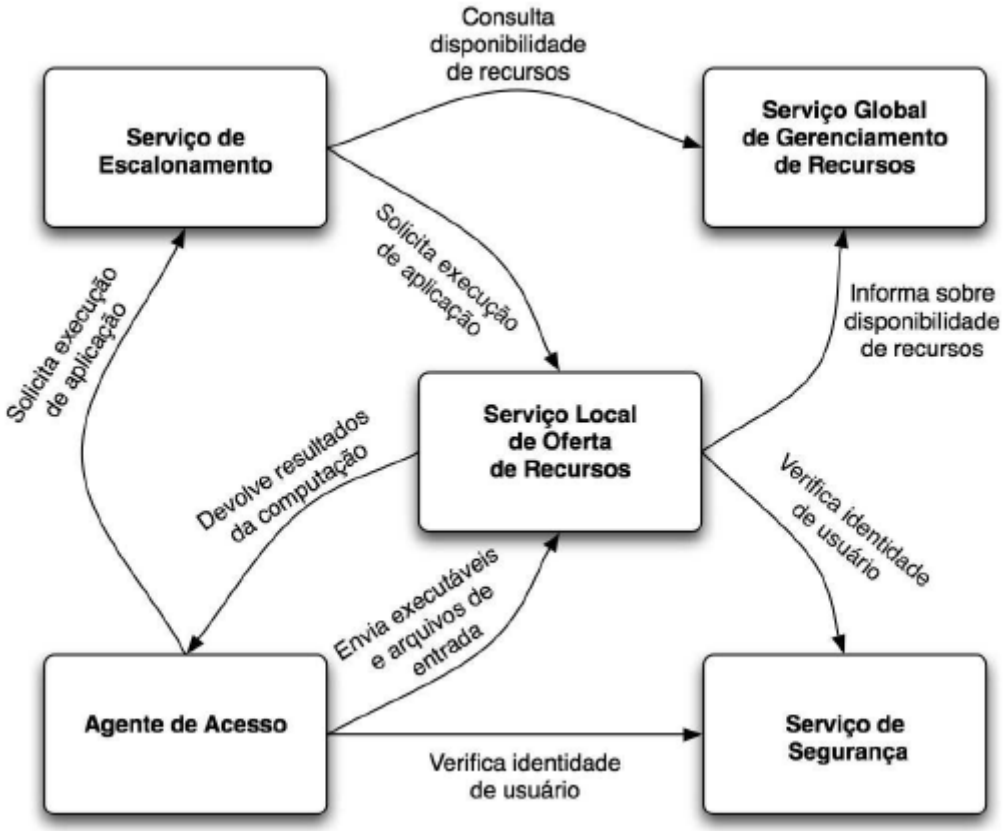

Figura 2.1: Modelo genérico de uma grade computacional (KON; GOLDMAN, 2008).

As arquiteturas adotadas pelos *middlewares* de grades são diversificadas. Cada solução costuma adicionar ou remover componentes de acordo com a necessidade da aplicação. Entre-

tanto, é possível definir elementos presentes na maioria dos casos, apresentados a seguir (KON; GOLDMAN, 2008):

- O agente de acesso é a interface de acesso do usuário final ao ambiente de grade. Em geral, oferece a funcionalidade de verificação de identidade além de permitir configurar a execução de aplicações. A tendência atual é oferecer um portal *web* representando tal componente.
- O serviço de escalonamento garante que o acesso aos recursos seja organizado de maneira coerente com a capacidade de utilização, sendo o papel deste componente semelhante ao escalonador de sistemas operacionais.
- O serviço local de oferta de recursos é um serviço executado diretamente no recurso computacional compartilhado, sendo responsável por informar aos outros componentes da grade o estado atual do recurso. Tal estado consiste dos dados necessários para a execução correta de aplicações que utilizem o recurso.
- O serviço global de gerenciamento de recursos é responsável por monitorar o estado dos recursos compartilhados em toda a grade e por responder a solicitações de utilização desses recursos.
- O serviço de segurança é responsável por limitar o acesso aos recursos apenas a usuários autorizados.

Além desses cinco elementos, algumas grades também implementam um sistema de criação de réplicas e um serviço de transferência de dados em alto desempenho (FOSTER, 2006). A organização dos serviços é representada na figura 2.1. Na literatura, encontra-se exemplos dos serviços citados implementados tanto de maneira centralizada (BERNHOLDT et al., 2005) ou distribuída (ARAUJO et al., 2005), de acordo com as necessidades de cada grupo de pesquisa.

2.2 Tipos de Grades Computacionais

As grades computacionais podem ser classificadas de acordo com o tipo de aplicação executada na plataforma (KRAUTER; BUYYA; MAHESWARAN, 2002)(KON; GOLDMAN, 2008). Tal critério estabelece a seguinte classificação:

- Grade computacional de alto desempenho é um sistema de Computação em Grade que visa a integração de recursos computacionais dispersos para prover uma maior capacidade combinada de processamento aos seus usuários.

- Grades de dados são sistemas cujo objetivo principal é o acesso, pesquisa e processamento de grandes volumes de dados, potencialmente distribuídos em vários repositórios, conectados por uma rede de grande área.

- Grades de serviços são sistemas que oferecem serviços viabilizados pela integração de diversos recursos computacionais como, por exemplo, um ambiente para trabalho colaborativo para execução de aplicações de e-Ciência.

Apesar de facilitar o estudo de sistemas já existentes, essa classificação informa pouco sobre a estrutura interna da grade. Outra classificação vigente divide grades em grades de serviços e grades de alto desempenho (CIRNE; SANTOS-NETO, 2005). Grade de serviços é uma infraestrutura que viabiliza servicos sob demanda, permitindo uma maior colaboração entre várias instituições, através do compartilhamento de seus serviços e recursos, e utilizando mecanismos que facilitem a interoperabilidade. O foco desse tipo de grade é a definição de aplicações através da composição de serviços sob demanda. Uma grade de alto desempenho é uma grade de serviços que contém um serviço de execução remota, permitindo a execução de tarefas que compõem aplicações paralelas. Essas definições não são exclusivas, sendo que qualquer grade de serviço pode ser classificada como grade de alto desempenho desde que possua um serviço de execução remota ou forneça serviços que possam ser compostos em uma aplicação de alto desempenho.

Observa-se que as grades de serviço costumam possuir um domínio de aplicação específico, com um conjunto finito de serviços oferecidos que não se altera com frequência (BERNHOLDT et al., 2005). O usuário final direciona a composição de serviços e os dados de entrada, sendo que o código a ser executado já se encontra encapsulado nos serviços da grade. As grades de alto desempenho permitem que o usuário forneça o código a ser executado, portanto podem atendar a diversos domínios de aplicação. É importante notar que a existência de um serviço de execução remota traz vários desafios, em especial na segurança da grade (e.g, o código do usuário é seguro?) e no escalonamento (e.g, como prever o desempenho do código do usuário?).

Uma terceira e última classificação considera a maneira com que as grades tratam a qualidade de serviço. A tabela 2.2 apresenta essa classificação (YU; BUYYA, 2006a).

Tabela 2.1: Grades Comunitárias e Grades de Utilidade (YU; BUYYA, 2006a)

	Grades Comunitárias	**Grades de Utilidade**
Disponibilidade	Melhor Esforço (*Best Effort*)	Reserva de Recursos
QoS	Melhor Esforço (*Best Effort*)	Contratos
Custo	Acesso Livre	Uso e Nível de QoS

Em grades de utilidade, os usuários podem fazer reservas com um provedor de serviços para garantir a disponibilidade dos serviços desejados. Tal reserva é realizada através da negociação de requisitos de funcionalidade e qualidade de serviço. Comparada com as grades de utilidade, a disponibilidade e qualidade de serviço em grades comunitárias não podem ser garantidas. Entretanto, as grades comunitárias fornecem acesso livre baseado em acordos mútuos, enquanto em grades utilitárias o usuário deve pagar pelo acesso. Essa classificação é importante no contexto deste trabalho, pois a solução proposta lida com a avaliação de parâmetros de QoS. No capítulo 4, o ambiente proposto é situado dentre as classificações apresentadas nesta seção.

2.3 Métricas de Qualidade de Serviço para Grades Computacionais

A qualidade do serviço (QoS) refere-se as características não funcionais pelas quais um serviço é avaliado por clientes ou outros interessados (MANI; NAGARAJAN, 2002). Existem outras interpretações para o termo QoS no estudo de redes de computadores (MORAIS; CARDOSO; GOMES, 2008), mas para os serviços de uma grade computacional, pode-se considerar dois aspectos de QoS.

O primeiro aspecto seria a qualidade de serviço que é fornecida para o usuário final. Essa aspecto pode ser avaliado pelo grau de satisfação que o usuário adquire ao executar suas aplicações na plataforma. Por exemplo, o tempo que o sistema leva para atender uma requisição é importante para o usuário.

O segundo aspecto considera do ponto de vista administrativo da grade. Como trata-se de um sistema colaborativo baseado em um acordo político entre as organizações integrantes, a qualidade de serviço da grade no ponto de vista administrativo analisa como o sistema distribui as requisições de forma justa entre todos os participantes. Por exemplo, um balanceamento de carga justo entre as máquinas de vários domínios diferentes é importante para os administradores da grade, pois evita que uma organização seja sobrecarregada.

Em relação aos requisitos de QoS importantes para o usuário final, os mais relevantes relacionados à serviços de grades computacionais são : duração da execução, custo da execu-

ção, reputação, confiabilidade, disponibilidade e segurança (YU; BUYYA, 2006b) (LUDWIG; REYHANI, 2007). O significado de cada um deles é descrito a seguir:

- A **duração da execução** de um serviço mede o tempo gasto entre a submissão por parte do usuário e o retorno do resultado final. Esse intervalo leva em consideração o tempo necessário para a transferência de dados entre os *peers* e o intervalo necessário para a invocação de cada componente do *workflow*. Em situações nas quais o tempo de transferência de dados é de grandeza muito menor do que o tempo de execução, a duração da execução pode ser considerada apenas como o tempo necessário para a invocação. Por exemplo, em uma aplicação onde as invocações demoram dias mas geram apenas dezenas de *megabytes* de dados, o tempo de transferência pode ser desconsiderado.

- O **custo da execução** se refere aos créditos debitados do cliente para a realização de cada invocação. Em grades computacionais, os créditos podem ser encarados de duas maneiras diferentes. Em grades comunitárias, tais créditos são usados para estabelecer sistemas de troca de favores, não tendo valor monetário real. Já nas chamadas grades de utilidade (*Utility Grids*) os créditos possuem relação com valores monetários reais ou outras riquezas concretas.

- **Reputação** de um serviço refere-se ao grau de certeza que o cliente tem de que o serviço retornará um resultado correto. Usuários diferentes podem ter opiniões conflitantes sobre a reputação de um mesmo serviço. Existem trabalhos que tratam do estabelecimento de reputação entre *peers* em grades computacionais (MARTINS et al., 2010). A estratégia mais utilizada é realização de testes de avaliação em *peers* da grade cujos resultados são objeto de análise estatística.

- **Confiabilidade** é a propriedade que um sistema possui de funcionar de maneira contínua sem falha (TANENBAUM; STEEN, 2007). É importante salientar que confiabilidade está relacionada a quantidade de falhas que um serviço pode apresentar. Sistemas pouco confiáveis são aqueles em que uma requisição tem pouca probabilidade de ser respondida corretamente em tempo hábil. Já a disponibilidade mede a probabilidade de um serviço estar disponível em determinado momento. Ao contrário da confiabilidade, a disponibilidade não apenas considera os intervalos em que o serviço está inacessível por falhas, mas também o tempo necessário para atividades de manutenção, por exemplo.

- **Segurança** se refere à confidencialidade da execução de requisições aos serviços da grade e a qualidade dos resultados. Trata-se de uma métrica complexa, sendo que possui relacionamentos com outras métricas como confiabilidade e reputação. As restrições impostas

pelos usuários podem ser variadas. Por exemplo, um usuário pode não se importar com a confidencialidade dos dados na execução de uma requisição, exigindo saber com certeza que apenas recursos autenticados na grade tomaram parte da execução. Em outro caso, tanto confidencialidade e autenticação podem ser exigidas. Ao contrário das outras métricas, segurança tem um caráter seletivo mais restrito, pois não há como um serviço cumprir parcialmente restrições de confidencialidade, por exemplo.

O cálculo de valores quantitativos para as métricas do usuário final citadas não é trivial em todos os casos. Por exemplo, enquanto o tempo de execução pode ser medido em quantidade de horas de execução, para a métrica de segurança não existem técnicas para quantificar em valores numéricos o grau de segurança. Existem trabalhos, tais como (FROTA, 2008), que abordam tal problema para a métrica de segurança. Entretanto, não é o objetivo deste trabalho fornecer uma técnica de mapeamento para todas as métricas existentes na literatura. Para a análise do ambiente proposto, as métricas escolhidas possuem valores quantitativos obtidos através de técnicas simples já encontradas na literatura (MASSIE; CHUN; CULLER, 2004). Maiores detalhes sobre a escolha dessas métricas estão no capítulo 5.

Os administradores de grades computacionais também possuem requisitos de QoS que devem ser considerados. A **distribuição de carga** entre os hospedeiros que fazem parte da grade é um objetivo importante de ser atingido, pois o acordo que forma a grade deve ser vantajoso para todos os participantes. Outro requisito importante é a manutenção do **equilíbrio de favores**, sendo que o número de favores corresponde a quantidade de requisições externas atendidas por domínio administrativo. Por exemplo, considere uma grade formada por duas organizações A e B, sendo que o domínio de A possui duas vezes mais máquinas do que B, todas com mesma capacidade. Se o número de requisições oriundas dos usuários associados a B for maior que o número gerado por A, podemos chegar em uma situação na qual há balanceamento de carga entre os hospedeiros, porém o equilíbrio de favores está desfavorável a instituição A. O equilíbrio de favores não deixa de ser um balanceamento de carga, sendo que o foco não são os hospedeiros, mas sim o equilíbrio entre as instituições.

2.4 Exemplos de Grades Computacionais

A seguir, uma descrição de dois projetos de grades computacionais. Ambos utilizam técnicas de comunicação e políticas diferentes. O primeiro, *Ourgrid*(CIRNE et al., 2006), é um sistema *peer-to-peer* baseado numa política de troca de favores que utiliza Java RMI para a comunicação tem o foco em fornecer uma grade de alto desempenho. O segundo, é o *Earth*

System Grid(BERNHOLDT et al., 2005), uma grade para simulação climática criada utilizando o *Globus Toolkit*, baseada no conceito de grade de serviços.

2.4.0.1 *Ourgrid*

A *Ourgrid* é uma grade cooperativa, aberta e de livre acesso na qual laboratórios disponibilizam seus recursos computacionais ociosos em troca de acesso a recursos de outros laboratórios quando estes estiverem ociosos(CIRNE et al., 2006). Os desenvolvedores desta ferramenta levaram em consideração que a maioria dos pesquisadores possuem recursos computacionais que apresentam períodos de ociosidade consideráveis, já que a própria atividade de pesquisa possui um momento de coleta ou geração de dados (no qual existe muito processamento) e um intervalo no qual há a análise dos dados obtidos (no qual o processamento é pouco ou inexistente).

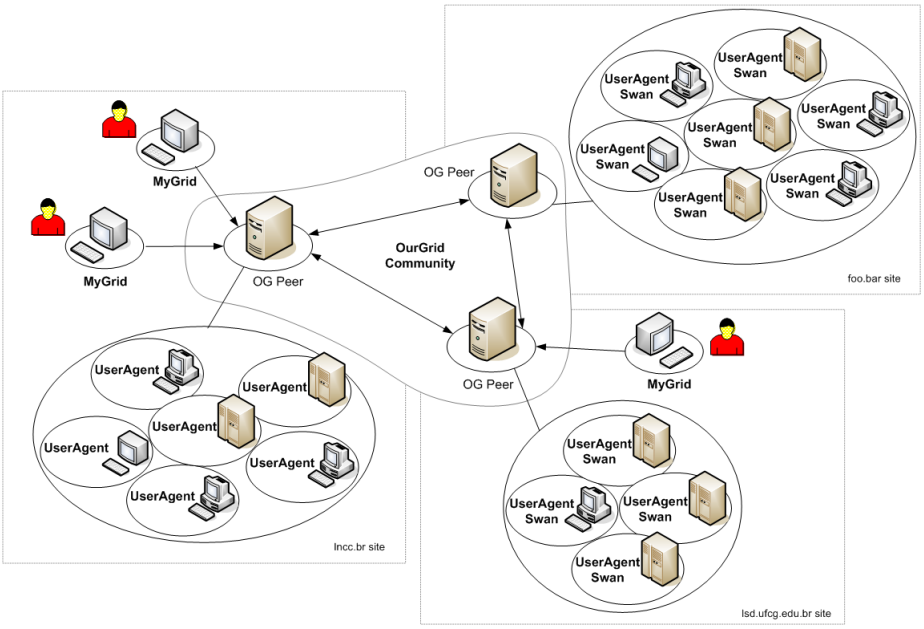

Figura 2.2: Arquitetura do *Ourgrid* (CIRNE et al., 2006).

Os criadores do *Ourgrid* estabeleceram que para obter sucesso, a *Ourgrid* deveria manter quatro características básicas: simplicidade, rapidez, escalabilidade e segurança. A rapidez desejada é aquela que permita a tarefas executadas na grade um tempo de execução menor que o obtido caso ela fosse executada localmente. A simplicidade deriva do fato de que a maioria dos pesquisadores não são cientistas da computação, sendo que sistemas complexos e de difícil utilização podem afastar tais pesquisadores e seus laboratórios da grade. A escalabilidade é um requisito comum à maioria dos sistemas distribuídos, que por sua natureza tendem a ser espalhar em diversos componentes. E por último, a segurança surge como uma necessidade tanto para o

usuário que deseja manter suas tarefas confidenciais, quanto para o proprietário do recurso que deseja protegê-lo de usuários maliciosos. A arquitetura desta solução é mostrada na figura 2.2 (CIRNE et al., 2006).

O compartilhamento de recursos é governado por regras inspiradas em sistemas *peer-to-peer* tradicionais. Cada laboratório que fornece recursos é representado por um *peer*. O único requisito existente para se tornar um *peer* é um endereço IP válido com uma porta TCP não bloqueada por *firewall*. A cada *peer*, são relacionadas máquinas no domínio que possuam instalado o serviço *useragent*. Assim, o *peer* é responsável por contabilizar o uso de suas máquinas *useragent* por usuários de outros domínios e também quantos recursos externos foram utilizados por usuários locais. Uma rede de favores é criada, na qual cada *peer* sabe como seus recursos locais e externos foram utilizados. De acordo com tal informação, o *peer* pode balancear o fornecimento de tempo computacional quando as requisições externas ocorrerem.

Como exemplo de funcionamento da rede de favores podemos imaginar uma situação em que um *peer* A recebe requisições simultâneas de dois outros *peers* B e C. Inicialmente, A verifica entre B e C qual deles forneceu mais recursos anteriormente para os usuários no mesmo domínio de A. O primeiro a ser atendido será aquele que tiver oferecido mais recursos. Após a troca de favores, cada *peer* envolvido na relação atualiza suas tabelas internas. Numa situação em que apenas um *peer* requisite recursos, eles são fornecidos independentemente das relações anteriores entre o pares, para que *peers* novatos também tenha chance de entrar na rede. A rede de favores simplifica em muito a contabilização de uso dos recursos, por evitar a necessidade de uma entidade central e limitar o dano de freeriders, *peers* que apenas consumem sem nunca fornecer.

Para manter a simplicidade, até o presente momento o *Ourgrid* só lida com aplicações *Bag-of-Tasks* (BoT). Este tipo de aplicação é aquele que cujas tarefas são independentes. Existem várias aplicações que se enquadram nesta classificação, por exemplo, *data mining* e biologia computacional. Os usuários em um laboratórios submetem tais aplicações através do *Mygrid*, um escalonador que requisita máquinas aos *peers* e monitora a execução. Ele assim assume o papel de escalonador de requisições, porém é importante notar que ele não se constitui de um ator global com informações precisas sobre o estado das máquinas participantes. Sem esta visão geral, é possível que o escalonador envie uma tarefa para uma máquina que já apresentou dificuldades para finalizar requisições submetidas por outros escalonadores.

Para tentar minimizar os danos causados pela falta de uma visão global, o *Mygrid* utiliza uma heurística chamada *Work Queue With Replication* (WQR). Tal heurística envia randomicamente determinada tarefa para uma máquina do grupo que o *Mygrid* recebeu depois da

requisição. Quando todas as tarefas já foram alocadas à máquinas, um das tarefas em execução é replicada. Desta forma, mesmo que a tarefa original falhe, alguma das réplicas pode funcionar corretamente. Quando uma tarefa termina corretamente, suas réplicas são descartadas imediatamente. Não é difícil perceber que para garantir um número maior de resultados corretos é gasto um tempo de processamento maior devido à criação de réplicas. Entretanto, além de ser possível configurar o número máximo de réplicas para cada tarefa, estudos demonstram que a queda de desempenho da grade somente é relevante quando o número de tarefas é da mesma ordem de magnitude do número de máquinas(CIRNE; BRASILEIRO; PARANHOS, 2003).

Existe também a heurística *Storage Affinity*, que possui o funcionamento básico da WQR, porém considerando dados sobre a distribuição dos arquivos de entrada utilizados pelas tarefas. As tarefas são alocadas de acordo com uma métrica que determina quantos bytes da entrada a ser utilizada estão presentes em cada máquina disponível. Tal informação é mais fácil de ser adquirida do que a carga da CPU ou a quantidade de memória disponível, pois não se altera regulamente.

O sistema de segurança do *Ourgrid* é bastante simples em termos de autorização. Qualquer laboratório pode entrar na grade, porém ele estará sujeito às regras da rede de favores. O esforço necessário para que um laboratório mal intencionado prejudique a grade é considerável, já que para executar tarefas em máquinas remotas ele também terá que fornecer recursos. Outro aspecto considerado é a confidencialidade da aplicação em execução remota. Atualmente, o *Ourgrid* não fornece uma solução que impeça o administrador de um domínio de visualizar o conteúdo das tarefas externas executando em suas máquinas. Na prática a aplicação se distribui de tal forma através da grade que seria necessário um esforço conjunto de boa parte dos integrantes para que a confidencialidade entrasse em risco. A proteção do laboratório em relação à tarefas nocivas é fornecida através da utilização de máquinas virtuais. Neste mecanismo, as tarefas não executam diretamente sobre o sistema operacional hospedeiro, mas sim encapsuladas em uma máquina virtual que controla o acesso aos recursos locais, possibilitando a criação de diversas políticas de acesso.

A publicação de recursos é feita automaticamente. No momento em que um *peer* se torna ativo, suas máquinas *useragents* são publicadas na página http://status.ourgrid.org. Neste sítio, um usuário pode descobrir informações importantes sobre os recursos, tais como qual sistema operacional está instalado nas máquinas de determinado laboratório, quais máquinas estão ociosas, entre outras.

Na atualidade, além de ter sido usada com sucesso para criar uma grade de processamento genérico, a ferramenta *Ourgrid* também é utilizadas na criação de grades específicas

a certas áreas de conhecimento. Um dos projetos derivados de maior sucesso é o SegHidro (ARAUJO et al., 2005), que visa utilizar aplicações em grade para otimizar a utilização de recursos hídricos naturais.

2.4.0.2 *Earth System Grid*

O *Earth System Grid* (ESG) (BERNHOLDT et al., 2005) é uma grade que se estende por vários laboratórios norte-americanos que visa facilitar o acesso à dados gerados por simulações metereológicas. Até o ano de 2005, a quantidade de informações coletadas já ultrapassavam a quantia de 100 terabytes. O ESG utiliza serviços criados utilizando os componentes do Globus Toolkit.

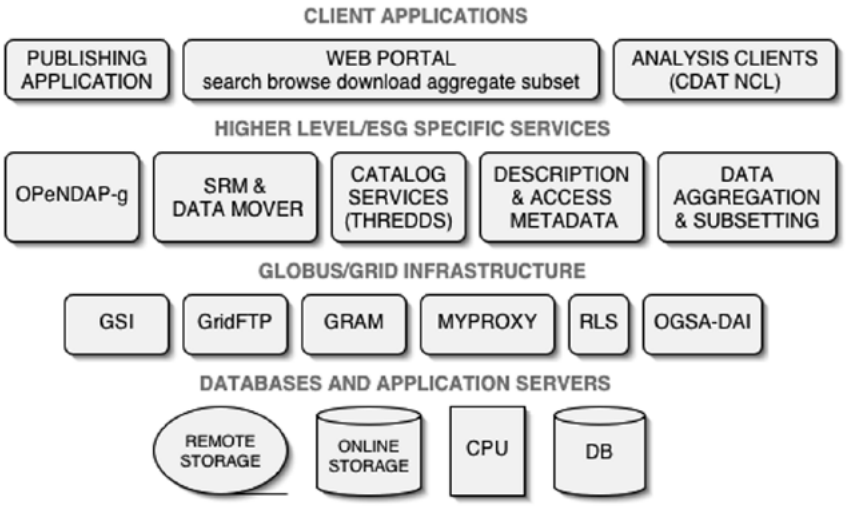

Figura 2.3: Arquitetura do *Earth System Grid* (BERNHOLDT et al., 2005).

Apesar de ter uma estrutura composta por vários atores, como demonstrado na figura 2.3 (BERNHOLDT et al., 2005), para o usuário comum o ESG é de utilização relativamente fácil, já que a maioria dos participantes não submetem tarefas, apenas acessam os dados através de serviços especializados. Os modelos climáticos a serem executados são bastante conhecidos e o número de laboratórios que fazem parte da grade é praticamente constante, sendo que estas duas características tornam o sistema de escalonamento simples. Um mesmo serviço de simulação produz uma quantidade de dados considerável que é de interesse de diversos usuários. Por exemplo, um grupo pode estar interessado em informações sobre as camadas mais altas da atmosfera, enquanto outro laboratório pode se interessar pelas condições dos mares, porém ambos acessam dados fornecidos por uma mesma simulação. Resta ao usuário saber como descrever os dados que procura dentro da enorme base de informações.

Um portal foi criado para facilitar a utilização pelos usuários. Através dele, um partici-

pante pode criar os chamados conjunto de dados virtuais (*Virtual Datasets*), que são abstrações criadas à partir de vários conjuntos dispersos de dados físicos. Este conceito é semelhante ao de visões em bancos de dados relacionais. O portal também permite uma descoberta de recursos transparente, já o usuário não precisa saber onde se encontram os dados.

A infraestrutura do Globus Toolkit se encontra logo acima dos recursos físicos. É através dela que são criados uma variedade de serviços utilizados diretamente pelo portal. Estes podem ser divididos entre serviços de manipulação e de catálogo. Os serviços de manipulação são o *Open-source Project for a Network Data Access Protocol for Grid* (OpenNDAP-g) e o *Storage Resource Manager* (SRM). Como o próprio nome já diz, eles são responsáveis pela transferência e alterações dos dados. Os serviços de catálogo são os responsáveis pela criação de metadados para facilitar a localização e criação de *Virtual Datasets*. São grande em número, sendo que alguns estão generalizados na figura 2.3 (BERNHOLDT et al., 2005).

A segurança é garantida pela *Globus Security Infrastructure* (GSI). Tal infraestrutura é capaz de garantir autenticação segura, única e mútua entre recursos e usuários. Também garante a confidencialidade dos dados através de criptografia de chave pública e delegação de privilégios utilizando *proxies*.

2.5 *The Open Grid Services Architecture*

Na seção 2.1 foram apresentados os componentes geralmente encontrados em grades computacionais. Na atualidade existem diversos outros atores que fazem parte do universo das grades, em especial naquelas classificadas como grades de serviços. Somente para manipulação de dados, podem existir serviços de replicação, transferência segura, transformação, entre outros. Pelo próprio conceito de grades, os elementos que implementam funcionalidades essenciais para o sistema costumam se espalhar por diversas máquinas interconectadas, além de estabelecerem uma comunicação constante entre si. Por exemplo, um escalonador necessita consultar a descoberta de recursos para localizar máquinas capazes de executar cada *job* submetido assim como precisa de informações do sistema de segurança para averiguar as permissões do usuário que submeteu a tarefa.

Devido a sua complexidade, sistemas de grades computacionais desenvolvidos de maneira independente apresentam problemas de interoperabilidade. A padronização permite aos desenvolvedores construir soluções com a garantia de um nível mínimo de interoperabilidade. A busca por essa garantia levou a criação da *Open Grid Services Architecture* (OGSA) (FOSTER et al., 2005), que define uma arquitetura comum, padronizada e aberta para aplicações em

grade. O objetivo da OGSA é padronizar praticamente todos os serviços geralmente encontrados em grades computacionais definindo uma interface padrão para cada um deles. Até 2009, a criação das interfaces ainda não foi concluída, mas a OGSA já é capaz de definir os serviços mais importantes que devem estar presentes nas grades.

2.5.1 Requisitos da OGSA

A OGSA foi criada a partir de um estudo de várias aplicações de grade computacionais. Uma elicitação completa de requisitos seguindo as boas práticas da engenharia de *software* moderna não foi realizada, porém as aplicações existentes serviram como entrada para o processo de definição da arquitetura. Os requisitos funcionais e não funcionais relevantes foram então organizados nas seguintes categorias: interoperabilidade, compartilhamento de recursos, otimização, qualidade de serviço, execução de tarefas, serviços de dados, segurança, redução de custo, escalabilidade, disponibilidade e facilidade de uso.

Na questão da interoperabilidade, a OGSA deve permitir a interoperação entre recursos e serviços diversos, distribuídos e heterogêneos assim como reduzir a complexidade de administrar tais sistemas heterogêneos. A virtualização de recursos utilizando interfaces de gerência e protocolos padronizados tem importante papel para permitir a interoperabilidade e a descoberta e busca de recursos.

O compartilhamento de recursos exige que mecanismos capazes de fornecer um contexto comum que permita relacionar usuários, requisições, recursos, políticas e acordos através dos limites das organizações. Um espaço de nomes global é imprescindível para o entendimento entre as partes, sendo que serviços de meta-dados realizam um papel auxiliar importante.

Otimização refere-se à técnicas utilizadas para alocar recursos de acordo com os requisitos do fornecidos pelo cliente. Nas grades computacionais, um requisito importante é a habilidade de ajustar dinamicamente prioridades de carga de trabalho (*workload*) para cumprir os objetivos gerais dos serviços. Mecanismos para documentar a utilização de recursos e alterar o esquema de alocação são os fundamentos para a otimização na grade.

A gerência de qualidade de serviço nas grades computacionais deve incluir mecanismos para gerenciar acordos entre os clientes e provedores de serviços são necessários, assim como sistemas para monitorar a qualidade do serviço. Migração dinâmica de serviços e recursos é importante para garantir acordos de qualidade de serviço.

A OGSA deve fornecer gerência para a execução de requisições de usuários durante seu ciclo de vida. Funções como escalonamento, controle de requisições e tratamento de exce-

ções devem ser suportadas mesmo quando a requisição estiver distribuída em um grande número de recursos heterogêneos.

Os serviços de dados da OGSA devem simplificar a criação de aplicações orientadas a dados adaptáveis a mudanças no ambiente. Além de fornecer políticas para o acesso seguro aos dados, interfaces comuns para o armazenamento e acesso coordenado são necessários. Um serviço de transferência de alto desempenho além de um mecanismo para a criação e manutenção de réplicas são outros requisitos importantes.

Mecanismos de autenticação são necessários para estabelecer a identidade de usuários e serviços. Fornecedores de serviços devem implementar mecanismos de autorização para reforçar políticas de acesso aos serviços. A OGSA precisa funcionar com infraestrutura de segurança diversas e permitir a delegação de credenciais de segurança do cliente do serviço para o provedor.

Para reduzir o custo de manutenção das grades, a OGSA deve permitir a automação das atividades de gerência. Gerenciamento baseado em políticas é um requisito para a automação das grades, já que as operações obedecem os objetivos do consórcio de organizações que opera a grade. Gerenciamento de conteúdo de aplicações pode facilitar a entrega, configuração e manutenção de sistemas complexos ao permitir que toda a informação local relacionada a uma aplicação seja especificada e controlada como uma única unidade lógica.

A escalabilidade pode aumentar o valor da grade ao permitir que o tempo de execução das tarefas seja reduzido pelo aumento de recursos disponíveis. A gerência de arquitetura precisa ser feita de uma maneira hierárquica ou *peer-to-peer*, usando técnicas colaborativas. Mecanismos de alta vazão de processamento são necessários para ajustar e otimizar a execução de tarefas paralelas.

Em um ambiente complexo como uma grade computacional, controle autônomo baseado em políticas é chave para criar sistemas de alta flexibilidade e poder de recuperação. Para ter um alto grau de disponibilidade, a OGSA deve fornecer mecanismos de recuperação e gerência de falhas em seus principais serviços, além de um serviço de diagnóstico e documentação das falhas.

O usuário deve ser capaz de utilizar a OGSA para esconder a complexidade do ambiente, se necessário. É crucial para facilitar o uso do sistema que ferramentas atuando em conjunto com serviços em tempo de execução sejam capazes de gerenciar boa parte do sistema fornecendo ao usuário abstrações no nível desejado.

2.5.2 Capacidades da OGSA

Nesta seção é apresentada uma visão das capacidades específicas necessárias para atender os requisitos levantados na seção anterior. Uma descrição mais detalhada foge do escopo desse trabalho e pode ser encontrada em (FOSTER et al., 2005).

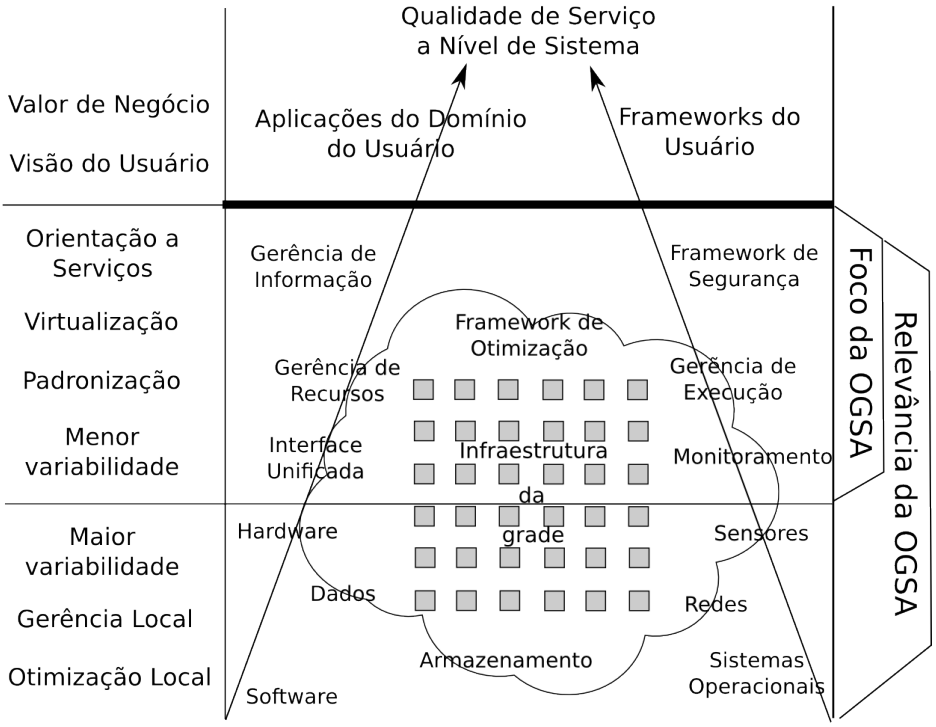

Figura 2.4: Organização lógica da OGSA (FOSTER et al., 2005).

A figura 2.4 mostra a representação abstrata de alguma das capacidades da OGSA. Três níveis são representados na figura. O nível mais baixo corresponde aos recursos primários, os quais são suportados por entidades ou artefatos externos que possuem relevância fora do escopo da OGSA. Exemplos desses recursos são processadores, memória, discos, licenças de *software* e processos de sistemas operacionais. A configuração e customização é feita em ambiente local. Esse nível apresenta alta variação de entidade e artefatos, levando a um conjunto de recursos instáveis em suas características.

O nível intermediário representa uma abstração mais alta na virtualização dos recursos. A virtualização e abstração são direcionadas a definição de uma ampla variedade de capacidades que são relevantes para grades OGSA. Tais capacidades podem ser utilizadas individualmente ou compostas para fornecer a infraestrutura necessária em aplicações de alto nível ou processos do domínio do usuário. Esse conjunto de capacidades é em sua maioria constante e invariável. A maneira na qual essas capacidades são implementadas em serviços reais determina a qualidade de serviço a nível de sistema experimentada pelo usuário. É importante notar que o primeiro e

o segundo nível são fortemente ligados, sendo que os serviços do nível intermediário acessam os recursos no nível mais baixo para prover funcionalidades para a aplicação no nível mais alto. Essa interação é baseada em um arquitetura orientada a serviços, sendo o padrão cliente-servidor um dos possíveis. A comunicação *peer-to-peer* é o caso comum.

O nível mais alto representa as aplicações e outras entidades que utilizam as capacidades OGSA para realizar funções e processos do usuário. Em sua maioria, são capacidades fora do escopo direto da OGSA, sendo necessário considerá-las apenas na elicitação de requisitos para a arquitetura ou no desenvolvimento direto de *software* para execução sobre a grade.

2.5.3 *Globus Toolkit*

O *Globus Toolkit* (FOSTER, 2006) é uma implementação da OGSA. Na verdade, como a OGSA ainda não é uma especificação completa, podemos afirmar apenas que o projeto *Globus* busca sempre manter compatibilidade total com o último *draft* da OGSA disponibilizado. Como norma técnica, a OGSA não determina qual tecnologia de comunicação deve ser utilizada, ela apenas define os elementos de uma grade e suas interfaces. O projeto *Globus* poderia então ter escolhido qualquer uma das tecnologias conhecidas (RMI, CORBA, *Web Services*, etc) para servir como *middleware* de comunicação. Duas das características dos *Web Services* foram decisivas na escolha desta tecnologia: a independência de uma linguagem específica para implementação dos serviços e a possibilidade de comunicação através de protocolos já existentes, como o HTTP. A primeira característica é muito importante, pois entidades com diferentes culturas de desenvolvimento não estão impossibilitadas de criarem uma grade *Globus.* O uso do HTTP aumenta a escalabilidade da aplicação, já que a maioria dos *firewalls* corporativos não bloqueia ou altera tráfego HTTP.

Apesar de, em teoria, os serviços *web* tradicionais serem capazes de armazenar o estado entre as invocações, não havia nenhuma padronização que descrevesse como isto poderia ser feito. No decorrer do desenvolvimento da OGSA, seus criadores definiram muitos serviços que precisavam manter algumas informações importantes durante os intervalos das invocações. Visando manter as características positivas dos serviços *web* tradicionais e adicionando mecanismos para manutenção de informações de estado, foi criada uma nova especificação que utiliza o conceito de recurso para atingir as exigências da OGSA. O nome de *Web Service Resource Framework* (WSRF) foi dado à esta nova normatização. Na figura 2.5 temos a relação entre a OGSA, o WSRF e os serviços *web* tradicionais.

Ao invés de simplesmente invocar um serviço e receber um resultado, como ocorre nos *Web Services*, no WSRF um cliente precisa, além de fornecer o endereço do serviço que deseja

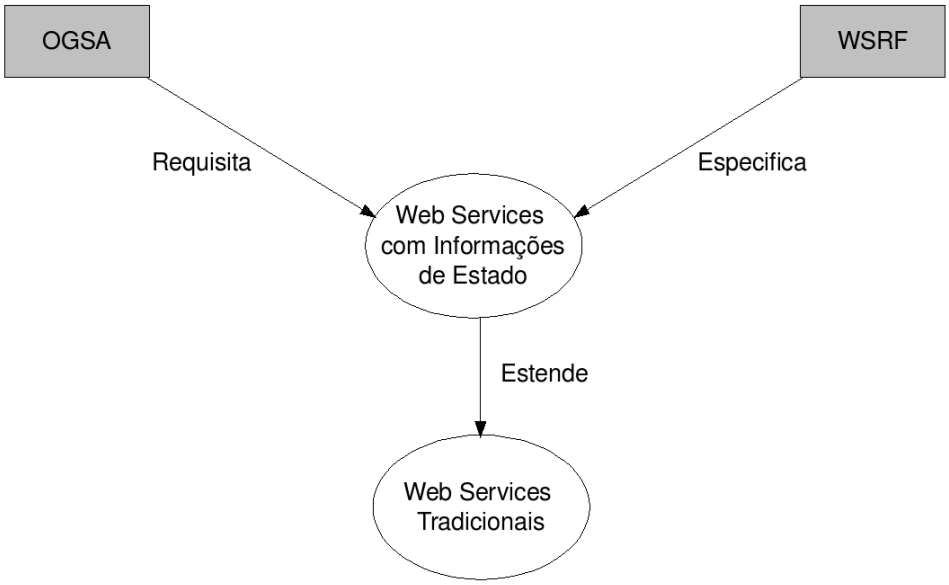

Figura 2.5: Relacionamento entre OGSA, WSRF e Web Services

invocar, informar quais recursos aquele serviço irá utilizar. Podemos fazer uma analogia com sistemas baseados em objetos remotos. Por exemplo, em Java RMI, o recurso seria um objeto remoto A que não é acessado diretamente, mas mantêm continuamente informações sobre suas interações com clientes (i.e,o estado). Para alterar o estado deste objeto, seria necessário invocar métodos em outro objeto B cujo papel é fornecer uma interface remota para que alterações sejam feitas em A. Neste esquema, B não precisaria armazenar nenhum valor em seus atributos nos intervalos entre as invocações, todas as alterações estariam em A. Outros objetos além de B poderiam funcionar como acesso à B. Completando a comparação, no WSRF, a função de B seria feita por um *Web Service* e o papel de A seria realizado por um recurso, criando assim a entidade chamada *WS-Resource*. Na figura 2.6, vemos a flexibilidade que o WSRF traz ao desenvolvimento de serviços *web* que precisam manter informações de estado. Por exemplo, um mesmo serviço pode refenciar recursos diversos, representando várias computações em momentos distintos.

Figura 2.6: *WS-Resource*

2.6 Conclusão

Desde a criação do das grades computacionais sua usabilidade já foi muito aprimorada (FOSTER, 2006). Os modelos de desenvolvimento já estão maduros e as ferramentas e metodologias criadas estão em sintonia com as necessidades dos usuários pois a maioria dos projetos adotam licenças de código aberto que permitem uma contribuição maior da comunidade (FOSTER, 2006) (GIL et al., 2007) (OINN et al., 2004). A OGSA e o *Globus Toolkit* são os alicerces para a construção de grades computacionais baseadas na invocação de serviços, sendo que os conceitos apresentados neste capítulo são necessárias para a compreensão da arquitetura do ambiente discutida no capítulo 5.

Os serviços fornecidos pelas grades computacionais podem ser vistos como elementos para a criação de *workflows*. Em conjunto com as tecnologias descritas no capítulo 3, os serviços de grades computacionais são uma alternativa para criação de um ambiente de execução de *workflows*.

3 Execução *Peer-to-peer* de *Workflows* em Grades Computacionais

Neste capítulo, a fundamentação teórica do trabalho desenvolvido é apresentada. Com o objetivo de levantar o estado da arte do escalonamento de *workflows* em grades computacionais, além dos conceitos teóricos sobre *workflows* são apresentados os aspectos gerais das redes do *peer-to-peer*. Tais redes são adotadas no escalonamento descentralizado de *workflows*. O escalonamento centralizado é discutido no contexto da utilização de algoritmos genéticos.

Uma discussão sobre *workflows* está nas seções 3.1 e 3.2. Em seguida, na seção 3.3 são apresentados conceitos sobre redes *peer-to-peer* que ajudam a contextualizar o ambiente em relação a organização dos *peers*. Na seção 3.4, a problemática da execução de *workflows* em grades computacionais é discutida, sendo que a subseção 3.4.1 discute o escalonamento centralizado e a subseção 3.4.2 detalha soluções que utilizam arquiteturas descentralizadas semelhantes à deste trabalho.

3.1 *Workflows*

Um *Workflow* é uma seqüência de etapas necessárias para que se possa atingir a automação de processos que compõem uma atividade computacional, de acordo com um conjunto de regras definidas, envolvendo a noção de processos e permitindo que estes possam ser transmitidos de uma máquina para outra de acordo com algumas regras (GIL, 2008).

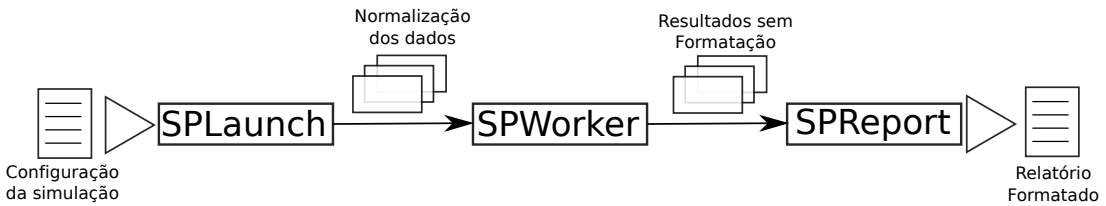

Figura 3.1: Exemplo de *workflow*.

A figura 3.1 apresenta um exemplo de *workflow*. Neste exemplo, estão presentes três etapas (*SPLaunch*, *SPWorker* e *SPReport*) de uma simulação. Os dados trocados entre as etapas

são descritos, sendo que cada etapa realiza uma computação a partir dos dados fornecidos pelo usuário até que o resultado desejado seja produzido pela última etapa.

Os sistemas de gerenciamento de *workflows* são ferramentas necessárias para controlar o cada vez mais complexo parque computacional de empresas e instituições de pesquisa (OLIVEIRA et al., 2009) (PEREIRA; TRAVASSOS, 2009). No início, quando os computadores possuíam grandes dimensões e estavam submetidos a um controle central, a utilização de linguagens *scripts* para controlar o fluxo de tarefas era comum (GIL, 2008). Essa abordagem persistiu dominante por muito tempo e, até hoje, existem algumas implementações. Entretanto, com o aumento da complexidade das computações e a distribuição dos recursos em hospedeiros conectados por redes, o esforço para manutenção de arquivos de *scripts* pouco estruturados se tornou oneroso. Para lidar com esses novos desafios, o consenso atual é que a melhor maneira de se utilizar *workflows* é através do uso de ferramentas baseadas em componentes ou serviços (GANNON, 2007).

Dois domínios de *workflows* se distinguem: empresariais e científicos. Soluções empresariais necessitam de maior integridade na execução e os usuários esperam um resultado final previsível e correto. Por exemplo, a alteração indevida do saldo de uma conta bancária pode trazer complicações legais para bancos. Nas aplicações científicas, uma flexibilidade maior é permitida, já que se tratam de experimentos que podem atender ou não as expectativas do pesquisador. O processo de criação de um *workflow* científico é incremental, sendo que o pesquisador não tem certeza das descobertas finais que obterá ao final do experimento. A infraestrutura empresarial é mais regular e confiável, os acordos de utilização de recursos são bem definidos, o que leva a poucas situações inesperadas. O mesmo não ocorre em ambientes acadêmicos onde os acordos muitas vezes são informais e os ambientes mais heterogêneos. Desta forma, os sistemas de *workflows* científicos devem ser projetados levando em conta as diferenças em relação aos sistemas empresariais (BARGA; GANNON, 2007).

A estrutura da representação de *workflows* pode ser de dois tipos: orientada a dados ou orientada a controle (SHIELDS, 2007). Em *workflows* orientados a controle, as conexões entre atividades representam a transferência de controle da tarefa anterior para a seguinte. *Workflows* orientados a dados são projetados para apoiar aplicações orientadas a dados, as dependências representam o fluxo de dados entre as atividades. A diferença entre as duas representações é muito tênue sendo comum *workflows* apresentarem uma estrutura híbrida. Como exemplo para *workflows* orientados a controle considera-se uma aplicação na qual todas as atividades precisam atualizar uma estrutura de dados compartilhada em determinada ordem. No caso não há transferência de dados entre as atividades, entre elas ocorrem apenas troca de mensagens para

sincronização na qual uma transfere o controle do banco de dados para outra. Uma aplicação na qual cada atividade recebe uma entrada, realiza alguma transformação e transfere o resultado para a atividade seguinte é um exemplo de *workflow* orientado a dados. Nesta dissertação, considera o modelo de representação orientado a controle.

3.2 Requisitos de *Workflows*

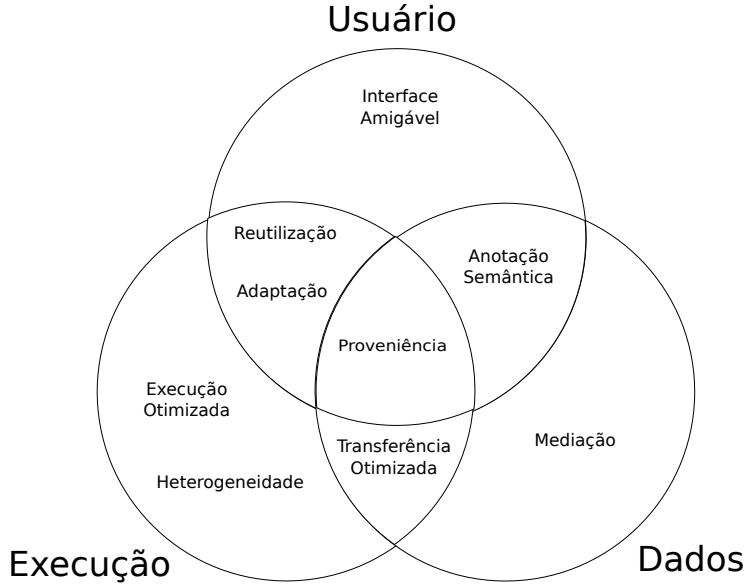

Figura 3.2: Requisitos de *Workflows*.

Nesta seção apresentamos requisitos de sistemas de *workflows* levantados a partir do estudo de aplicações (GANNON et al., 2007) (MAECHLING et al., 2007). Não é o objetivo deste trabalho realizar um processo formal de especificação de requisitos, mas apenas adequar a proposta no contexto das necessidades dos sistemas de gerenciamento de *workflows*. Um detalhamento aprofundado pode ser conferido em (YU; BUYYA, 2006b). Os requisitos são organizados levando em consideração a interação com o usuário, a gerência de execução e a gerência de dados. A figura 3.2 apresenta os requisitos discutidos de acordo com as categorias citadas.

Em relação a **interação com o usuário**, sistemas de *workflows* devem automatizar o máximo possível as tarefas e oferecer uma interface amigável e focada no domínio da aplicação (GANNON et al., 2007). Um importante aspecto é a abstração da complexidade do sistema, mas não sem permitir que o usuário tenha acesso a configurações mais extensas caso deseje. As linguagens de definição de *workflows* têm um papel importante nesse quesito. Elas podem servir como representação intermediária para um portal ou interface gráfica assim como permitir

que o usuário descreva um *workflow* diretamente nela. A BPEL e a YAWL são exemplos de linguagens para definição de *workflows* (BARGA; GANNON, 2007). O poder de expressividade ideal é uma questão em aberto na literatura (DEELMAN, 2007), pois adicionar várias estruturas de controle a uma linguagem de *workflow* pode torná-la tão difícil de usar como uma linguagem de programação normal. Uma linguagem pouco expressiva pode não ser suficiente para o usuário, pois é preciso balancear o poder de expressividade visando permitir a definição correta de *workflows* mantendo a simplicidade e clareza para o usuário final. Um outro requisito relevante é a possibilidade do usuário utilizar a linguagem para criar componentes de *workflows* reutilizáveis através da composição direta de código fonte ou da composição de serviços. Permitir ao usuário inspecionar e alterar a configuração de *workflows* em tempo de execução é também um requisito importante não relacionado a linguagem de definição. Como é explicado na seção 3.1 e em (MATTOSO et al., 2008), o processo de experimentação científica é incremental sendo assim necessário adaptar o *workflow* de acordo com resultados preliminares obtidos nas primeiras etapas.

Já a **gerência de execução** deve fornecer as funcionalidades necessárias para atender aos requisitos do usuário. *Workflows* executam por longos intervalos de tempo e é necessário tanto armazenar o estado de execução quanto atualizá-lo de acordo com a ocorrência de eventos (GANNON et al., 2007). A natureza dos eventos que influenciam a execução pode ser externa (novos dados recebidos de sensores) ou interna (ocorrência de uma falha). Em ambos os casos, a adaptação ao novo contexto com o mínimo de interferência pelo usuário é crucial. Em casos nos quais a consulta ao usuário não pode ser descartada, o sistema deve oferecer um resultado aproximado ou então uma opção de execução alternativa que consiga terminar a computação. Um ambiente de execução heterogêneo deve ser suportado e a relação de custo benefício deve compensar a curva de aprendizado.

Por fim, a **gerência de dados** tem como principal requisito a rapidez na transferência de dados. Exceto nos casos de *workflows* orientados apenas a controle, a transferência de volumes consideráveis de dados entre os componentes é comum e possui impacto determinante no desempenho do sistema. Soluções de mediação de dados também são necessárias devido a heterogeneidade inerente ao sistema. A anotação semântica dos dados enriquece o contexto de execução com informações importantes tanto para a adaptação do sistema quanto para a tomada de decisão pelo usuário final. É denominada de proveniência a tecnologia que associa semântica às diversas etapas do experimento científico (MATTOSO et al., 2008). Pode-se afirmar que proveniência de dados fornece meios de catalogar a origem dos dados utilizados durante a computação, algo importante no processo científico pois possibilita a reprodução idêntica dos experimentos para validação de descobertas. Um sistema de gerência de dados para execução de

workflows precisa lidar com a proveniência distribuída de maneira organizada e de fácil acesso ao usuário.

3.3 Aspectos Gerais de Redes *Peer-to-peer*

Podemos contabilizar dois motivos principais para o advento das redes *peer-to-peer*. Primeiro, o avanço da tecnologia fez com que as estações de trabalho adquirissem poder computacional capaz de rivalizar com o processamento de servidores dedicados. Segundo, a expansão da Internet possibilitou a conexão de várias estações de trabalho distribuídas em escala global em uma rede com velocidade e latência suficientes para o estabelecimento de elaborados protocolos para troca de mensagens a nível de aplicação. Em conjunto, esses dois fatores tornaram as redes *peer-to-peer* uma solução adequada para o provisionamento global uniforme de um determinado serviço na Internet.

Uma das vantagens das redes *peer-to-peer* é a distribuição de carga. Se na arquitetura cliente-servidor o servidor hospedeiro é responsável por atender requisições de vários clientes, na arquitetura *Peer-to-peer*, ao contrário, o papel duplo de cada hospedeiro (cliente e servidor) permite que a carga seja distribuída através do sistema. A possibilidade de falha geral do sistema também é reduzida, já que a ausência de um hospedeiro pode ser compensada por outro hospedeiro que represente o mesmo recurso.

O primeiro uso da tecnologia P2P a se popularizar foi o compartilhamento de arquivos multimídias (COULOURIS; DOLLIMORE; KINDBERG, 2005). O Napster (FANNING, 1999) foi um dos protocolos pioneiros, entretanto, sua estrutura ainda apresentava algum grau de centralização. Os dados eram distribuídos entre os vários hospedeiros da rede, mas existia um hospedeiro central que mantinha um mapeamento de cada arquivo para todos os hospedeiros que o armazenava. Essa arquitetura centralizada também está presente em um dos primeiros projetos a utilizar P2P para computação de alto desempenho, o projeto SETI (ANDERSON et al., 2002). No SETI, um servidor central é responsável por criar tarefas que são submetidas aos hospedeiros distribuídos ao redor do globo. Seja por razões legais ou por uma necessidade de eliminar gargalos ainda existentes, soluções posteriores trataram de abandonar qualquer serviço centralizado e adotar arquiteturas nas quais as funcionalidades essenciais como a descoberta e a requisição de recursos não dependiam de de nenhum hospedeiro especial. Essas funcionalidades eram realizadas de maneira colaborativa por um grupo de hospedeiros. Para tal, são utilizados algoritmos distribuídos de roteamento para criação de redes sobrepostas sobre o roteamento IP da Internet.

3.3.1 Redes *Peer-to-peer* Sobrepostas

As redes sobrepostas são responsáveis pelo encaminhamento de mensagens em sistemas P2P. Os tipos de redes sobrepostas estão na tabela 3.1. As redes sobrepostas estruturadas controlam rigidamente a topologia da rede e os objetos ou referências são armazenados não em *peers* aleatórios, mas em hospedeiros específicos que otimizam as buscas posteriores (LUA et al., 2005). Já as redes não estruturadas usam técnicas de inundação para trocas de mensagens.

Tabela 3.1: Classificação de Sistemas *Peer-to-peer* (CARDOSO, 2007).

Não estruturada			Estruturada
P2P Centralizada	P2P Híbrida	P2P Descentralizada	P2P Baseada em DHT
Entidade central é necessária, responsável pelo índice geral.	Entidade centrais dinâmicas.	Sem entidades centrais.	Sem entidades centrais, conexões com alto controle.

Os sistemas estruturados possuem forte base teórica e baixa complexidade em relação ao número de mensagens trocadas nas tarefas de busca e de inserção. Os identificadores dos objetos e dos hospedeiros obedecem o mesmo formato, em geral alguma notação hexadecimal. Dado um objeto ou referência X, o hospedeiro que o contém é aquele cujo identificador é o mais próximo de X de acordo com uma função matemática. Cada *peer* possui uma tabela com seus vizinhos com identificadores mais próximos de acordo com a mesma função. Devido a essa tabela, tais sistemas costumam ser chamados de DHT (Distributed Hash Table) (COULOURIS; DOLLIMORE; KINDBERG, 2005). Porém, os identificadores dos objetos não guardam nenhuma semântica em relação ao conteúdo do objeto em si. Em geral, a codificação hexadecimal é criada a partir de uma função *hash*. Portanto, em ambientes nos quais a interação com o usuário é direta e as buscas são feitas baseadas em palavras chave ou conteúdo semântico do objeto, as redes não estruturadas são mais adequadas. Tal adequação é observada em situações reais, nas quais a maioria dos sistemas de compartilhamento de arquivos multimídia são não estruturados. Entretanto, os sistemas de arquivos distribuídos ou de replicação de cache optam por soluções estruturadas (LUA et al., 2005).

Uma rede não estruturada é formada por hospedeiros que entram na rede com regras flexíveis, sem nenhum ou pouco conhecimento da topologia anterior. A rede utiliza inundação como o mecanismo para distribuir consultas através da rede. Quando um hospedeiro recebe uma consulta, envia uma lista de todos os objetos que atendem a consulta ao hospedeiro remetente. Esse esquema apresenta algumas desvantagens. Se o parâmetro de propagação não for bem configurado, é possível que objetos raros na rede não seja encontrados. Se a quantidade de

buscas crescer exponencialmente e o processo de busca for oneroso, os hospedeiros logo se tornam sobrecarregados, pois precisam processar boa parte das consultas submetidas à rede.

Dentre os sistemas não estruturados, três organizações são possíveis. As redes *peer-to-peer* centralizadas possuem uma entidade central que fornece um índice geral com informações sobre a rede. Esse índice central permite controlar a inundação de mensagens, mas cria um gargalo para o sistema. As redes *peer-to-peer* híbridas replicam o índice entre vários servidores centrais dinâmicos, diminuindo o impacto das limitações de escalabilidade e mantendo o controle da inundação de mensagens. Entretanto, um desafio nas redes híbridas é decidir quais *peers* podem atuar como servidores centrais. As redes descentralizadas não apresentam nenhum *peer* com privilégios, sendo que qualquer integrante pode ser retirado sem perda de funcionalidade. A principal desvantagem das redes descentralizadas é que a utilização da técnica de inundação pode causar impacto negativo no desempenho da rede de comunicação.

3.3.2 *Peer-to-peer* e Grades Computacionais

A união dos conceitos de grades e *peer-to-peer* têm como objetivo o uso coordenado de recursos pertencentes a comunidades distribuídas. Os dois tipos de infraestrutura são construídos como estruturas sobrepostas que operam com ampla independência de relacionamentos institucionais (FOSTER; IAMNITCHI, 2003). Tanto as grades quanto as redes *peer-to-peer* buscam soluções para o mesmo problema do compartilhamento ordenado de recurso em organizações virtuais. A abordagem utilizada nas duas tecnologias envolve a criação de estruturas sobrepostas que coexistem com a estrutural organizacional real, porém sem a necessidade de correspondência entre as estruturas. Todavia, não existem apenas semelhanças entre os dois. Grades computacionais costumam apresentar uma grande variedade de serviços dispostos em um sistema distribuído de escala média, no qual há uma infraestrutura de segurança que permite a utilização confiável dos serviços. Já ambientes P2P fornecem apenas um tipo de serviço ou uma pequena variação do mesmo, em uma escala global, nem sempre com a segurança adequada. Os recursos presentes nas grades apresentam disponibilidade maior, enquanto nas redes *peer-to-peer* boa parte das máquinas conectadas tem presença intermitente. As aplicações que fornecem acesso a redes *peer-to-peer* em geral são de fácil utilização, usuários não especializados conseguem instalar e acessar os serviços oferecidos. Em contraste, a instalação e configuração de grades computacionais não é uma tarefa simples, sendo que sua implantação exige conhecimentos técnicos avançados.

Apesar das diferenças mencionadas anteriormente, o sucesso de projetos como o *OurGrid* (CIRNE et al., 2006) (CIRNE; BRASILEIRO; PARANHOS, 2003) e o SETI (ANDER-

SON et al., 2002) mostra que é possível tirar proveito dos benefícios de cada tecnologia para construção de sistemas distribuídos. Essas soluções flexibilizam características das grades e das redes P2P em busca de uma maior facilidade de uso. Por exemplo em redes *peer-to-peer*, não há um ambiente de segurança completo como nas grades tradicionais, algo que é de configuração complicada. Em compensação nas redes *peer-to-peer*, a replicação de tarefas e a comparação de resultados permite o estabelecimento de um sistema de reputação que é transparente ao usuário. Nas grades computacionais, não há técnicas para aumentar a disponibilidade dos recursos, porém os hospedeiros da rede, apesar de serem estações de trabalho, fazem parte de universidades ou instituições de pesquisa, não sendo tão imprevisíveis quanto as máquinas de usuários domésticos que constituem as redes P2P de escala global. Desta forma, sistemas distribuídos capazes de tirar proveito dos dois conceitos são viáveis e se apresentam como boa alternativa para a construção de soluções robustas e de fácil utilização.

3.4 Escalonamento de *Workflows* em Grades Computacionais

Escalonamento de *workflows* trata do mapeamento e gerência de um conjunto interdependente de etapas em vários recursos computacionais. No caso das grades computacionais, os recursos computacionais pertencem a organizações diferentes. O objetivo de uma estratégia de escalonamento eficiente é alocar recursos adequados de modo a considerar os requisitos dos usuários e mantenedores do sistema. A arquitetura lógica de um ambiente de gerência de *workflows* é apresentada na figura 3.3 (WFMC, 2009) (HOLLINSWORTH, 1994).

As funções de um sistema de gerência de *workflows* podem ser classificadas em duas categorias: funções de tempo de projeto e funções de tempo de execução. O primeiro conjunto de funções está ligado à definição e modelagem das etapas de um *workflow* e suas dependências. O segundo grupo de funcionalidades controla a execução de *workflows* e as interações necessárias com os recursos e serviços da grade. Os usuários interagem com as ferramentas de modelagem de *workflows* para gerarem uma especificação de *workflow*, que por sua vez é submetida a um serviço de tempo de execução chamado Motor de Execução de *Workflow*. As principais funcionalidades fornecidas por esse serviço são escalonamento, gerência de falhas e de dados.

O escalonamento de *workflows* descobre recursos e distribui tarefas em recursos adequados de acordo com os requisitos dos usuários. A gerência de dados cuida da transferência de dados entre os recursos selecionados e a gerência de falhas habilita mecanismos para lidar com falhas durante a execução. O usuário tem uma visão do processo de execução através do

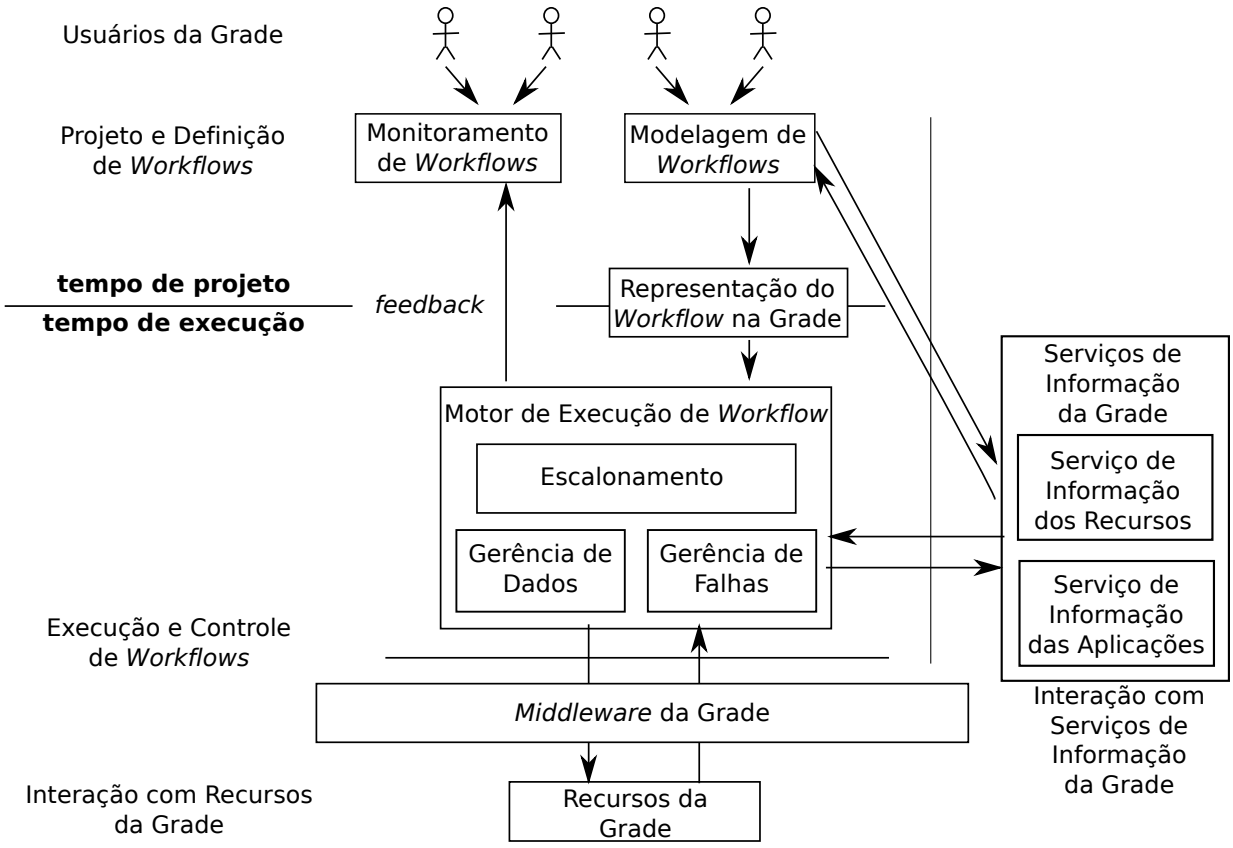

Figura 3.3: Sistema de gerência de *Workflow* (YU; BUYYA, 2006b).

feedback fornecido pelo Motor de Execução ao Sistema de Monitoramento de *Workflows*. O Motor de Execução de *Workflow* pode ser construído sobre vários *middlewares* de grades computacionais, como por exemplo o Globus (FOSTER, 2006). Através do *middleware*, o Motor de Execução é capaz de invocar os serviços fornecidos pelos recursos da grade. Ambas funções de projeto e execução consultam os serviços de informações da grade para recuperar meta-dados sobre recursos e aplicações.

Em relação ao escalonamento, arquitetura de distribuição da infraestrutura é importante para a escalabilidade, autonomia e desempenho do sistema. Há três categorias principais para arquiteturas de escalonamento de *workflows* em grades computacionais (YU; BUYYA, 2006b): centralizada, hierárquica e descentralizada.

Nas arquiteturas centralizadas, um escalonador centralizado é responsável pelas decisões de escalonamento de todas as tarefas no *workflow*. O escalonador possui informações sobre todo o *workflow* e coleta as informações necessárias em todos os recursos disponíveis. A presença de informações descrevendo o estado global da grade permite uma tomada de decisões eficiente, porém tal arquitetura não é escalável em relação ao número de tarefas e quantidade de recursos.

No escalonamento hierárquico, há uma entidade central e vários escalonadores distribuídos em níveis inferiores. A entidade central é responsável por controlar a execução do *workflow* e pela atribuição de partições do *workflow* para escalonadores de níveis inferiores. O escalonamento é submetido à entidade central, que analisa quais os recursos são necessários e distribui porções do *workflow* para escalonadores de acordo com a capacidade e disponibilidade de recursos de cada um. Cada escalonador de baixo nível é responsável pela distribuição de etapas do *workflow* dentro de uma organização. A maior vantagem dessa arquitetura é a possibilidade de implantar diferentes políticas em cada organização. Em virtude dessa característica, o escalonamento hierárquico se encaixa no perfil das grades computacionais com perfeição. Entretanto, a falha da entidade central resulta, para o usuário final, na total ausência de funcionamento da grade.

O escalonamento descentralizado ocorre quando vários escalonadores sem controle central decidem sobre o fluxo da execução do *workflow*. Para tal, é preciso etapas de negociação entre os processos distintos para se chegar a uma decisão conjunta. Comparada com as outras arquiteturas, a opção descentralizada é mais escalável e tolerante a falhas, porém enfrenta dificuldades para gerar soluções ótimas e minimizar problemas de conflito. Uma maneira de minimizar as carências dessa estratégia é organizar os processos de escalonamento em uma rede *peer-to-peer*. As redes sobrepostas otimizam a troca de mensagens entre seus membros, portanto escalonadores organizados em uma rede *peer-to-peer* obtêm as informações que precisam para chegar a um consenso entre si de maneira controlada.

3.4.1 Escalonamento Centralizado

Nesta seção, são apresentadas soluções para escalonamento que fornecem resultados eficientes para as arquiteturas centralizadas e hierárquicas. Esta dissertação propõe uma solução descentralizada, entretanto é importante apresentar o paradigma centralizado já que o mesmo é utilizado em projetos importantes (OINN et al., 2004) (ANDERSON et al., 2002). Essa seção também serve como fundamentação para a análise feita no capítulo 5, na qual a solução descentralizada proposta nesta dissertação é comparada com um escalonador centralizado.

Em geral, o problema do mapeamento de etapas de um grafo de dependências em recursos distribuídos pertence a classe de problemas conhecida como NP-Difíceis (FERNANDEZ-BACA, 1989). Desta forma, algoritmos exatos que levam à uma solução ótima possuem complexidade limitada por uma função exponencial. Em situações nas quais o número de etapas ou recursos aumentam de maneira considerável, o tempo de execução de tais algoritmos torna proibitivo o seu uso em ambientes reais.

As estratégias centralizadas se dividem em dois grupos: abordagens tradicionais (seção 3.4.1.1) e algoritmos genéticos (seção 3.4.1.2). As abordagens tradicionais são soluções com caráter guloso (*greedy*) acentuado, lidando com apenas uma métrica de qualidade de serviço. Algoritmos genéticos são uma meta-heurística que pode ser utilizada para tratar problemas nos quais existem mais do que uma métrica a ser considerada. Existem análises (YU; BUYYA, 2006a) que comparam ambos os grupos e demonstram que os algoritmos genéticos apresentam resultados melhores. Por essa razão, tal técnica foi escolhida como base para a análise feita no capítulo 5.

3.4.1.1 Abordagens Tradicionais

As abordagens tradicionais para o escalonamento centralizado lidam com a complexidade do problema ignorando a interdependência das etapas de um *workflow*. Partindo do início do *workflow*, cada etapa é avaliada individualmente, sendo que os recursos disponíveis são qualificados de acordo com um critério. Tal critério é baseado em uma única métrica de qualidade de serviço, em geral o custo ou o tempo de execução. A diferença entre as técnicas está em como a critério é avaliado, seja considerando apenas informações recentes ou uma análise estatística do passado da grade. Um exemplo dessas estratégias é a técnica *MinMin* (ZHANG; LI, 2009). Um estudo detalhado de outras abordagens pode ser encontrado em (YU; BUYYA; THAM, 2005).

3.4.1.2 Algoritmos Genéticos

Uma maneira de preservar a escalabilidade do sistema e obter soluções de qualidade para o escalonamento é o emprego de algoritmos empíricos que possuem complexidade governada por funções polinomiais. Uma das técnicas utilizadas com sucesso em grades computacionais é a heurística de algoritmo genéticos (TALUKDER; KIRLEY; BUYYA, 2009). Como a maioria das heurísticas, os algoritmos genéticos possuem uma fase de busca local que é intercalada com a perturbação das soluções já encontradas. A busca local descobre dentro de uma secção do espaço de soluções quais são as melhores, enquanto a perturbação força o algoritmo a periodicamente realizar a busca local em outras secções. Desta forma, a meta é vasculhar o espaço de soluções em busca do ótimo global, evitando a armadilha de encontrar apenas soluções ótimas locais.

A descrição do uso de algoritmos genéticos apresentada nesta dissertação foi retirada de (YU; BUYYA, 2006a). Nesse trabalho, os autores consideram os requisitos de qualidade de serviço relacionados ao intervalo de tempo e ao custo necessários para completar a execu-

ção. Vale ressaltar que o objetivo desta dissertação não é fornecer uma explicação geral do funcionamento dos algoritmos genéticos, mas sim direcionar a explicação para o problema de escalonamento de *workflows*. Para maiores informações sobre o funcionamento geral dos algoritmos genéticos, o leitor deve procurar (VIANA, 1998).

Algoritmos genéticos fornecem técnicas robustas que permitem derivar uma solução de alta qualidade a partir de um grande espaço de soluções em tempo polinomial (GOLDBERG, 1989). Qualquer solução no espaço de busca é representada por um conjunto de parâmetros chamados de indivíduo.

Um algoritmo genético mantêm uma população formada por um conjunto de indivíduos que evolui através das gerações. A qualidade de um indivíduo é determinada pela função de aptidão (*fitness function*). Um algoritmo genético típico consiste dos seguintes passos:

1. Criação da população inicial de indivíduos aleatórios;
2. Geração de herdeiros a partir da aplicação dos operadores genéticos (*crossover* e *mutation*);
3. Avaliação da aptidão de cada indivíduo na população e a seleção dos que farão parte da próxima geração;
4. Repetição dos passos 2 e 3 até que o algoritmo atinja uma condição terminal, que pode ser a quantidade de iterações ou um valor adequado encontrado.

Utilizar algoritmos genéticos para resolver o problema do escalonamento de *workflows* requer a definição de como o indivíduo é representado, da função de aptidão, das operações genéticas e do esquema de seleção. Os detalhes de cada um desses requisitos são apresentados nas seções seguintes.

Representação do Indivíduo

Para o escalonamento de *workflows*, de acordo com (YU; BUYYA, 2006a), uma solução adequada deve atender as seguintes restrições:

- Uma etapa do *workflow* só pode iniciar após a finalização de todas antecessoras;
- Cada etapa aparece apenas uma vez no escalonamento; e,

- Cada etapa só deve ser alocada para um recurso.

O termo etapa é usado para designar a tarefa ou invocação de serviço integrante do *workflow*, dependendo do paradigma adotado. Cada indivíduo no espaço de soluções é representado por um vetor de atribuições. Cada atribuição consiste de dois elementos: o identificador da etapa e o identificador do recurso que irá atender a etapa. Para cada vetor definido, existe também uma fila que representa a ordem de escalonamento de cada etapa. Tal fila é importante para reforçar a restrição de que uma etapa só pode ser iniciada após as antecessoras.

A figura 3.4 representa as duas estruturas de dados para representação de um indivíduo *workflow* no algoritmo. O vetor de atribuição designa qual recurso atenderá cada etapa. Por exemplo, a etapa $E0$ é designada para o recurso $R1$. A fila codifica a ordem de cada alocação. Por exemplo, a etapa $E3$ só é escalonada após $E0$ e $E1$.

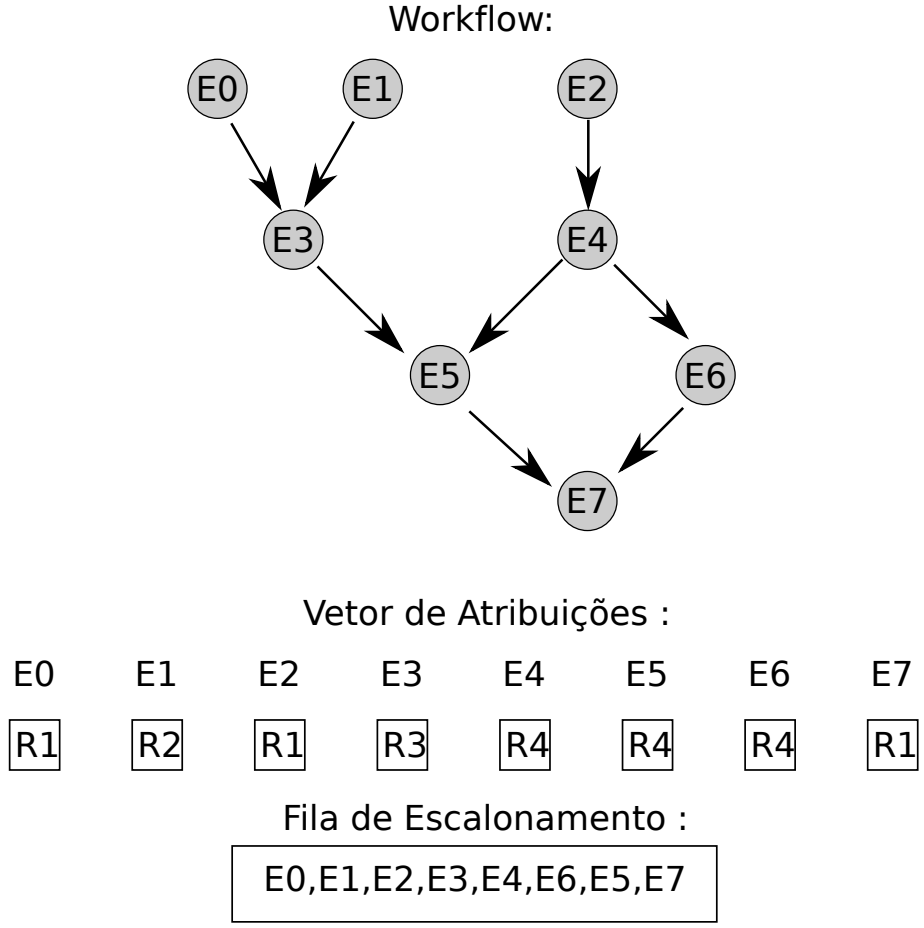

Figura 3.4: Representação do Indivíduo.

Para definição da população inicial, é preciso estabelecer um conjunto aleatório de indivíduos. O que ocorre de fato é a geração aleatória do vetor de atribuição e da fila de submissão. Entretanto, é importante observar que as duas estruturas de dados só são válidas se obedecerem

as restrições do problema. Por exemplo, o vetor de atribuição só pode atribuir uma etapa para um recurso se o mesmo tiver todas as características necessárias para atender a etapa. No caso da submissão de tarefas, uma tarefa só pode ser submetida a um recurso se o mesmo tiver o processamento necessário para executá-la. Caso a etapa seja uma invocação de serviços, uma invocação só pode ser submetida a um recurso se o mesmo tiver o serviço necessário. Já a fila de submissão precisa observar a ordem de antecedência das etapas para poder ser consistente.

Uma solução adotada por (YU; BUYYA, 2006a) para definição da população consiste em utilizar um algoritmo que forneça arranjos aleatórios entre as etapas e os recursos. Tal algoritmo é executado até ser encontrado um arranjo que satisfaça as restrições do problema. Da mesma forma, um algoritmo que permute as etapas de maneira aleatória pode ser executado até que se encontre uma solução adequada. Ambos os casos não apresentam a complexidade computacional ideal, mas como só são usados apenas uma vez no início de cada execução do algoritmo genético, o impacto dessa deficiência é minimizado.

Operações Genéticas

Crossovers são utilizados para criar novas soluções através da reorganização das soluções existentes na população atual. A ideia por trás do *crossover* é que uma solução melhor pode resultar da combinação de duas soluções já existentes. Para o escalonamento de *workflows*, o *crossover* é feito da seguinte maneira (YU; BUYYA, 2006a) :

1. Dois ancestrais são escolhidos de maneira aleatória na população;
2. Duas posições aleatórias são escolhidas nos vetores de atribuição para formar uma janela *crossover* nos dois vetores; e
3. Todas as etapas inclusas na janela *crossover* são permutadas entre os vetores.

Na figura 3.5 está representada a operação de *crossover* de acordo com o processo descrito acima. Uma janela de tamanho três é definida em dois vetores de atribuição, cujas secções (ou genes) são permutadas.

Mutações ocorrem ocasionalmente para permitir um descendente obter características que não são oriundas dos seus ancestrais. Dois tipos de mutações existem para o problema de escalonamento de *workflows*: reordenação e substituição. Os operadores de mutação são aplicados para as soluções escolhidas de acordo com certas probabilidades.

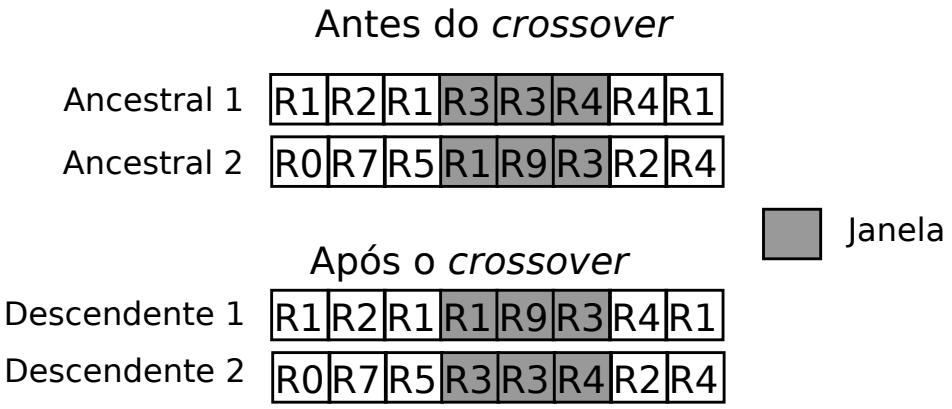

Figura 3.5: Operação de *Crossover*.

Uma mutação de reordenação tem como objetivo mudar a ordem de execução de tarefas independentes. Foi implementada em (YU; BUYYA, 2006a) da seguinte maneira:

1. Uma etapa na fila de escalonamento é escolhida de maneira aleatória; e
2. Uma posição alternativa para essa etapa é escolhida de maneira aleatória entre todas as possíveis de acordo com restrições de antecedência e recursos.

As posições possíveis para uma tarefa na fila são aquelas posteriores aos seus ancestrais e anteriores aos seus descendentes. Trata-se então de uma simples permutação dentro da fila de escalonamento. Uma mutação de substituição procura realocar um recurso alternativo para uma etapa na solução de acordo com os seguintes passos:

1. Uma tarefa aleatória é escolhida no vetor de atribuições; e
2. Um recurso alternativo que é capaz de executar a etapa é selecionado de maneira aleatória para substituir a atribuição corrente.

A mutação é mais simples do que o *crossover*, pois envolve apenas uma permutação aleatória do formato das duas estruturas de dados usadas para representar os indivíduos. Entretanto, é importante notar que por serem baseados em processos biológicos reais, os algoritmos genéticos em geral adotam a probabilidade de ocorrência de uma mutação como menor do que a probabilidade de um *crossover*.

Função de Aptidão

Uma função de aptidão é usada para medir a qualidade dos indivíduos na população de acordo com um critério de otimização. No trabalho em questão (YU; BUYYA, 2006a), são definidas duas funções de aptidão: custo e tempo de execução.

Para o escalonamento com restrição de custo, a função de aptidão de custo encoraja a formação de soluções que satisfação as restrições de orçamento. Para o escalonamento com restrição de prazo, a função incentiva o algoritmo genético escolher os indivíduos com menor custo. A função de aptidão de custo de um indivíduo I é definida por:

$$F_{cost}(I) = c(I)/(B^{\alpha}(maxCusto^{1-\alpha})) \tag{3.1}$$

Na equação 3.1, $c(I)$ significa o custo do indivíduo e *maxCusto* é a solução mais cara da população atual, B é o orçamento do *workflow*. O valor α é binário, no qual α é 0 para escalonamento com restrição de prazo e 1 para restrição de orçamento.

Com restrições de orçamento, a função de aptidão ao prazo é projetada para encorajar o algoritmo a escolher indivíduos com o tempo de finalização mais próximos. No caso da restrição de prazo, incentiva a formação de indivíduos que satisfação a restrição de prazo. A função de tempo de execução de um indivíduo é dada por:

$$F_{time}(I) = t(I)/(D^{\beta}(maxTempo^{1-\beta})) \tag{3.2}$$

Na equação 3.2, $t(I)$ significa o tempo de finalização do indivíduo e *maxTempo* é a solução com tempo de finalização mais distante da população atual, D é o prazo do *workflow* . O valor β é binário, no qual β é 0 para escalonamento com restrição de orçamento e 1 para restrição de prazo.

Suponha que o usuário forneça um prazo para execução do *workflow*. O ideal é o ambiente cumprir tal prazo com o menor custo possível. Nessa situação, $F_{cost}(I) = c(I)/maxCusto$ e $F_{time}(I) = t(I)/D$. O que dita a aptidão ao tempo do indivíduo é o fator D, informado pelo usuário. Já a aptidão ao custo é considerada apenas pela comparação com o restante da população, sem nenhuma entrada do usuário. No caso da restrição por orçamento fornecido pelo usuário, ocorre situação semelhante. A entrada do usuário consiste de qual métrica deseja impor limites, sendo que a outra métrica sempre é otimizada em relação à população.

Para o problema da restrição de prazo, a função de aptidão final é dada pela equação 3.3.

$$F(I) = \begin{cases} F_{time}(I) & \text{se } F_{time}(I) > 1. \\ F_{cost}(I) & \text{caso contrário.} \end{cases} \tag{3.3}$$

Para o problema da restrição de orçamento, a função de aptidão final é dada pela equação 3.4.

$$F(I) = \begin{cases} F_{cost}(I) & \text{se } F_{cost}(I) > 1. \\ F_{time}(I) & \text{caso contrário.} \end{cases} \tag{3.4}$$

Esquema de Seleção

Após o processo de avaliação de aptidão, os novos indivíduos são comparados com a geração anterior. O esquema de seleção é então conduzido para manter os indivíduos mais aptos na população. As opções existentes são:

- **Seleção Roleta Russa**: nessa seleção, cada indivíduo ocupa uma seção de uma roleta com tamanho de acordo com o valor da função de aptidão. Um indivíduo mais apto ocupa uma maior área na roleta. Um ponteiro é jogado aleatoriamente na roleta, o indivíduo escolhido é aquele cuja área for atingida.
- **Seleção por Categoria**: esse método ordena os indivíduos de acordo com seu valor da função de aptidão e escolhe primeiro aqueles com os melhores valores até determinado patamar.
- **Roleta Russa Elitista**: similar ao roleta russa, porém primeiro escolhe o melhor indivíduo e o promove para a próxima geração, depois realiza roleta russa com o restante.
- **Categoria Elitista**: similar à seleção por categoria, porém primeiro escolhe o melhor indivíduo e o promove para a próxima geração, depois realiza seleção por categoria com o restante.

A escolha do esquema seleção ideal é deixada a cargo do desenvolvedor do escalonador. No trabalho (YU; BUYYA, 2006a), o esquema que demonstrou melhores resultados foi a escolha por categoria elitista. Tal opção foi escolhida a partir do resultados de simulações.

Arquitetura de Escalonamento

Para fazer uso dos algoritmos genéticos, o problema do escalonamento de *workflows* precisa ser modelado em um problema de otimização combinatória. As restrições impostas são derivadas dos requisitos de QoS do usuário e do sistema. A princípio, qualquer uma das categorias de arquiteturas podem ser utilizadas para implementar ambientes de execução que fazem uso de heurísticas para otimizar o escalonamento. Entretanto, para heurísticas como algoritmos genéticos funcionarem, precisa-se de informações sobre todos os recursos disponíveis na grade. As arquiteturas centralizadas ou hierárquicas se apresentam como as mais adequadas, pois existe uma maneira ordenada de recuperar os dados necessários nessas abordagens.

3.4.2 Escalonamento Descentralizado

Nesta seção, de acordo com conceitos apresentados nas seções 3.2 e 3.3, é apresentado o levantamento das principais soluções descentralizadas para a execução de *workflows* em grades computacionais. É importante observar, nas soluções que adotam o paradigma *peer-to-peer* por completo, se a rede sobreposta adotada trata-se de uma opção estruturada ou não estruturada. A organização da rede sobreposta influencia no formalismo adotado para a descrição dos recursos na rede *peer-to-peer*, que por sua vez tem impacto no tipo de grade que pode ser formada.

3.4.2.1 SwinDeW

Considerando sua natureza conceitual e tecnológica, *workflows* devem ser executados em ambientes distribuídos. Apesar de ser utilizado em inúmeros sistemas distribuídos, o paradigma cliente-servidor não considera a distribuição total das responsabilidades do sistema, sendo o controle principal da computação restrito a alguns servidores. Os criadores do SwinDeW (YAN; YANG; RAIKUNDALIA, 2006) afirmam que o contraste entre a natureza distribuída dos *workflows* e o grau de centralização das arquiteturas cliente-servidor acarretam a vários problemas, dentre eles, a queda de desempenho e problemas de escalabilidade. A alternativa proposta pelo SwinDew consiste em um sistema de gerência de *workflows* construído utilizando os paradigmas arquiteturais de sistemas *peer-to-peer* não estruturados.

No SwinDeW, as funcionalidades oferecidas pelo sistema de gerenciamento de *workflows* são divididas em duas categorias: tempo de desenvolvimento e tempo de execução. Funcionalidades de tempo de desenvolvimento são relacionadas com o desenvolvimento de um

modelo de *workflow*, incluindo sua representação e armazenamento. O modelo é então transformado em uma instância no momento da execução e as funcionalidades de tempo de execução são responsáveis pela gerência da instância criada. Utilizando o paradigma cliente-servidor, as funcionalidades se concentram em sua maior parte nos servidores. Para construir um ambiente similar com melhorias, um sistema *peer-to-peer* deve apresentar as seguintes características:

- Adoção de uma estrutura pouco acoplada, sem a presença de repositórios ou coordenação centrais.
- Armazenamento distribuído.
- Migração de serviços de servidores para os todos os nós participantes.
- Otimizar o tráfego do sistema através de descoberta inteligente.
- Permitir que o cliente e o provedor de um serviço se comuniquem diretamente.
- Aceitar aplicações orientadas a serviços.

Para os desenvolvedores do SwinDeW, uma capacidade é um objeto possuidor de regras com um papel em processos de *workflow*, incluindo a responsabilidade desse papel, cenários de uso e restrições relacionadas a aplicação. Nada é definido sobre o formalismo a ser utilizado para construir tais capacidades, apenas que elas devem ser criadas manualmente a partir do conhecimento do domínio.

Se um *peer* possui os componentes necessários para desempenhar o papel descrito por uma capacidade, então ele é dito possuidor dessa mesma capacidade. Os *peers* são agrupados pelas capacidades que possuem. Existem tantos grupos quanto capacidades. Se um *peer* possui uma capacidade, então ele pertence ao grupo dessa capacidade. Um *peer* pode possuir várias capacidades, logo ele pode pertencer a vários grupos. Existe então uma rede de grupos interligados pelos *peers* com capacidades em comum. A descoberta de *peers* é baseada em busca de capacidades. Uma busca possui anexada uma capacidade desejada. O *peer* que recebe a busca procura no seu repositório informações sobre *peers* que atendam tal capacidade. Na ausência dessa informação, a busca é repassada para os *peers* que fazem parte dos mesmos grupos do *peer* inicial. O processo é repetido a partir de cada um deles até que um resultado positivo seja retornado ou a rede seja totalmente vasculhada. Um caso indesejável seria a existência de um conjunto de *peers* com apenas uma capacidade, sendo que essa capacidade não é compartilhada por nenhum outro *peer* na rede. Dessa forma haveria um grupo isolado do restante da rede. Segundo os autores, essa situação é rara e é prevista a necessidade de intervenção gerencial para saná-la.

A execução de uma instância de *workflow* no SwinDew começa pela alocação de *peers* para as tarefas componentes. O resultado da alocação completa é uma rede de *peers* interligados capaz de executar o *workflow*. O primeiro passo para a criação dessa rede é recebimento por parte de um *peer* de um estímulo externo, seja do usuário ou de outro sistema, que engatilha o processo de definição da rede. O *peer* que recebe tal estímulo é chamado de *peer* de instanciação atual. Esse *peer* procura outros *peers* capazes de executar as tarefas sucessoras, armazenando o resultado da busca no seu repositório de *peers*. Se por sua vez as tarefas sucessoras também tiverem tarefas sucessoras, os *peers* selecionados passam a agir como *peers* de instanciação e o processo se inicia novamente até que as tarefas finais sejam atingidas, ou seja, aquelas que não possuem sucessoras.

Em vários momentos da criação da rede que irá executar o *workflow*, existe a necessidade de negociação. O SwinDeW opta por adotar um processo que leve ao balanceamento de carga do sistema. O primeiro *peer* a receber uma requisição em um grupo de candidatos se torna responsável pelo processo de negociação. Para otimizar o sistema local, dentro de um mesmo grupo de capacidade, todos os *peers* candidatos são identificados. A carga de trabalho de cada *peer i* em um período de um dia, por exemplo, é calculada usando a fórmula $w_i = \Sigma t_k$. A variável t_k representa a carga de trabalho de uma tarefa k que tenha sido designada ao *peer* no período de tempo em estudo. A tarefa é designada para o *peer* que tiver o menor valor de w. O objetivo é tentar aproximar a carga de todos os *peers* para um valor similar.

É importante observar que tanto a execução ocorre de maneira *peer-to-peer*, sendo a troca de mensagens orientada a *peers* pertencentes a determinada comunidade, sem grandes custos devido a *broadcasts* para toda a rede. O protótipo do SwinDew implementado por seus criadores utiliza a tecnologia JXTA para a criação de sistemas *peer-to-peer*. Dessa forma, apesar de utilizar uma rede *peer-to-peer* sobreposta não estruturada, a principal desvantagem desse tipo de sobreposição é minimizada.

3.4.2.2 *Cooperative and Decentralized Workflow Scheduling in Global Grids*

Em (RANJAN; RAHMAN; BUYYA, 2008) é apresentada uma solução que utiliza um índice distribuído baseado em DHT (Distributed Hash Table) para troca de mensagens e coordenação entre os *peers* de uma grade computacional, formando uma rede *peer-to-peer* estruturada. O índice funciona como um espaço de memória distribuída compartilhada, no qual cada peer pode informar requisições vindas dos seus usuários ou publicar informes sobre recursos disponíveis. Essa memória representa um espaço vetorial de várias dimensões, sendo que cada dimensão representa um aspecto das requisições como processador, memória ou largura

de banda.

O espaço de informações é particionado e cada *peer* fica responsável por manter uma partição, atendendo aos pedidos de publicação de requisições ou informes de recursos disponíveis. Quando um informe de recursos disponíveis publicado por um *peer* combina com uma requisição publicada por outro *peer*, o primeiro *peer* se torna responsável por atender a requisição. Para a execução de um *workflow*, o usuário submete a instância do mesmo com as tarefas anotadas com o recursos que são necessários para a execução. Um agente local em cada peer trata e publicar os requisitos de cada tarefa e espera por um outro peer responder ao pedido. Nesse momento a tarefa é despachada para o recurso escolhido.

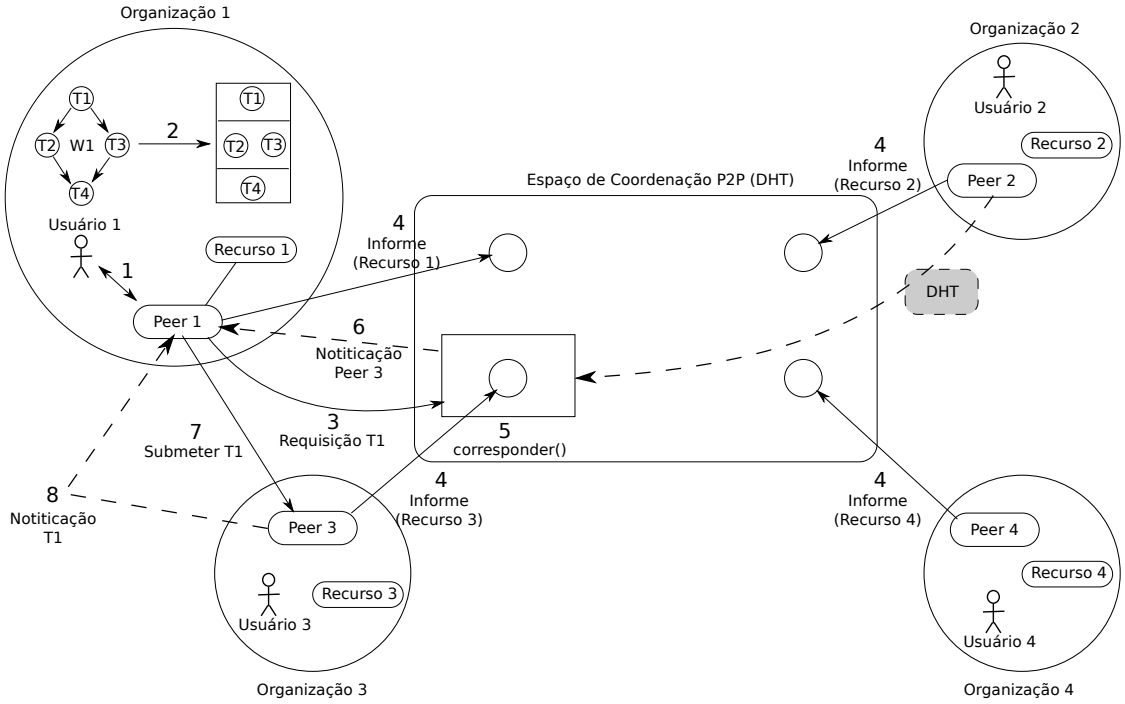

Figura 3.6: Funcionamento de uma rede *peer-to-peer* estruturada em uma grade computacional (RANJAN; RAHMAN; BUYYA, 2008).

Um exemplo do processo de escalonamento de tarefas, fornecimento de recursos e coordenação está descrito na figura 3.6. Em (1), o usuário submete o *workflow* $W1$ para o *peer* local. O *peer* 1 separa as tarefas em $W1$ de acordo com a precedência entre elas (2). Em seguida (3), o mesmo *peer* escolhe a tarefa com maior *rank* (T1) e publica no espaço distribuído uma requisição pelos recursos que a tarefa necessita. No instante (4), todos os *peers* publicam seus informes de recursos disponíveis no espaço distribuído. O informe publicado pelo *peer* 3 corresponde à requisição publicada pelo *peer* 1 (5). O *peer* 1 recebe a mensagem de correspondência de recurso (6) e submete a tarefa 1 ao *peer* 3, que executa a tarefa e retorna o resultado (7). Os passos (4) a (8) se repetem até a conclusão de $W1$.

Uma avaliação de desempenho foi feita através dos simuladores GridSim (BUYYA; MURSHED, 2002) e PlanetSim (AHULLO; LOPEZ, 2008). O primeiro estudo feito considera a definição de diversas filas para os componentes da grade, apresentando assim uma modelagem analítica da mesma. Os autores adotam um modelo de *workflow fork-join*, no qual uma tarefa origina várias tarefas paralelas que concentram resultados em uma outra tarefa. Esse padrão de divergência e convergência de valores pode ser repetido várias vezes em uma instância. Também assumem que as tarefas são de intensa computação, desprezando as dependências de dados entre as mesmas. Na análise realizada, vários aspectos como o número de troca de mensagens por quantidade de requisições são analisados.

O segundo estudo realizado, também nos simuladores, compara a solução proposta na qual os *peers* cooperam trocando mensagens no espaço DHT e outra na qual cada peer obtém dados dos recursos alheios através de um sistema de descoberta da grade e não executa negociação, simplesmente escolhe o primeiro recurso disponível. O resultados mostram que a abordagem defendida diminui o tempo de execução médio dos *workflows* e também reduz o número de médio de execuções de tarefas por processador. As duas análises comprovam que a solução é menos onerosa em termos de quantidade de trocas de mensagens por utilizar uma rede *peer-to-peer* estruturada e que o escalonamento cooperativo leva à uma distribuição de carga melhor, além de apresentar melhores tempos de execução para as instâncias de *workflows* submetidas.

A abordagem apresentada possui fundamentação teórica devido ao uso da DHT, mas como não é fácil modelar serviços desta forma, não se aplica à grades baseadas no paradigma de invocação de serviços. Os recursos são representados através de valores em um espaço linear, como a frequência de um processador e quantidade de memória. A funcionalidade de um serviço não pode ser representada utilizando o mesmo formalismo. Por exemplo, não há como mapear a descrição de um serviço que fornece um algoritmo de sequenciamento genético para valores de uma equação em um sistema linear.

Outra característica que torna a abordagem desaconselhável para grades de serviços diversos é a necessidade de estabelecer um sistema de reserva de recursos. Quando a descrição dos requisitos é feita em termos de processamento e memória, realizar a reserva de recursos é uma atividade direta. Entretanto, a reserva de recursos não é trivial quando os requisitos são descritos em termos de funcionalidade de serviços e atributos de QoS associados (YU; BUYYA, 2006b). Os requisitos de processamento e memória estão relacionados com a implementação do serviço. Na grade computacional, podem existir serviços com a mesma interface, mas com detalhes de implementação diferentes, sendo que essa discrepância pode levar a requisitos equi-

vocados informados pelo usuário.

3.4.2.3 Triana

Triana é um ambiente visual de análise de *workflow* desenvolvido pela Universidade de *Cardiff* (TAYLOR; SHIELDS; WANG, 2004) (TAYLOR et al., 2007). Seu foco é o suporte de serviços oriundos de vários ambientes diferentes, tais como redes *peer-to-peer* e grades computacionais, integrando vários tipos de *middleware*. Para permitir o suporte aos diferentes tipos de tecnologia, o Triana possui uma camada chamada GAT (*Grid Application Toolkit*), que consiste de uma coleção de interfaces para ligação com diferentes tipos de *middleware*.

A interface de programação visual do Triana é representada por unidades. Aplicações são escritas através da conexão de unidades de uma paleta em uma área de trabalho. Além de já possuir uma vasta biblioteca de unidades, Triana também permite a criação de unidades personalizadas através da implementação de uma interface. Estruturas de controle como laços são fornecidas para permitir o controle de execução dos fluxos, sendo implementadas também como unidades. Desta forma, a abstração de unidades permite ao desenvolvedor criar fluxos de execução complexos utilizando um conjunto pequeno de unidades simples.

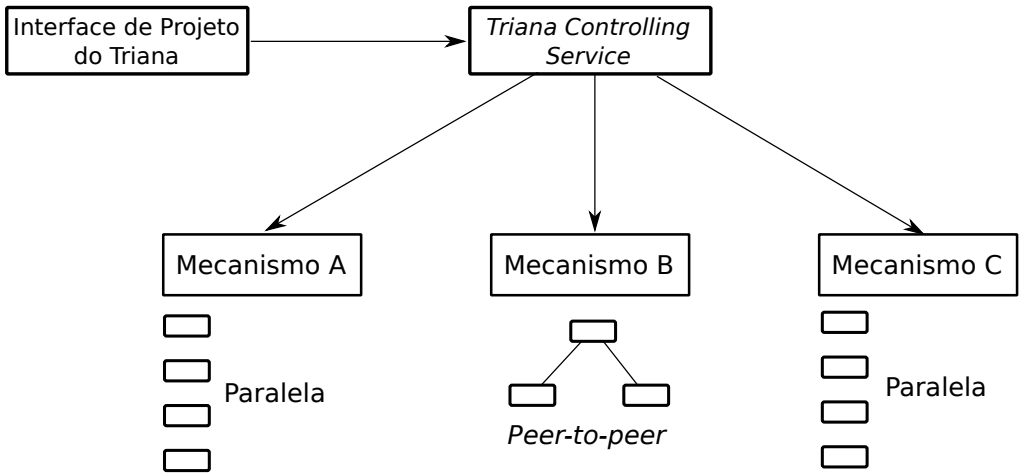

Figura 3.7: Esquema de funcionamento do Triana.

Clientes podem se conectar em um serviço de controle do Triana (TCS, *Triana Controlling Service*) e submeter instâncias de *workflows* projetadas na sua estação de trabalho. Cada TCS interage com um mecanismo Triana e cada mecanismo fornece um serviço, sendo capaz de executar localmente grafos de tarefas parciais ou completos. O mesmo mecanismo pode distribuir as tarefas entre outros servidores de acordo com uma política de distribuição definida pelo usuário no subgrafo. A política de distribuição é baseada no conceito de grupos de unidades e duas políticas de distribuição foram implementadas, uma chamada paralela e outra

peer-to-peer. Ambas políticas distribuem cada unidade no grupo para hospedeiros separados, entretanto enquanto o mecanismo *peer-to-peer* assume que dados intermediários são transferidos entre hospedeiros, não há comunicação entre as tarefas na política paralela. O esquema descrito é mostrado na figura 3.7.

Para a equipe de desenvolvimento do Triana, o principal objetivo no desenvolvimento do sistema foi criar um ambiente gráfico de fácil utilização que pudesse se conectar à diferentes *middlewares* para execução de tarefas. Esse objetivo foi atingido, sendo que atualmente o sistema suporta *middleware* estabelecidos como o WSRF para grades e o JXTA (OAKS; GONG, 2002). Entretanto, o escalonamento das unidades para os *peers* ou hospedeiros que formam um conjunto de recursos acessível por um mecanismo não é padronizado. Ou seja, cabe ao usuário ou ao administrador do mecanismo definir como um grupo de unidades é escalonado aos *peers*. Por exemplo, na literatura (TAYLOR et al., 2007), dois estudos de caso apresentados consistem da distribuição de serviços de grade e a integração de aplicações legadas. Em ambos os casos, a política de escalonamento é determinada pelos administradores do ambiente. A conclusão é que o Triana é um *framework* que pode ser utilizado para implementar soluções para gerência de *workflows* em grades, não definindo uma política de escalonamento fixa e permitindo personalização por parte das entidades que formam a grade.

3.4.2.4 ASKALON

O ambiente de desenvolvimento de aplicações para grades ASKALON (FAHRINGER et al., 2005) (QIN; FAHRINGER, 2008) tem como objetivo final fornecer uma grade transparente para os desenvolvedores de aplicações. ASKALON permite a composição de *workflows* em uma linguagem gráfica baseada na UML, que no momento da submissão são transformados em definições na linguagem AGWL (*Abstract Grid Workflow Language*). Tal linguagem fornece um rico conjunto de componentes para expressar sequência, paralelismo, escolha e iteração na estrutura do *workflow*.

Em relação ao escalonamento, ASKALON possui cinco serviços principais na sua arquitetura. O serviço gerente de recursos é responsável pela negociação, reserva e alocação de recursos, além da implantação automática de serviços necessários para executar aplicações na grade. O escalonador é um serviço que determina mapeamentos eficientes das instâncias de *workflows* em grades utilizando heurísticas que se beneficiam dos serviços de previsão de desempenho e gerência de recursos. O serviço mecanismo de execução trata da tolerância à falhas e da definição de pontos de restauração para permitir a migração de tarefas. O serviço de análise de desempenho permite a detecção de gargalos durante a execução de *workflows*. O

serviço de previsão de desempenho estima o tempo de execução de *workflows* através de uma fase de treinamento e de métodos estatísticos utilizando o serviço de análise de desempenho.

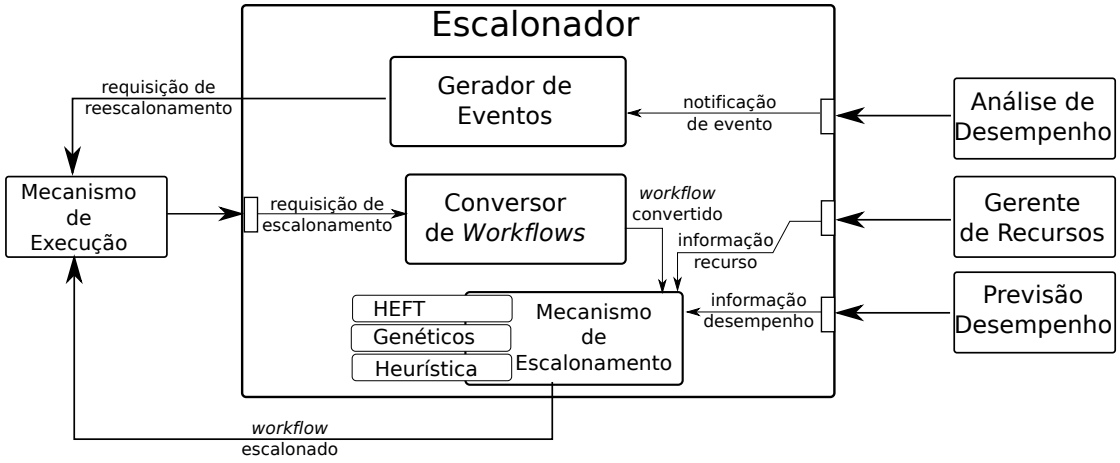

Figura 3.8: Arquitetura de escalonamento do ASKALON (FAHRINGER et al., 2005).

Os serviços de escalonamento são organizados na arquitetura da figura 3.8. Os serviços mecanismo de execução, conversor de *workflows* e mecanismo de escalonamento definem um esquema inicial baseado na utilização de uma heurística. Existem três heurísticas implementadas, porém o ambiente é extensível neste aspecto. O mapeamento inicial é utilizado para iniciar a execução do *workflow*. À medida que a execução prossegue, os serviços de monitoramento de desempenho e o gerente de recursos ativam eventos que são alimentados nos mecanismos de execução e escalonamento com o objetivo de adaptar a execução às mudanças no ambiente. Por exemplo, o serviço de análise de desempenho pode realizar que um determinado recurso não está atendendo as necessidades de uma tarefa, disparando um evento que indica ao mecanismo de execução que a tarefa deve ser migrada.

A decisão inicial sobre o escalonamento no ASKALON é feita em um conjunto restrito de serviços, sendo implementada de maneira centralizada. Entretanto, os serviços que disparam os eventos e o próprio sistema de tratamento de eventos são distribuídos através da grade em vários hospedeiros, sendo que cada domínio pertencente à grade tem uma réplica própria desses serviços. A opção por uma estrutura híbrida não foi feita buscando apenas diminuir gargalos, mas sim considerando que a natureza distribuída dos eventos emitidos é melhor tratada adotando uma arquitetura distribuída.

3.4.2.5 Comparação entre as Soluções Descentralizadas

A tabela 3.2 apresenta as principais informações sobre as soluções apresentadas. No quesito de descentralização, o SwinDew e a solução apresentada na seção 3.4.2.2 apresentam

maior grau de distribuição, adotando por completo o paradigma *peer-to-peer*. O Triana e o Askalon possuem em algum momento do escalonamento um ponto de tomada de decisão centralizado. Em relação a requisitos de QoS do usuário, apenas nas soluções *Global Grids* e o ASKALON eles são considerados, sendo que nenhuma solução aceita um número personalizável de métricas, considerando apenas o custo e o tempo de execução. De todas as soluções estudadas, apenas a *Global Grids* não considera a orquestração de serviços como uma opção para definição de aplicações.

Tabela 3.2: Tabela comparativa.

Ambiente	Descentralização da arquitetura	QoS do usuário	Quantidade de métricas	Reserva de recursos	Paradigma das aplicações
SwinDeW	*Peer-to-peer*	Não	Não se aplica	Não	Tarefas e serviços
Global Grids	*Peer-to-peer*	Sim	Fixo	Sim	Tarefas
Triana	Híbrida	Não	Não se aplica	Não	Tarefas e serviços
ASKALON	Híbrida	Sim	Fixo	Sim	Tarefas e serviços

Duas das soluções fazem uso da reserva de recursos para garantir o funcionamento de suas estratégias: *Global Grids* e ASKALON. Reserva de recursos consiste em dividir o acesso ao recurso em fatias de tempo e distribuir tais intervalos de maneira exclusiva entre os interessados em utilizar o recurso. Por exemplo, em uma grade computacional, um processo pode conseguir reservar o uso de um *cluster* por cinco horas. Desta forma, o trabalho do escalonador é facilitado, pois há garantias que depois da alocação do recurso, quando ele for utilizado será de forma exclusiva. Entretanto, um sistema de reserva de recursos pode ser considerado intrusivo pelos participantes da grade. Enquanto um recurso está reservado para uma requisição externa, um usuário local de alta prioridade pode necessitar fazer uso do recurso com urgência. Sendo assim, uma técnica de escalonamento que necessite da garantia de reserva de recursos tem flexibilidade menor e atende um público limitado de instituições interessadas em formar grades computacionais.

3.5 Conclusão

Neste capítulo apresentamos os aspectos gerais de *workflows* e redes *peer-to-peer*, além de suas relações com grades computacionais. Sobre os *workflows*, a arquitetura geral de um sistema de gerenciamento foi discutida, com ênfase na técnica centralizada mais utilizada com sucesso em projetos recentes, os algoritmos genéticos. As duas abordagens predominantes para a criação de redes sobrepostas *peer-to-peer* foram detalhadas, cada uma com suas vantagens e desvantagens, além de como os conceitos de grades computacionais e computação

peer-to-peer se alinham.

Um conjunto de soluções que faz uso de *workflows* para definição de aplicações em grades computacionais foi apresentado, com o foco na descentralização do escalonamento. Das soluções levantadas, entretanto, nenhuma consegue fornecer uma grade computacional com suporte à execução de *workflows* de serviços e métricas de QoS de maneira totalmente descentralizada sem a necessidade do emprego de reserva de recursos.

No próximo capítulo, é descrita uma proposta que busca apresentar um grau de descentralização ao passo que não impõe as restrições necessárias para aplicar reserva de recursos nas grades computacionais.

4 O Ambiente de Execução Proposto

4.1 Funcionamento Geral da Proposta

Este capítulo apresenta em detalhes o ambiente proposto neste trabalho. O restante do capítulo se organiza da maneira descrita a seguir. A seção 4.2 descreve os termos utilizados para a definição de serviços, a estrutura de grade computacional considerada na elaboração deste trabalho e um levantamento bibliográfico das métricas de QoS relacionadas à serviços de grades computacionais. Na seção 4.3, o papel do usuário no ambiente é detalhado e explica-se quais ações são necessárias para a submissão de um *workflow* ao ambiente proposto. A forma como o sistema executa as instâncias de *workflows* submetidas é assunto da seção 4.4. Nela, primeiro é explicada a transição de uma etapa do *workflow* para a seguinte levando em conta os parâmetros de QoS fornecidos pelo usuário (seções 4.4.1 e 4.4.2). Depois, uma visão geral da execução global de um *workflow* é apresentada (seção 4.4.3).

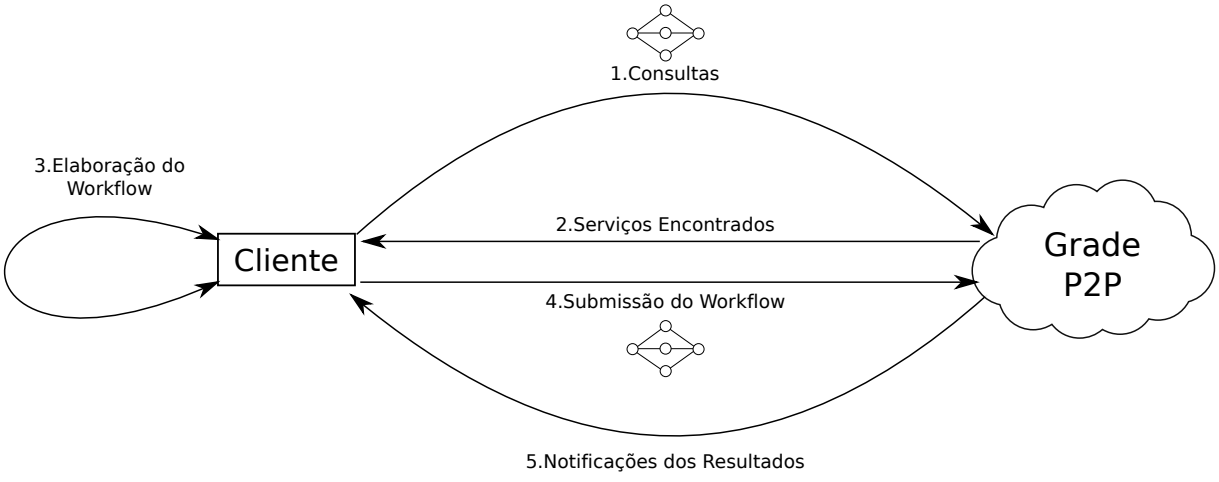

Figura 4.1: Modelo geral do funcionamento da *workflows*.

A visão geral do funcionamento está descrita na figura 4.1. O usuário interage com uma aplicação cliente, que por sua vez troca informações com os *peers* da grade computacional. O primeiro passo consiste na submissão por parte do usuário de um conjunto de consultas que correspondem aos serviços que deseja encontrar na grade para a composição de um *workflow*.

São retornados aos usuários os serviços encontrados que satisfazem as consultas submetidas. A partir desse resultado, o usuário pode desenvolver a versão final do *workflow* de acordo com os serviços existentes na grade. A versão final do *workflow* é submetida à grade *peer-to-peer* para execução e à medida que a execução prossegue a aplicação cliente recebe notificações sobre as etapas concluídas. As etapas mencionadas anteriormente são detalhadas nas próximas seções deste capítulo.

No decorrer do capítulo, um exemplo de *workflow* é utilizado como recurso didático para explicar os conceitos relacionados ao ambiente. Na figura 4.2 está o exemplo inicial de *workflow* usado. Ele é baseado em uma aplicação integrante do conjunto de aplicativos *NASA Supercomputing Parallel Benchmarks* (WEERATUNGA et al., 1994). Tal aplicação realiza simulações relativas a área de mecânica de fluidos. A mesma aplicação será utilizada no capítulo 5 para validação da proposta. Mais detalhes detalhes sobra a aplicação podem ser encontrados na seção 5.2.2. Por enquanto, é necessário saber que uma instância inicial do *workflow* pode ser formada por três serviços: *SPLaunch*, *SPWorker* e *SPReport*.

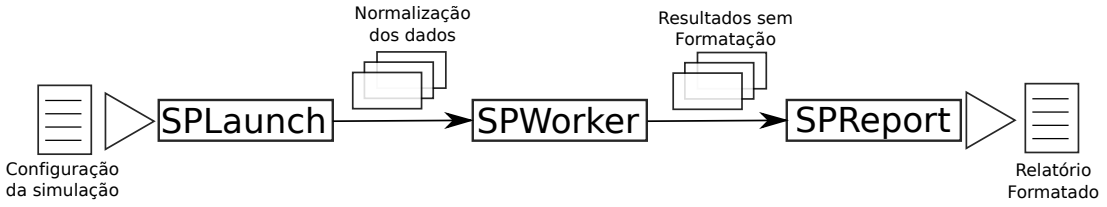

Figura 4.2: Exemplo inicial de *workflow*.

O serviço *SPLaunch* recebe do usuário um arquivo com as configurações para realização da simulação, dentre elas a descrição da superfície simulada e as forças em ação. O serviço se encarrega de normalizar os dados fornecidos na configuração para a execução da simulação, que é realizada pelo serviço *SPWorker*. O resultado da simulação é então submetido ao serviço *SPReport*, que retira ambiguidades e dados repetidos, retornando ao usuário um relatório formatado sobre o experimento.

4.2 Modelo da Grade Adotado

Nesta seção é descrito o modelo de grades computacionais adotado para este trabalho. Entretanto, antes de apresentar a arquitetura das grades consideradas, é importante apresentar os conceitos adotados para definição de serviços a fim de evitar ambiguidades. Desta forma, a seção 4.2.1 realiza a conceitualização adequada, enquanto na seção 4.2.2 a infraestrutura de grades considerada é descrita, com foco na formação da rede *peer-to-peer* sobreposta à grade. Por fim, a seção 4.2.3 discute as abordagens para aplicação de requisitos de QoS.

4.2.1 Definição de Serviços

Uma entidade fornece um serviço para outra quando a primeira entidade realiza um conjunto de ações para o benefício da segunda (PREIST, 2007). Para realizar tais ações, um serviço pode oferecer uma ou mais operações aos seus clientes.

Um serviço abstrato é aquele que atende as necessidades do usuários mas não tem existência comprovada. Um serviço concreto tem existência comprovada, com interface de suas operações e identificador definidos. No escopo desta proposta, um serviço abstrato tem o formato de uma consulta que é submetida pelo usuário à grade. Os serviços concretos são os resultados retornados pela busca de serviços abstratos. Para decidir se um determinado serviço concreto corresponde a um serviço abstrato, a consulta do abstrato é realizada na descrição do serviço concreto, no qual suas operações estão detalhadas. Existem vários formalismos para descrições de serviços e consultas (LAUSEN et al., 2007), dentre eles a lógica descritiva e as tecnologias de *web* semântica derivadas. Neste trabalho, para a descrição de serviços é adotada a linguagem WSDL, sendo que os serviços são identificados por uma URI (W3C, 2009a). As consultas são baseadas em palavras-chaves interligadas pelos conectivos lógicos AND e OR. De maneira similar, um *workflow* abstrato é aquele cujas invocações são serviços abstratos e um *workflow* concreto é aquele cujas invocações já são serviços existentes na grade.

Para exemplificar os conceitos de serviços concretos e abstratos, considere os serviços do *workflow* da figura 4.2. Um serviço abstrato seria a consulta [("fluidos"AND "mecânica") AND ("configuração"OR "SPLaunch")], que submetida à grade retornaria o serviço concreto [http://www.great.ufc.br/jmarcelo/bechmarkNASA/SPLaunch].

4.2.2 Organização dos *Peers*

O modelo descrito nesta seção define a infraestrutura de grade computacional necessária para a implantação da proposta desta dissertação. A figura 4.3 é uma representação conceptual da grade utilizada. Nessa figura, um hospedeiro é uma máquina que faz parte de uma instituição integrante da grade computacional.

A arquitetura do sistema é baseada no conceito de grade federativa (RANJAN; RAHMAN; BUYYA, 2008), na qual um hospedeiro ou conjunto de hospedeiros funcionam como um ponto de acesso aos recursos de um domínio administrativo. No caso da definição original de grade federativa, o ponto de acesso hospeda um serviço de submissão de tarefas que é acessado pelo escalonador da grade. Como a solução deste trabalho utiliza a invocação de serviços ao invés da submissão de tarefas, o ponto de acesso hospeda vários serviços que possuem funcio-

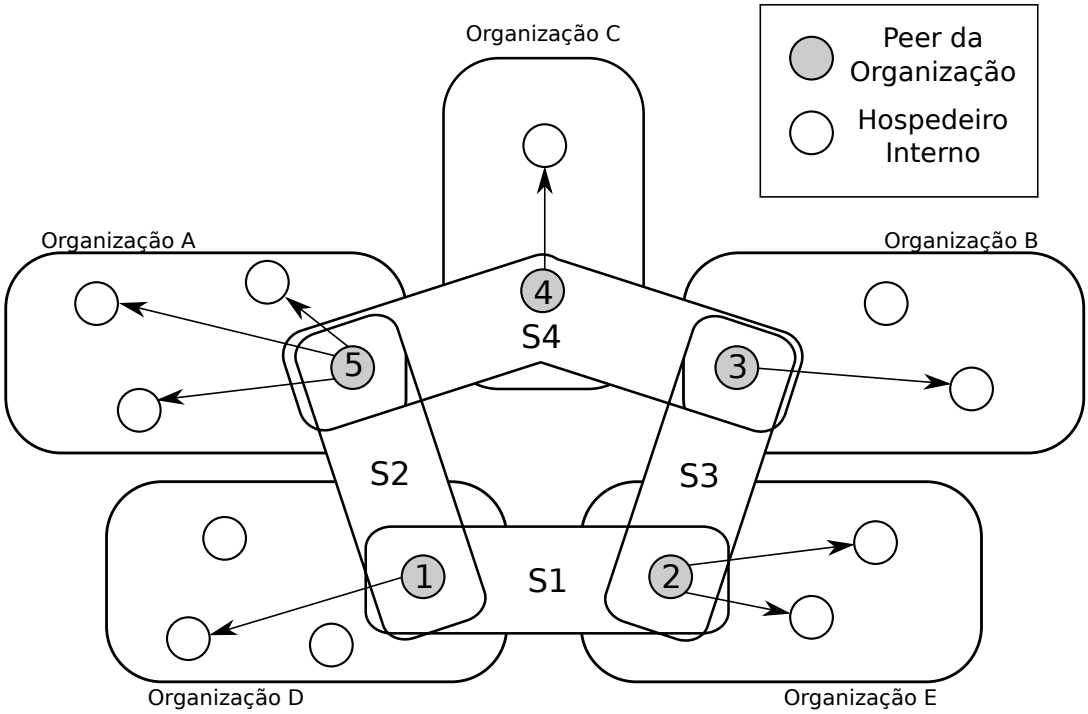

Figura 4.3: Organização do modelo de Grades Computacionais adotado.

nalidade definida e podem ser utilizados pelos usuários da grade na composição de *workflows*. O modelo assume o uso de uma linguagem aceita por todos os membros da grade para a definição das interfaces desses serviços. No caso deste trabalho, a linguagem utilizada é a WSDL (W3C Consortium, 2007).

Cada serviço invocado nos pontos de acesso faz uso dos recursos locais das organizações para atender as requisições. A decisão de quais outros hospedeiros serão utilizados é tomada pelo implementador do serviço invocado e o administrador do domínio. Os hospedeiros que funcionam como ponto de acesso mantêm um repositório com informações sobre os estados dos recursos da organização, tais dados são usados para definir os valores em cada *peer* dos parâmetros de QoS descritos na seção 4.2.3.

Os pontos de acesso são os *peers* da rede *peer-to-peer* não-estruturada sobreposta à grade computacional. Os *peers* são agrupados de acordo com os serviços que possuem. Na figura 4.3 temos quatro serviços distribuídos por cinco organizações diferentes. Considerando o *workflow* da figura 4.2, o serviço S1 poderia representar o serviço *SPLaunch*, enquanto S2 representaria *SPWorker* e S4 representaria o serviço *SPReport*. No caso, S3 seria um serviço sem relação com o *workflow* em questão.

Para a manutenção da rede *peer-to-peer* são utilizados algoritmos para redes não-estruturadas descritos na literatura (LUA et al., 2005) (TANENBAUM; STEEN, 2007). Os

algoritmos específicos utilizados são os mesmos empregados pelo SwinDew (YANG et al., 2007). Estes algoritmos exigem que duas propriedades estejam presentes no modelo de grade. Primeiro, é necessário que cada *peer* armazene um mapeamento dos seus serviços em relação aos outros *peers*. Para cada serviço do *peer* existem referências aos *peers* portadores do mesmo serviço. Um exemplo desse mapeamento para a figura 4.3 está na tabela 4.1. Os algoritmos para a manutenção da rede *peer-to-peer* sobreposta atuam nessa tabela, atualizando-a quando necessário. A segunda propriedade é a existência de um sistema de troca de mensagens que permita a um processo executando em um *peer* realizar subscrições em notificações que podem ser emitidas por outros processos externos. Tal paradigma é relacionado ao padrão de projeto *Observer/Observable* (GAMMA et al., 1995) e é importante para permitir a comunicação assíncrona entre as entidades da grade. Na verdade, o bloqueio de processos de comunicação síncrona pode levar a queda de desempenho.

Tabela 4.1: Exemplo de mapeamentos de vizinhos nos *peers*.

Peer	Mapeamento
1	[S1:1,2][S2:1,5]
2	[S1:1,2][S3:2,3]
3	[S3:2,3][S4:3,4,5]
4	[S4:3,4,5]
5	[S4:3,4,5][S2:1,5]

Considera-se que os serviços oferecidos por uma organização à grade possuem funcionalidades que já eram importantes para a organização antes do ingresso na grade. Por exemplo, um departamento que realiza pesquisa na área de biologia possui um algoritmo de sequenciamento genético para as espécies que são alvo de estudo. Ao formar uma grade com laboratórios de outras instituições, o algoritmo é disponibilizado como um serviço provido pelo *peer* da organização. Como todos os membros da grade possuem área de pesquisa semelhantes, a probabilidade de outros *peers* terem serviços similares é alta. Uma interface comum pode ser definida pelos administradores para serviços similares, agrupando-os em um mesmo grupo de *peers* na grade. Essa abordagem é semelhante a utilizada em soluções de grades baseadas na invocação de serviços como o Taverna (OINN et al., 2004), que adotam *frameworks* que permitem aos administradores unir serviços de diferentes tecnologias em uma mesma interface.

4.2.3 Métricas de Qualidade de Serviço

Existem basicamente duas maneiras de considerar métricas de qualidade de serviço em *workflows* (YU; BUYYA, 2006b). A primeira opção é considerar restrições de QoS a nível de cada serviço componente do *workflow*. Por exemplo, em uma instância do *workflow* da figura

4.2, o usuário pode determinar que a execução do serviço *SPLaunch* não pode passar de dois minutos, que a invocação de *SPWorker* não deve ultrapassar os dez minutos e que invocação de *SPReport* deve durar mais que um minuto.

A segunda maneira consiste em considerar as restrições como globais, a nível da instância completa do *workflow*. Essa abordagem exige a presença de um processo monitor com visão geral da execução para garantir que a invocação de cada serviço componente respeite as restrições gerais da instância de *workflow*. Por exemplo, em uma instância do *workflow* da figura 4.2, o usuário pode determinar que a execução dos três serviços em série não pode ultrapassar os quinze minutos.

Como uma das metas deste trabalho é oferecer uma solução de execução descentralizada, a abordagem indicada é a primeira opção (YU; BUYYA, 2006b), pois ela permite que a decisão sobre qual entidade é responsável por cada componente seja tomada de maneira distribuída entre os *peers* da grade.

4.3 Submissão de *Workflows*

Nesta seção é descrita o papel do usuário na utilização do ambiente proposto. O pesquisador faz uso de uma aplicação cliente que permite a definição de *workflows* na forma de um grafo DAG, no qual cada nó representa uma invocação a ser feita e as arestas determinam o fluxo de dados entre os *peers* responsáveis pela invocação. São duas as etapas realizadas para a submissão de um *workflow*.

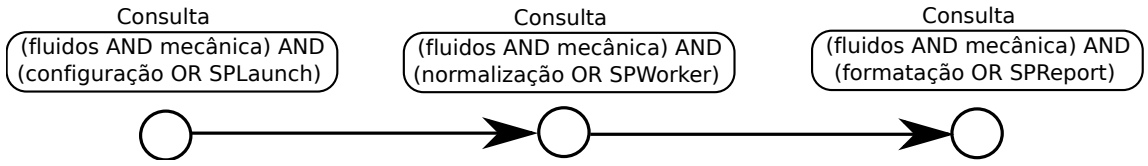

Figura 4.4: Representação abstrata do *workflow* com as consultas para cada etapa.

A primeira etapa consiste em estabelecer um grafo com serviços abstratos para cada etapa. A descrição de cada serviço é feita através da definição de uma consulta baseada em palavras chaves e os operadores lógicos AND e OR. Na figura 4.4, para efeito didático está descrito o modelo abstrato de um *workflow* com apenas três etapas. No caso, trata-se do *worflow* da figura 4.2. As consultas são submetidas à aplicação cliente, que por sua vez realiza uma busca na rede *peer-to-peer* para cada uma das consultas submetidas. As consultas da figura 4.4 são propagadas por toda a rede da figura 4.3 como demonstrado na figura 4.5 e aqueles *peers* que possuam algum serviço que combine com as consultas retornam sua descrição para o cliente.

No exemplo, os *peers* com os serviços *SPLaunch*, *SPWorker* e *SPReport* atendem as consultas do usuário, sendo que S3 representa um serviço que não combina com nenhuma das consultas.

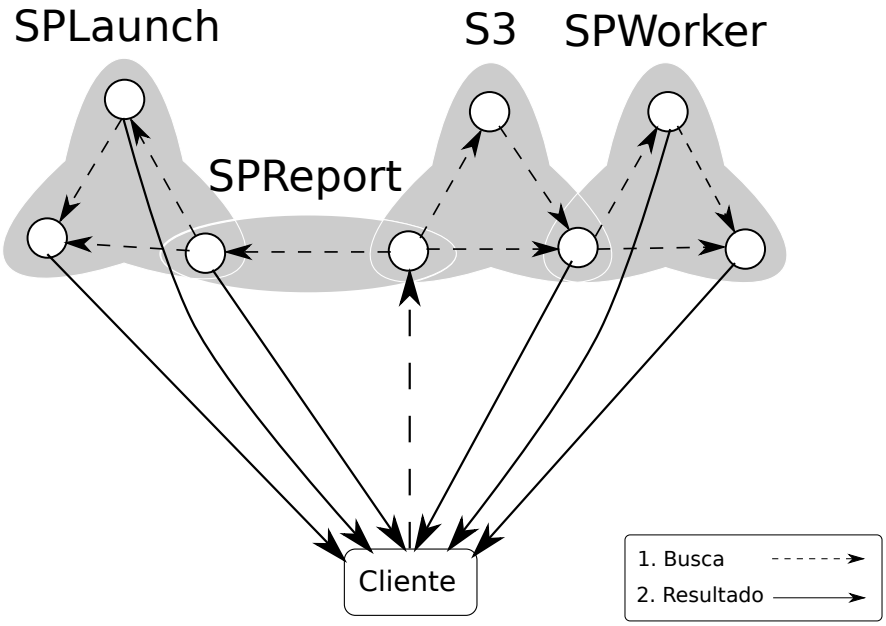

Figura 4.5: Realização da busca contendo as consultas para cada serviço.

Na segunda etapa, após receber o resultado da busca com todos os serviços concretos candidatos, cabe ao usuário definir o *workflow* concreto a partir dos serviços retornados. A figura 4.6 apresenta o modelo concreto para o exemplo, sendo que para finalizar a definição o usuário deve informar os dados de entrada e as preferências de QoS.

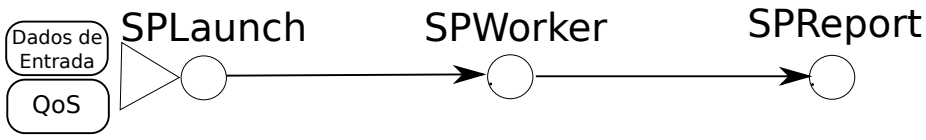

Figura 4.6: Representação concreta do *workflow* com os serviços existentes na grade.

Na figura 4.7 a execução do *workflow* da figura 4.6 é realizada considerando os requisitos de QoS informados de acordo com a estratégia descrita na seção 4.4.1. Todo o processo de negociação e escolha de *peers* é abstraído da visão do usuário, porém à medida que as invocações são finalizadas, cada *peer* retorna ao cliente os valores intermediários ou uma referência para um serviço de transferência de dados que permita a recuperação dos mesmos valores intermediários. Os dados intermediários são importantes para o usuário no caso de falha de execução de alguma etapa do *workflow*.

Existe a possibilidade dos serviços retornados pela primeira etapa não serem compatíveis. Por exemplo, alterando a natureza do exemplo considerado, o serviço *SPLaunch* poderia

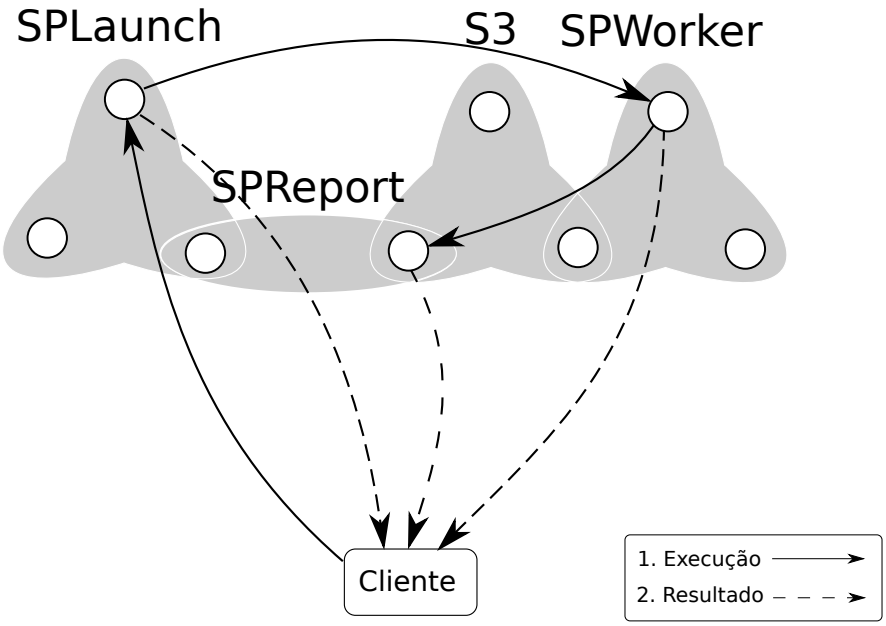

Figura 4.7: Execução do *workflow*.

ter como dados de saída um tipo que não é equivalente à entrada do serviço *SPWorker*. É preciso utilizar uma técnica de mediação para resolver esse impasse. Nesta situação, o usuário pode proceder de duas maneiras.

A primeira alternativa seria implementar um serviço que transforme um tipo de dados em outro e hospedá-lo em um dos *peers* da grade. Desta forma, bastaria intercalar o novo serviço entre os serviços anteriores e a situação estaria resolvida. Entretanto, essa solução apresenta dificuldades, pois exige o desenvolvimento de um novo serviço e acesso administrativo à um *peer*.

A segunda opção é mais simples, porém só funciona no caso das informações trocadas estejam no formato XML (W3C, 2009b). Neste caso, além de informar em cada nó do *workflow* o serviço a ser invocado, o usuário precisa acrescentar uma declaração de folha de estilo XSLT (CORCHO; LOSADA; BENJAMINS, 2007). Os dados de saída da invocação e a folha de estilo são fornecidas a um *parser* de XSLT, que realiza a transformação de dados antes de encaminhá-los para o *peer* seguinte.

4.4 Execução de *Workflows*

Nesta seção, é explicado como procede a execução de um *workflow* DAG submetido por um usuário. Primeiro, na seção 4.4.1 é mostrado como a execução de uma etapa e a negociação de um *peer* para a próxima etapa são intercaladas. Em seguida, a seção 4.4.2 detalha

como um *peer* seleciona um sucessor baseado nas métricas de QoS. O foco nessas duas primeiras seções é detalhar o relacionamento entre *peers* responsáveis por etapas adjacentes em um *workflow*. Por último, a seção 4.4.3 apresenta como os conceitos das seções anteriores se completam na execução de um *workflow* DAG completo.

4.4.1 Estratégia de Escalonamento

A estratégia de escalonamento adotada é baseada no paradigma apresentado pelo SwinDew, introduzido no capítulo 3. No SwinDew original, a alocação completa dos *peers* ocorre antes da execução de um *workflow*. A negociação para decidir qual *peer* será o escolhido para executar a requisição dentro de um grupo considera a carga média de cada *peer*. Um parâmetro determina qual a duração do período de tempo que deve ser analisado no passado para decidir a carga média, sendo o valor padrão equivalente a um dia. Nos casos em que o tempo para executar uma requisição seja uma porcentagem relevante em relação ao parâmetro escolhido, a alocação feita no momento inicial pode não ser mais adequada. Por exemplo, na figura 4.8, um *workflow* constituído por três requisições com serviços diferentes exige em média um intervalo de tempo de um dia para invocação do primeiro serviço S1. No momento da alocação, são escolhidos os *peers* de cada grupo com a menor influência na carga do sistema. Porém, ao final da execução da primeira invocação, os dois outros *peers* escolhidos para as requisições seguintes (serviços S2 e S3) podem não ser mais as escolhas ideais já que a alocação para um *workflow* não impede que um *peer* atenda requisições de outros *workflows*.

A inovação apresentada neste trabalho consiste em não alocar todos os *peers* necessários antes de iniciar a execução das invocações. Um conjunto inicial é alocado e a execução das invocações designadas é iniciada. Após o término da execução, mais um conjunto de *peers* é alocado e o processo se reinicia. Por exemplo, na figura 4.9, o *peer* de instanciação inicia a alocação de um *peer* para atender a invocação do serviço S1. Após finalizar a execução do serviço, o *peer* escolhido se responsabiliza por alocar mais um *peer* para a execução da invocação do serviço S2. No caso da execução no primeiro *peer* for demorada, a alocação do segundo *peer* leva em conta o estado da grade ao fim da execução, considerando as informações mais recentes do contexto de execução. As requisições do *workflow* são alocadas à medida que a execução do mesmo avança. No exemplo da figura 4.9, uma nova requisição é alocada a cada invocação realizada. Uma variação dessa estratégia consiste em alocar inicialmente N *peers*, executar as requisições correspondentes e depois alocar mais N *peers*, prosseguindo dessa forma por todo o *workflow*. A cardinalidade do conjunto alocado após a conclusão das execuções pendentes é um parâmetro que pode ser aprimorado pelo usuário na medida que o *workflow* é desenvolvido.

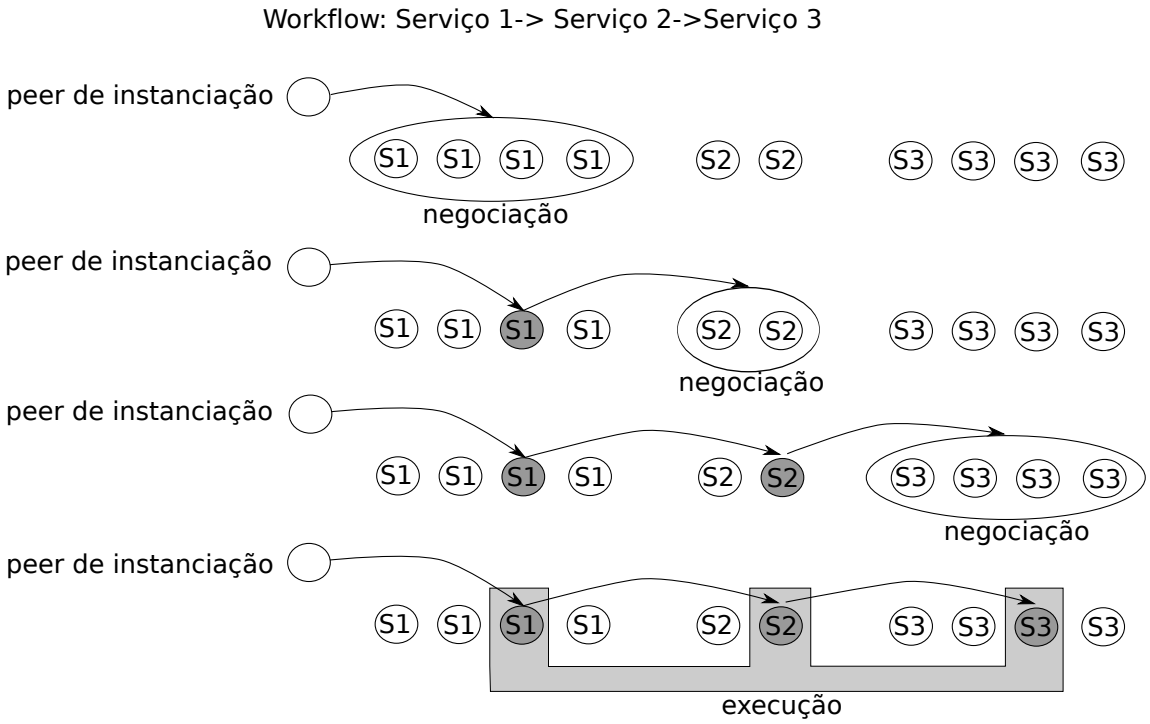

Figura 4.8: Alocação de Tarefas pelo SwinDeW para um *Workflow* Simples.

O objetivo dessa inovação é permitir uma execução global do *workflow* adaptável a mudanças no contexto de execução, fazendo com que as informações utilizadas nas fases de negociação reflitam o estado atual da grade. No SwinDew, a alocação apenas no momento da submissão pode invalidar as escolhas feitas para as etapas posteriores de um *workflow*, já que a grade não é exclusiva e o estado dos *peers* pode ser alterado por submissões de usuários distintos.

Uma possível desvantagem seria a necessidade de cada *peer* precisar manter informações não apenas sobre sua requisição, mas também sobre as todas requisições posteriores. Entretanto, a quantidade de informação extra mantida em relação ao SwinDew original é equivalente aos dados de uma instância do *workflow*.

4.4.2 Negociação de Sucessores

Na etapa de negociação é decidido qual *peer* atenderá uma requisição dentro de um grupo de *peers* que possuem o mesmo serviço. A Figura 4.10 apresenta uma visão geral da negociação para a escolha de um *peer*. O processo avança da esquerda para a direita no decorrer do tempo. A negociação é realizada pelo *peer* de instanciação ou por um *peer* que tenha acabado de realizar a execução de uma invocação do *workflow*. Como demonstrado na figura 4.9, a negociação utiliza informações sobre o estado de um grupo de *peers* candidatos e aplica os

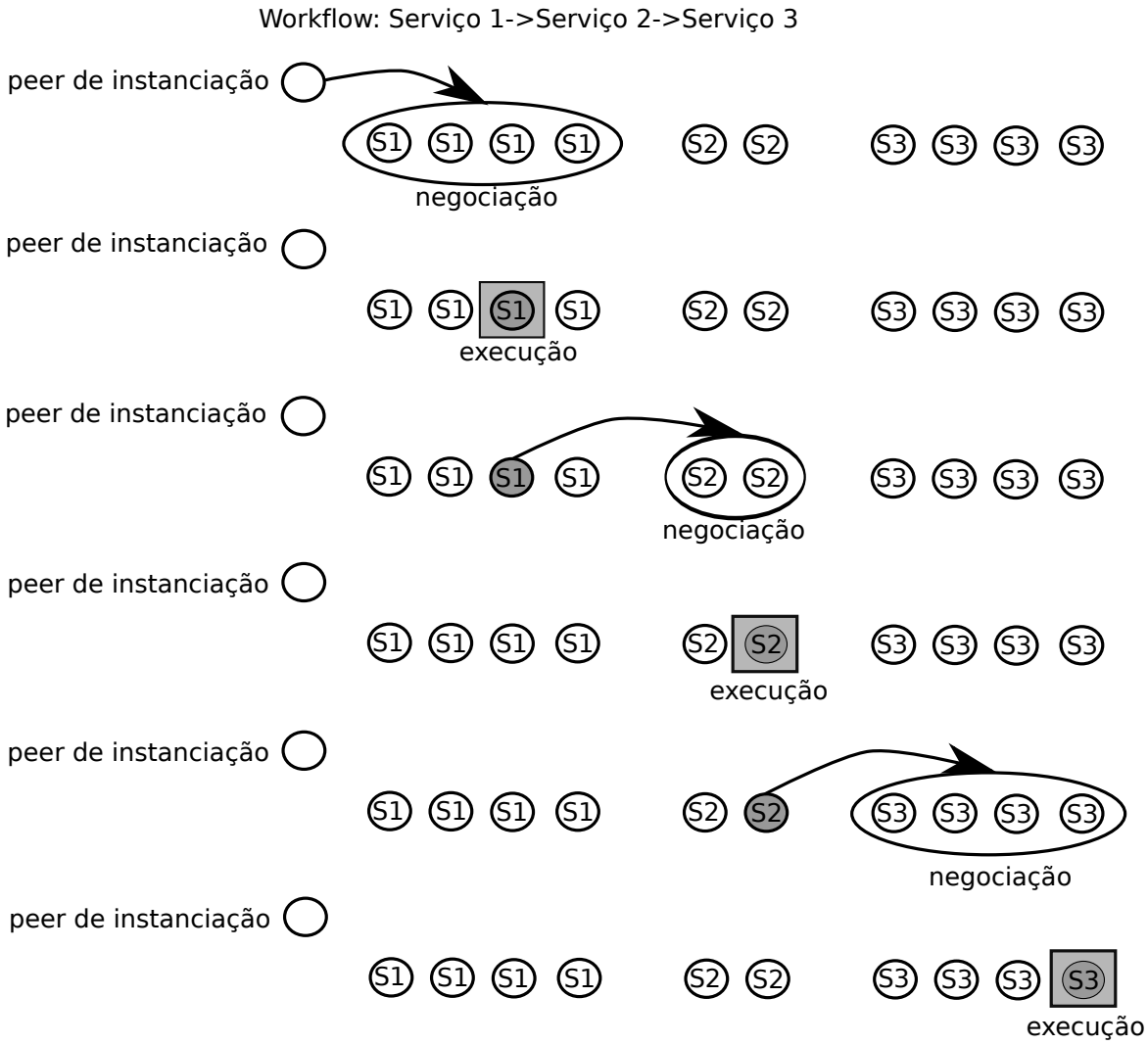

Figura 4.9: Alocação de Tarefas proposta para um *Workflow* Simples.

requisitos de QoS informados pelo usuário no *workflow* concreto para definir o próximo *peer*. A rede *peer-to-peer* é formada por grupos de *peers* organizados de acordo com os serviços que possuem. A primeira seleção que ocorre ao atender uma requisição é a descoberta de qual grupo possui serviços apropriados. Este processo é idêntico ao utilizado do SwinDeW.

A segunda e terceira etapa aplicam um processo semelhante que leva em consideração a prioridade de cada métrica. No caso da segunda etapa, as métricas consideradas são aquelas relacionadas aos requisitos administrativos da grade como um todo, por exemplo o balanceamento de carga. A terceira etapa utiliza as métricas importantes para o usuário finais. O grau de satisfação de cada uma das métricas por um serviço de um *peer* precisa ser mapeado para um valor numérico. Para métricas como custo de execução ou tempo de execução, o mapeamento é imediato. Entretanto, há casos em que o mapeamento não é direto, como na métrica de segurança. Seja qual for o mapeamento escolhido, ele precisa ser adotado por todas as instituições

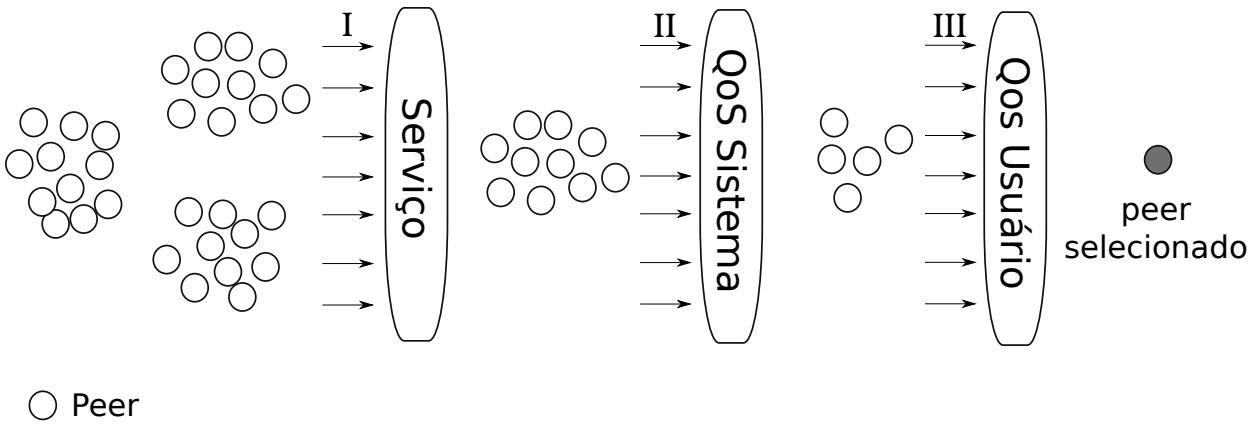

Figura 4.10: Etapas de negociação.

que possuem *peers* na grade.

O processo de seleção recebe como entrada do usuário ou do administrador uma tupla no formato da equação 4.1, sendo q o número de métricas avaliadas. Cada valor de p_i releva a prioridade da métrica i em relação às outras. Por exemplo, considerando apenas as métricas de custo de execução e tempo de execução, um vetor no formato $[0.75, 0.25]$ significa que a prioridade do custo para o usuário é três vezes maior que o prioridade do tempo de execução, ou seja, o pesquisador prefere uma execução mais lenta do que uma que acarreta maior débito de créditos. O valor M é um patamar que serve para indicar a porcentagem dos *peers* que passam para a próxima fase, sendo seu papel explicado nos próximos parágrafos.

$$< [p_1, p_2, p_3, ..., p_q], M > \text{,com} \sum_{i=0}^{q} p_i = 1 \text{ e } 0 <= M <= 1. \tag{4.1}$$

As informações sobre os estados dos *peers* que são candidatos para a próxima execução também servem como entrada para o processo de seleção, já que são necessárias para o cálculo dos valores de cada métrica considerada. As métricas são calculadas e organizadas em um vetor como na equação 4.2. O valor m_j^i representa o valor da métrica i para o *peer* candidato j.

$$[m_j^0, m_j^1, m_j^2, ..., m_j^q], \text{ para todo } \textit{peer} \text{ j em análise.} \tag{4.2}$$

Após a reunião das informações fornecidas pelo usuário e pelos *peers* candidatos, os dados são reestruturados para permitir a escolha do próximo *peer*. As métricas são normalizadas de acordo com a equação 4.3, na qual $max_{(j=n)}(m_j^i)$ é o valor máximo da métrica no grupo de n *peers* analisado e $min_{(j=n)}(m_j^i)$ corresponde ao valor mínimo.

$$[[m_j^0],[m_j^1],[m_j^2],...,[m_j^q]] \text{ , com} \qquad [m_j^i] = \begin{cases} 1 & \text{se } m_j^i = max_{(j=n)}(m_j^i) \\ \frac{m_j^i - max_{(j=n)}(m_j^i)}{max_{(j=n)}(m_j^i)} & \text{caso contrário.} \end{cases} \tag{4.3}$$

Outro ajuste necessário é adequar os valores das métricas de acordo com a natureza da mesma, conforme a equação 4.4. Isso é necessário porque algumas métricas como o custo de execução, sempre devem ser minimizadas. Já outras métricas, como a reputação de um *peer*, são melhores quanto maior for o seu valor.

$$[m_j^i] = \begin{cases} [m_j^i] & \text{caso minimização de métrica.} \\ 1 - [m_j^i] & \text{caso maximização de métrica.} \end{cases} \tag{4.4}$$

Os pesos e os valores das métricas são utilizados para calcular o índice k_j na fórmula 4.5 para cada *peer* j. Aqueles com menor valor para o índice são os *peers* do conjunto candidato com maior probabilidade de atender as restrições de QoS do usuário. No caso da minimização de métricas, para uma dada prioridade p_i, o *peer* com menor contribuição relativa de $p_i * [m_j^i]$ ao valor final de k_j será aquele com menor valor de $[m_j^i]$. No caso da maximização, uma mesma prioridade p_i torna os *peers* com menores contribuições relativas de $p_i * [m_j^i]$ em k_j, aqueles com maiores valores originais de $[m_j^i]$, pois a equação 4.4 inverte tais valores.

$$p_1 * [m_j^1] + p_2 * [m_j^2] + p_3 * [m_j^3] + ... + p_q * [m_j^q] = k_j \tag{4.5}$$

$$\sum_{i=0}^{q} p_q * [m_j^q] = k_j \tag{4.6}$$

Os *peers* são então ordenados em ordem crescente de acordo com os índices k_j, em um vetor de n elementos. O número de aprovados para a próxima etapa é de tamanho igual ao maior inteiro menor que $n * M$, sendo que se o resultado da multiplicação for menor que 1, então o número de aprovados também será 1. É definido um conjunto com tamanho igual ao número de aprovados contendo os primeiros *peers* do vetor de candidatos ordenados por k_j. É aqui que surge a diferença no processo em relação as duas últimas etapas da negociação. Caso seja a segunda etapa, o conjunto de aprovados é passado adiante para a terceira etapa. Já na terceira e última etapa, um *peer* é escolhido de maneira aleatória do conjunto de aprovados, passando a ser o *peer* eleito para a próxima invocação. A escolha aleatória ocorre para evitar que a métrica com maior peso domine a execução, permitindo que os outros pesos também sejam considerados.

Para exemplificar o funcionamento do processo, considere um *peer* que necessite executar a segunda e terceira etapa da negociação e tenha como ponto de partida o conjunto de candidatos detalhado na tabela 4.2. Tal conjunto de candidatos consiste de *peers* que possuem o serviço necessário para a invocação da próxima etapa do *workflow*.

Tabela 4.2: Conjunto candidato de *peers* e as métricas de QoS associadas.

Peer	Custo	Tempo de Execução	Disponibilidade	Carga
Peer 1	10 créditos	900 segundos	50%	0.1
Peer 2	50 créditos	250 segundos	60%	0.7
Peer 3	70 créditos	100 segundos	80%	0.4
Peer 4	30 créditos	500 segundos	90%	0.2
Peer 5	80 créditos	90 segundos	100%	0.8
Peer 6	90 créditos	50 segundos	90%	0.5
Peer 7	100 créditos	20 segundos	70%	0.9
Peer 8	40 créditos	400 segundos	70%	0.6
Peer 9	60 créditos	200 segundos	90%	1.0
Peer 10	10 créditos	1000 segundos	50%	0.1

Como descrito, o primeiro passo consiste em normalizar os dados de acordo com as equações 4.3 e 4.4 . O resultado é apresentado na tabela 4.3. As métricas são então divididas em métricas administrativas e do usuário. No exemplo, só há uma métrica administrativa, a carga do *peer*. Neste caso, não é necessária a atribuição de pesos, o valor de k_j nesta etapa é determinado apenas pelo valor da métrica em questão.

Tabela 4.3: Métricas normalizadas.

Peer	Custo	Tempo de Execução	Disponibilidade	Carga
Peer 1	0.00	0.90	0.00	0.10
Peer 2	0.45	0.23	0.20	0.67
Peer 3	0.67	0.08	0.60	0.34
Peer 4	0.23	0.49	0.80	0.12
Peer 5	0.78	0.07	1.00	0.78
Peer 6	0.89	0.03	0.80	0.45
Peer 7	1.00	0.00	0.40	0.89
Peer 8	0.34	0.39	0.40	0.56
Peer 9	0.56	0.18	0.80	1.00
Peer 10	0.00	1.00	0.00	0.00

O administrador acaba estabelecendo apenas o parâmetro M de acordo com a equação 4.7. Com isso, temos os *peers* ordenados de acordo com k_j na tabela 4.4 (no caso apenas a carga é computada em k_j). Aplicando $M_{adm} * n$, com $M_{adm} = 0.5$ e $n = 10$, temos que os cinco *peers* com menores valores de k_j devem passar para a terceira e última fase. São eles os *peers* 1, 3, 4, 6, 10.

$$< [], M_{adm} = 0.5 > \text{ , apenas uma métrica para avaliação.} \tag{4.7}$$

Tabela 4.4: Valores de k_j calculados e ordenados na segunda etapa.

k_j	Valor
k_1	0.1
k_{10}	0.1
k_4	0.2
k_3	0.4
k_6	0.5
k_8	0.6
k_2	0.7
k_5	0.8
k_7	0.9
k_9	1.0

Na terceira etapa, o usuário deve fornecer os pesos para cada métrica. Sejam m_j^1 a métrica custo, m_j^2 a métrica tempo de execução e m_j^3 a disponibilidade, todas em relação a um *peer* j. O usuário fornece os pesos e o parâmetro de acordo com a equação 4.8, na qual $p_1 = 0.5$ representa o peso do custo e assim sucessivamente. A tabela 4.5 apresenta os valores das métricas apenas para os *peer* candidatos na segunda fase.

$$< [0.5, 0.25, 0.25], M_{user} = 0.4 > \text{ , três métricas para avaliação.} \tag{4.8}$$

Tabela 4.5: Métricas dos *peers* candidatos na segunda fase.

Peer	m_j^1	m_j^2	m_j^3
Peer 1	0.00	0.90	0.00
Peer 3	0.67	0.08	0.60
Peer 4	0.23	0.49	0.80
Peer 6	0.89	0.03	0.80
Peer 10	0.00	1.00	0.00

As equações 4.9 e 4.10 demonstram como é feito o cálculo do valor de $k_j^{'}$ para a última etapa. A multiplicação de uma matriz formada por uma linha com os pesos por outra com os valores das métricas representa o conjunto de equações que são formadas pela aplicação da equação 4.5 em cada *peer* candidato.

$$\begin{bmatrix} p_1 & p_2 & p_3 \end{bmatrix} * \begin{bmatrix} [m_1^1] & [m_3^1] & [m_4^1] & [m_6^1] & [m_{10}^1] \\ [m_1^2] & [m_3^2] & [m_4^2] & [m_6^1] & [m_{10}^1] \\ [m_1^3] & [m_3^3] & [m_4^3] & [m_6^1] & [m_{10}^1] \end{bmatrix} = \begin{bmatrix} k_1^{'} & k_3^{'} & k_4^{'} & k_6^{'} & k_{10}^{'} \end{bmatrix} \tag{4.9}$$

$$\begin{bmatrix} 0.50 & 0.25 & 0.25 \end{bmatrix} * \begin{bmatrix} 0.00 & 0.67 & 0.23 & 0.89 & 0.00 \\ 0.90 & 0.08 & 0.49 & 0.03 & 1.00 \\ 1.00 & 0.40 & 0.20 & 0.20 & 1.00 \end{bmatrix} = \begin{bmatrix} 0.475 & 0.455 & 0.287 & 0.502 & 0.500 \end{bmatrix} \tag{4.10}$$

Tabela 4.6: Valores de $k_j^{'}$ calculados e ordenados na terceira etapa para o vetor de prioridades [0.5,0.25,0.25].

$k_j^{'}$	Valor
$k_1^{'}$	0.475
$k_3^{'}$	0.455
$k_4^{'}$	0.287
$k_6^{'}$	0.502
$k_{10}^{'}$	0.500

A aplicação final das fórmulas leva à tabela 4.6. O número de candidatos que são aprovados para a seleção aleatória é definido por $M_{user} * n$, com $M_{user} = 0.4$ e $n = 5$, chegamos a 2 *peers* aprovados. Eles são os *peers* com menor valor para o índice $k_j^{'}$ na tabela, ou seja, os *peers* 3 e 4. Uma escolha aleatória seria feita entre esses dois *peers* para obter o responsável final pela invocação.

Uma discussão é necessária para justificar a utilização da seleção aleatória na última etapa. Permitir que o *peer* 3, com custo maior, seja escolhido no lugar do *peer* 4 é um reflexo da proporção entre os pesos informados. No vetor $[0.5, 0.25, 0.25]$ (equação 4.8), a prioridade do custo é duas vezes maior do que as prioridades do tempo de execução e da disponibilidade. Entretanto, o custo de execução de 3 é cerca de duas vezes o custo de 4 e o tempo de execução de 4 é cinco vezes maior que o de 3 (de acordo com a tabela 4.2). Logo, a diferença entre os pesos (0.5 e 0.25) não é suficiente para justificar a escolha direta do *peer* 4 já que a proporção entre os tempos de execução é maior do que a proporção entre os custos.

Considere que, no lugar do vetor $[0.5, 0.25, 0.25]$, fosse informado o vetor $[0.8, 0.1, 0.1]$. A prioridade do custo agora é oito vezes maior do que a do tempo de execução. O cálculo de $k_j^{'}$ para a última etapa com o novo vetor está na equação 4.11. O novos valores de $k_j^{'}$ estão na tabela 4.7.

$$\begin{bmatrix} 0.80 & 0.10 & 0.10 \end{bmatrix} * \begin{bmatrix} 0.00 & 0.67 & 0.23 & 0.89 & 0.00 \\ 0.90 & 0.08 & 0.49 & 0.03 & 1.00 \\ 1.00 & 0.40 & 0.20 & 0.20 & 1.00 \end{bmatrix} = \begin{bmatrix} 0.190 & 0.584 & 0.253 & 0.735 & 0.200 \end{bmatrix} \tag{4.11}$$

Tabela 4.7: Valores de $k_j^{'}$ calculados e ordenados na terceira etapa para o vetor de prioridades [0.8,0.1,0.1].

$k_j^{'}$	Valor
$k_1^{'}$	0.190
$k_3^{'}$	0.584
$k_4^{'}$	0.253
$k_6^{'}$	0.735
$k_{10}^{'}$	0.200

Aumentando a prioridade do custo, os candidatos restantes são os *peers* 1 e 10, no lugar dos *peers* 3 e 4. Tanto o *peer* 1 quanto o 10 possuem os menores cursos, porém seus valores para o tempo de execução estão entre os piores. Desta forma, o usuário pode influenciar a execução ao definir o vetor de pesos, sendo que quanto maior a proporção entre os pesos, maior será a importância dada para a métrica de maior prioridade.

A estratégia de execução proposta não garante a escolha da melhor solução possível, já que esse problema pertence a classe NP-Completo (YU; BUYYA, 2006b). O que é feito é a redução do número de soluções candidatas possíveis de acordo com os requisitos do usuário e da grade. Nesse aspecto, a proposta guarda semelhanças com heurísticas de otimização. Por ser tratar de uma escolha aleatória, como o espaço amostral encontra-se mais adequado às necessidades do problema, a tendência é que o processo direcione a grade a atender aos requisitos dos usuários e administradores com mais regularidade.

Uma análise de como a proposta se comporta perante as soluções encontradas na literatura para o mesmo problema é apresentada no capítulo 5.

4.4.3 Execução de um *Workflow* DAG

As seções anteriores trataram da interação entre *peers* adjacentes no grafo DAG que representa um *workflow*. O processo de execução de um grafo completo é considerado nesta seção. No grafo, os nós representam invocações que são atribuídas a *peers*. Na figura 4.11, temos os quatro blocos básicos de construção de um grafo DAG para representação de um *workflow*. Na situação (1), temos os exemplos que foram apresentados até agora, uma sequência

serial de nós. Na situação (2) temos uma situação na qual um *peer* precisa negociar uma série de sucessores (três no exemplo, mas poderiam ser mais). Neste caso, o *peer* aplica o processo da seção 4.4.2 para cada sucessor e transfere seus dados de saída para cada um escolhido.

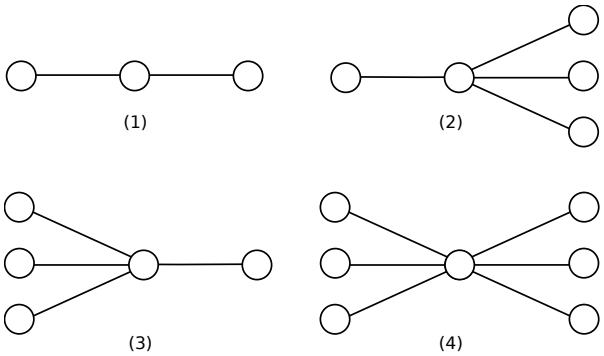

Figura 4.11: Possíveis configurações de nós em um DAG.

Um problema surge nos casos (3) e (4). Nessas situações, é preciso que os *peers* antecessores negociem em conjunto qual será o único sucessor, em seguida direcionando seus dados de saída para o escolhido. Existem várias maneiras de se atingir um consenso entre processos distribuídos (TANENBAUM; STEEN, 2007). As técnicas de maior sucesso envolvem a eleição de um processo coordenador. A solução adotada neste trabalho define que um processo é eleito coordenador, sendo ele responsável por negociar o próximo *peer*, notificar os *peers* interessados sobre sua escolha, coletar os dados dos mesmos e acionar a execução com as entradas consolidadas no *peer* escolhido como sucessor.

Para a eleição do coordenador ocorrer, no momento da seleção do sucessor, o *peer* precisa inspecionar o *workflow* para descobrir se existem nós vizinhos de execução em relação ao nó sucessor. Um nó é vizinho de execução de outro nó quando ambos possuem sucessores em comum. Na figura 4.11, os casos (3) e (4) possuem nós que são vizinhos de execução.

Durante a negociação, se existem nós vizinhos de execução, é realizada uma busca na rede para descobrir quais são os *peers* responsáveis pelas etapas vizinhas representadas pelos nós. Após a descoberta desses *peers*, eles trocam em si uma mensagem contendo o tempo de funcionamento de cada um. Aquele que estiver a mais tempo em execução é escolhido como líder e organiza a seleção dos sucessores comum. O objetivo é priorizar a escolha feita por *peers* que fazem parte da grade a mais tempo.

Para dar suporte ao processo de eleição entre vizinhos de execução, a estrutura de dados que representa o *workflow*, além das características discutidas na seção 4.3, precisa apresentar os detalhes da figura 4.12. Como em um grafo DAG qualquer, os nós e as arestas direcionadas indicam o fluxo de execução. O diferencial são os registros armazenados em cada nó. Dois

identificadores são utilizados para localizar unicamente a invocação de cada nó. As referências de entrada e saída representam os dados de entrada para a invocação do serviço e a saída produzida pelo mesmo. O campo serviço indica a serviço que será invocada por cada nó. O estado de invocação determina se o nó já realizou ou não a invocação designada. E o *peer* associado indica o endereço do *peer* que realizou a invocação, caso ela já tenha ocorrido. Essa representação está de acordo com a representação que o usuário submete na figura 4.6, sendo que nela também são armazenados os requisitos de QoS.

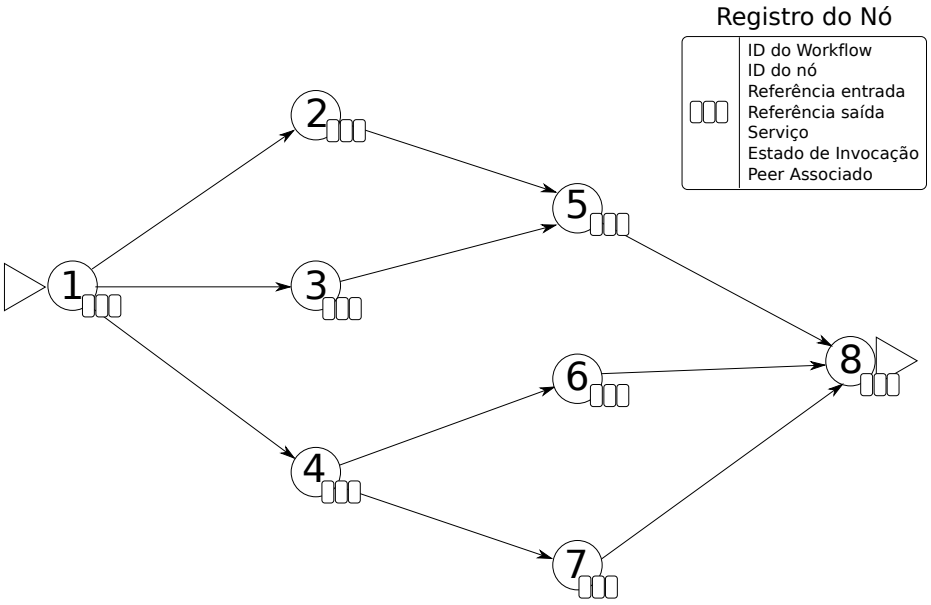

Figura 4.12: Estrutura de dados para o *workflow*.

A estrutura de dados de um *workflow* é atualizada e encaminhada por todos os *peers* que fazem parte da execução da instância. Imediatamente após a submissão, o ambiente armazena na estrutura os requisitos de QoS, adiciona os dados de entrada iniciais no registro do primeiro nó e configura seu estado de invocação para ativo, sendo que o mesmo estado para todos os outros nós tem valor padrão inicial de inativo. Quando o primeiro *peer* é escolhido para executar a primeira invocação, ele procura no *workflow* qual nó está com estado de invocação ativo e executa a invocação associada a este nó corrente. Logo em seguida, ele desabilita o estado invocação do nó corrente, adiciona os dados de saída na estrutura do *workflow* e aciona o estado de invocação para cada um dos seus sucessores. Com o sucesso na negociação de escolha de seus sucessores, ele transfere a estrutura para cada um deles. O processo se repete em cada um dos sucessores até a conclusão do *workflow*.

Os procedimentos *runInvocation(Workflow workflow)* e *searchForNeighbors(workflow, currentNode, successor, neighbors)* representam os passos necessários para a execução completa do *workflow* na grade *peer-to-peer*. O primeiro representa o fluxo de execução principal

que um *peer* executa ao ser escalonado para uma etapa do *workflow*. O *peer* recebe como parâmetro a estrutura de dados contendo o *workflow* em execução, e dela descobre através do estado de invocação qual nó contém as informações da etapa a ser realizada (linha 1). Em seguida, a invocação em si é realizada com os dados do nó corrente (linha 2). O laço seguinte (linhas 3 a 22) trata de definir qual serão os *peers* a executar cada um dos nós sucessores do nó corrente no *workflow*.

A primeira ação para decidir sobre um sucessor é descobrir se há ou não vizinhos de execução relativos a ele no *workflow* (linha 5). Se existirem vizinhos, então é realizada uma busca a partir do *peer* responsável pelo nó anterior no *workflow* (linhas 6 a 10). A invocação do procedimento *searchForNeighbors(workflow, currentNode, successor, neighbors)* não é igual a dos outros métodos, pois faz uso do sistema de notificações mencionado na seção 4.2. O método não retorna apenas ao final da busca, mas sim cadastra o vetor de vizinhos passado como parâmetro para ser atualizado com as notificações, retornando logo em seguida. Como demonstrado na descrição do procedimento, o nó antecessor verifica se é vizinho (linha 1, no segundo procedimento), caso não seja, encaminha a busca para seus descendentes e antecessor (linha 4 a 6, também no segundo procedimento). O procedimento principal então fica em espera até que todos os outros vizinhos tenham terminado sua execução, reenviando a busca em intervalos periódicos (linhas 8 a 10). No momento em que todos os vizinhos estão prontos, um coordenador é escolhido através da negociação. Se o vencedor for o *peer* local, ele seleciona o sucessor (linha 13), espera pelos dados dos vizinhos (linha 15), e encaminha todos os resultados para o mesmo (linha 16). Caso contrário, se o vencedor for outro, o *peer* local encaminha os dados da sua invocação para o vencedor (linha 18).

Caso não existam vizinhos de execução, o *peer* negocia por si só, atualiza o *workflow* e transmite os dados para o próximo *peer* (linhas 19 a 22). A figura 4.13 apresenta uma situação de negociação de vizinhos que é resolvida de acordo com os procedimentos discutidos. Antes da transição 1, a etapa 7 está pronta para executar, porém detecta a existência de vizinhos e realiza a transição para a escolha do sucessor comum. O processo de busca é realizado e quando todos os vizinhos respondem à consulta, um processo de negociação é iniciado. Após um consenso ser atingido, a transição 2 leva para uma situação na qual o *peer* escolhido para executar a etapa 8 pode prosseguir sem empecilhos. Para definir quem é o *peer* vencedor em uma negociação por vizinhos, as mensagens de negociação trocadas possuem o tempo de execução de cada *peer*. É declarado vencedor aquele que estiver há mais tempo em execução.

O método *selectSucessor(workflow, sucessor)* é usado tanto nas situações com ou sem vizinhos. É o responsável pela negociação para a escolha do sucessor, como descrito na seção

Procedimento `runInvocation(`*Workflow workflow*`)`

```
1  currentNode = workflow.getCurrentNode();
2  currentNode.output = LocalPeer.invoke(currentNode.service, currentNode.input);
3  foreach sucessor in workflow.getSuccessors(currentNode); do
4      numberOfNeighbors = isThereNeighbors(workflow, successor);
5      if numberOfNeighbors != 0 then
6          neighbors[] = new Node()[];
7          ParentPeer.searchForNeighbors(workflow, currentNode, successor, neighbors);
8          while neighbors.size() < numberOfNeighbors do
9              sleep(60);
10             ParentPeer.searchForNeighbors(workflow, currentNode, successor,
               neighbors);
11         winner = negotiateSuccessor(workflow, sucessor, neighbors);
12         if winner == LocalPeer then
13             sucessorPeer = selectSuccessor(workflow, successor);
14             updateWorkflow(workflow, sucessor, output);
15             waitForNeighborsData(neighbors);
16             successorPeer.runInvocation(workflow);
17         else
18             winner.updateFromNeighbor(workflow, currentNode, successor, output);
19     else
20         successorPeer = selectSuccessor(workflow, successor);
21         updateWorkflow(workflow, successor, output);
22         successorPeer.runInvocation(workflow);
```

Procedimento `searchForNeighbors(`*workflow, currentNode, successor, neighbors*`)`

```
1  if LocalPeer.isNeighbor(currentNode, successor) then
2      neighbors.add(LocalPeer);
3  else
4      foreach successor in LocalPeer.getSuccessors(workflow) do
5          sucessor.searchForNeighbors(workflow, currentNode, successor, neighbors);
6      ParentPeer.searchForNeighbors(workflow, currentNode, successor, neighbors);
```

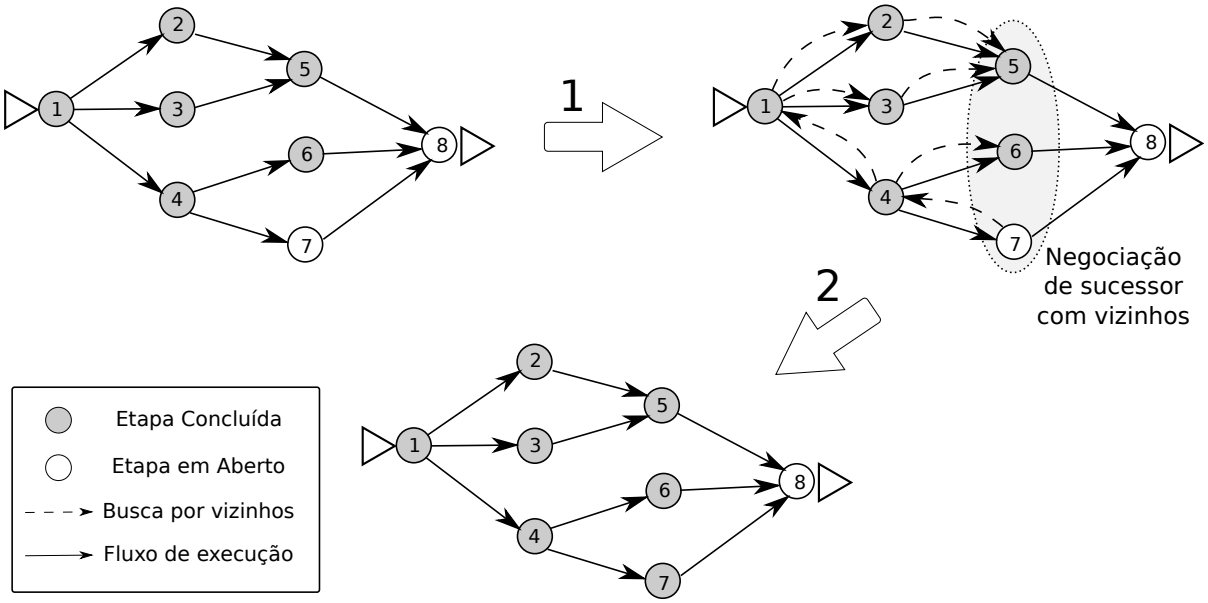

Figura 4.13: Negociação de Vizinhos.

4.4.2. É importante diferenciar os dois tipos de negociações existentes. A negociação entre vizinhos visa definir qual *peer* será responsável pela negociação do sucessor, sendo que esta última ocorre tanto na ausência quanto na presença de vizinhos.

4.5 Conclusão

Neste capítulo foi apresentada uma solução que permite aos administradores de grades computacionais estabelecerem um ambiente para execução *peer-to-peer* de *workflows* com alto nível de descentralização, fornecendo ao usuário a abstração da orquestração de serviços em *workflows* e possibilitando ao mesmo influenciar no escalonamento ao informar prioridades entre as métricas de QoS. A técnica desenvolvida leva em consideração os avanços da área descritos no capítulo 3 e aprimora as soluções existentes para a execução de *workflows* em grades comunitárias considerando requisitos de QoS. No capítulo 5, um protótipo da implementação do ambiente utilizando o *framework* WSRF é avaliado em relação às estratégias existentes com propósitos semelhantes.

5 Análise Comparativa

Neste capítulo é apresentada uma avaliação da execução de *workflows* de acordo com as técnicas detalhadas no capítulo 4. O objetivo é estabelecer um protótipo de uma grade, submeter uma carga de trabalho composta de instâncias de *workflows* e analisar o comportamento da proposta deste trabalho em relação à soluções existentes como o SwinDew e os algoritmos genéticos.

Na seção 5.1 está a descrição da implementação do ambiente. A seção 5.2 apresenta as aplicações que foram utilizadas para desenvolver os *workflows* utilizados na análise. As seções 5.3 e 5.4 explicam como as aplicações são organizadas em uma carga de trabalho e quais são os recursos computacionais que foram utilizados para a formação da grade protótipo, respectivamente. Os resultados da análise são apresentados na seção 5.5.

5.1 Arquitetura do Ambiente

O ambiente de execução faz uso de uma arquitetura orientada à serviços para representar as funcionalidades necessárias para execução de *workflows*. Os três serviços que compõem o sistema estão na figura 5.1, com suas principais operações descritas na tabela 5.1. Os serviços são implementados na linguagem de programação Java, de acordo com os paradigmas do WSRF, apresentados no capítulo 2. Neste contexto, a cada serviço está associado um recurso que representa um repositório de dados necessários para o serviço desempenhar sua funcionalidade. Tais recursos são acessíveis através da operação *GetResourceProperty* (SOTOMAYOR; CHILDERS, 2005), presentes em todos os serviços da arquitetura.

O serviço *Peer Management Service* oferece operações relacionadas a manutenção da rede *peer-to-peer*. Essas operações atualizam a tabela de roteamento da rede *peer-to-peer* sobreposta, de acordo com a entrada e saída de *peers*. Há também operações de busca que recebem consultas baseadas em palavras chaves ou nomes de serviço. O recurso anexado ao serviço trata-se de um mapeamento entre *peers* e serviços, semelhante ao descrito na tabela 4.1. En-

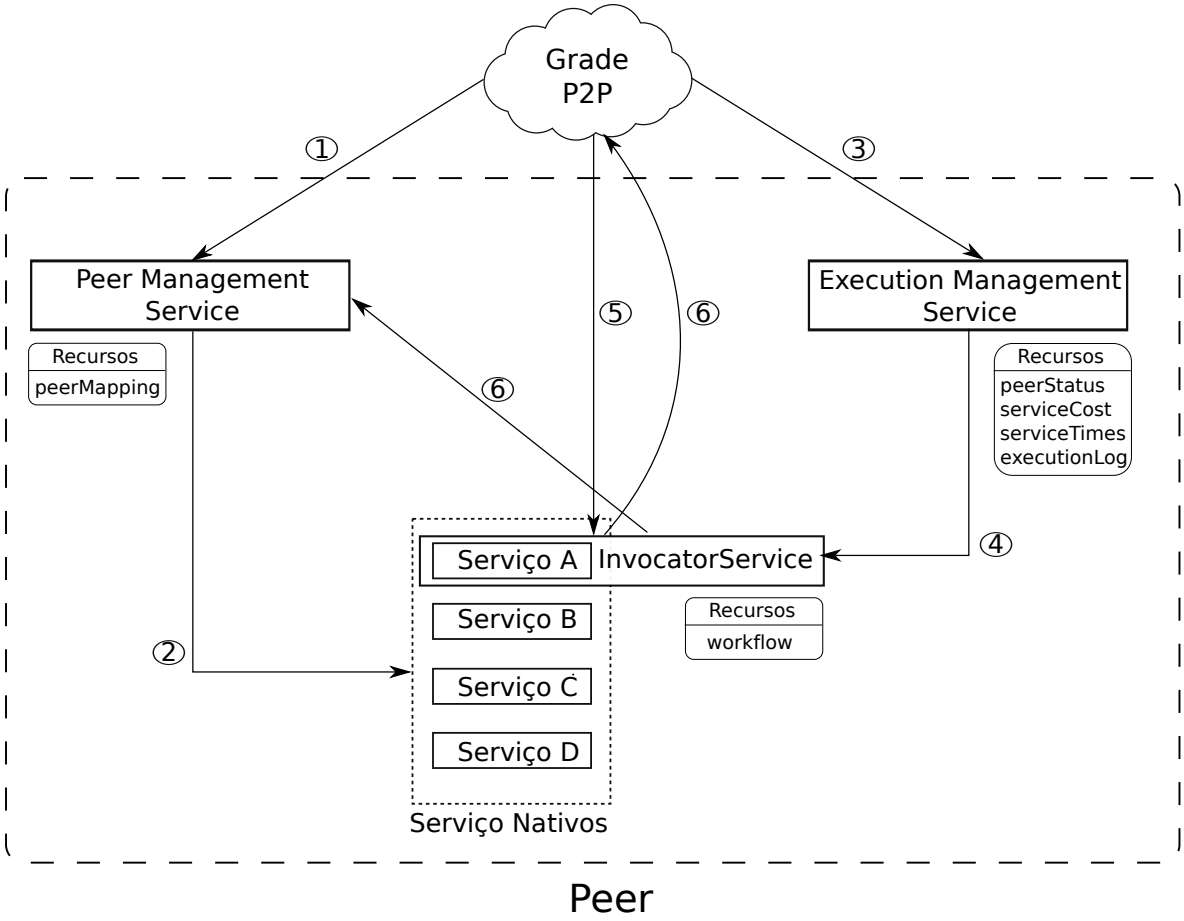

Figura 5.1: Arquitetura do ambiente.

tretanto, nesse caso as chaves do mapeamento são os identificadores dos serviços e os valores são os endereços dos serviços *Peer Management Service* associados a cada *peer*. Na figura 5.1, as operações de busca originadas dos clientes ou de outros *peers* (1) são tranformadas pelo serviço *Peer Management Service* em consultas no mapeamento de serviços locais (2). Existindo serviços locais que atendam a consulta, uma mensagem de evento é encaminhada ao remetente. Em seguida, a busca é enviada para todos os outros *peers* presentes no mapeamento. O funcionamento dessas operações não difere das operações existentes na literatura (AKON et al., 2010) para manutenção de redes *peer-to-peer* não estruturadas. A implementação utilizada na proposta é similar a adotada pelo SwinDew, mais detalhes podem ser encontrados em em (YAN; YANG; SHEN, 2006).

O serviço *Execution Management Service* trata da execução de invocações aos serviços hospedados no *peer*. Quando um outro *peer* ou um cliente precisa de informações sobre o estado do *peer* que respondeu a uma busca, os recursos anexados ao serviço *Execution Management Service* do *peer* são consultados (3). São esses recursos que armazenam os dados que permitem calcular os valores das métricas de QoS do *peer*. Na figura 5.1, o recurso *peerStatus* contém

informações atualizadas sobre a carga do *peer*, enquanto *serviceCost* e *serviceTimes* armazenam o preço de cada serviço e o tempo de execução médio de cada serviço, respectivamente.

Caso o *peer* seja o escolhido para tratar uma invocação, seu serviço *Execution Management Service* cria um novo *WS-Resource* que implementa o serviço *InvocatorService*, retornando a referência ao novo serviço para o remetente (4). Cada novo *WS-Resource* tem suas informações de referência armazenadas no recurso *executionLog*. Desta forma, o *Execution Management Service* pode processar as requisições feitas a sua operação *searchForNeighbors*, detalhada no procedimento 2 do capítulo 4.

Tabela 5.1: Serviços do ambiente.

Serviço	Operações
Peer Management Service	addPeer(endpoint) getServices() addService(endpoint) searchService(query)
Execution Management Service	createInvocator(workflow,input) searchForNeighbors(workflow,sucessor)
Invocator Service	runInvocation() negotiateSucessor(sucessor) updateFromNeighbor(input)

A nova instância do *InvocatorService* contém como recurso anexado a estrutura de dados do *workflow* em execução, como descrita na seção 4.4.3. A operação *runInvocation* representa o procedimento de mesmo nome detalhado na seção 4.4.3. No momento que o detentor da referência ao serviço realizar a invocação dessa operação (5), o serviço trata de realizar a invocação do serviço local com os dados de entrada armazenados na estrutura de dados do *workflow*. O detentor da referência é o mesmo *peer* que iniciou o processo nos passos (1) e (3). Em seguida, consultando as instâncias locais e remotas do serviço *Peer Management Service* (6), as etapas para definição do próximo *peer* são realizadas, reiniciando o processo. As operações *negotiateSucessor* e *updateFromNeighbor* são necessárias para realizar a negociação entre vizinhos e a consolidação de dados de oriundos de vários predecessores, respectivamente.

Para representar a estrutura de dados dos *workflows* descrita na seção 4.4.3, foi utilizado o *framework* JUNG (MADADHAIN; FISHER; NELSON, 2007). A razão da escolha desse *framework* é a sua versatilidade na representação de vários tipos de grafos, além de permitir a extensão das classes que representam os vértices e as arestas de grafos, facilitando a criação da estrutura necessária. Os *workflows* podem ser construídos através da programação de uma classe Java ou definidos em um arquivo XML baseado na linguagem GraphML (BRANDES et al., 2005). O *framework* permite a leitura de arquivos XML nessa linguagem, oferecendo

flexibilidade na definição de *workflows*, já que as ferramentas existentes para edição de XML podem ser utilizadas para definição de *workflows* em GraphML.

5.2 Aplicações

As aplicações utilizadas para avaliar o comportamento da solução proposta nesta dissertação adotam o modelo de *workflow fork-join*. Tal modelo é adotado por aplicações de áreas diversas como química quântica (SCHWARZ; BLAHA, 2003) e astronomia (BERRIMAN et al., 2007). Uma representação esquemática desse modelo está na figura 5.2.

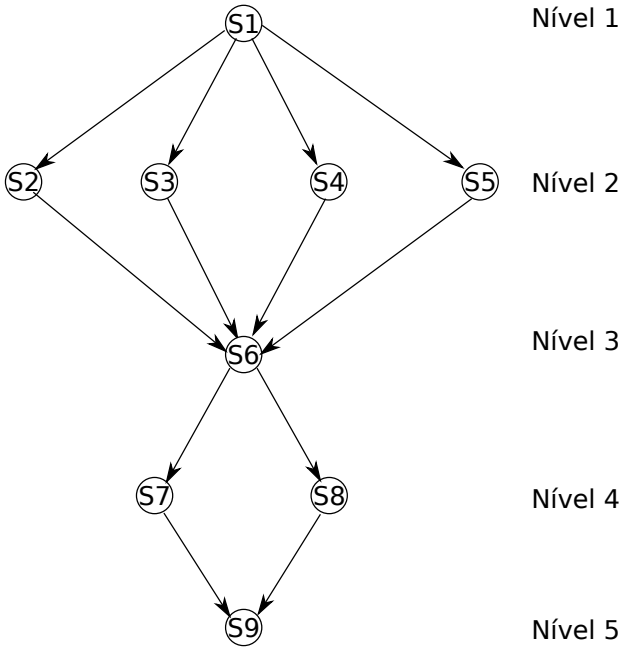

Figura 5.2: Um exemplo de *workflow fork-join*.

Nesse tipo de *workflow*, invocações de serviços são criadas (*fork*) e então convergem para uma só invocação (*join*), de tal forma que só existe uma invocação inicial e outra final. O motivo da escolha desse modelo é que ele permite representar aplicações com alto nível de paralelização, no qual as deficiências e qualidades das técnicas de escalonamento possuem maior impacto na execução de *workflows*. Por essa razão, trabalhos que analisam o desempenho de técnicas de escalonamento optam por esse modelo (MURY; SCHULZE; GOMES, 2010) (RANJAN; RAHMAN; BUYYA, 2008). Para o estudo de caso, dois tipos de aplicações desse modelo foram escolhidas, uma simulação de Monte Carlo e uma aplicação de mecânica de fluidos.

5.2.1 Simulações de Monte Carlo

O método de Monte Carlo é uma técnica para analisar fenômenos através de algoritmos computacionais que empregam a geração de sequências de números aleatórios (SHONKWILER; MENDIVIL, 2009). Essa técnica desempenha papel importante na previsão do comportamento de processos estocásticos, tais como o estudo da movimentação de partículas no espaço e a dispersão de espécimes no meio ambiente.

Como exemplo para a definição de um *workflow*, utiliza-se uma variação da técnica de *Buffon* para estimativa do valor de π. Se um círculo de raio R é inscrito dentro de um quadrado com lado $2 * R$, então a área do círculo será $\pi * R^2$ e a área do quadrado será $(2 * R)^2$. De N pontos escolhidos aleatoriamente dentro do quadrado, por volta de $N * \pi/4$ também estão dentro do círculo. Para calcular o valor de π, um procedimento escolhe pontos aleatórios dentro do quadrado durante um número de iterações passado com parâmetro. A quantidade de pontos dentro do quadrado que também pertencem ao círculo é armazenada. No final, o valor aproximado é $\pi = 4 * M/N$, no qual M são os pontos dentro do círculo e N o total gerado.

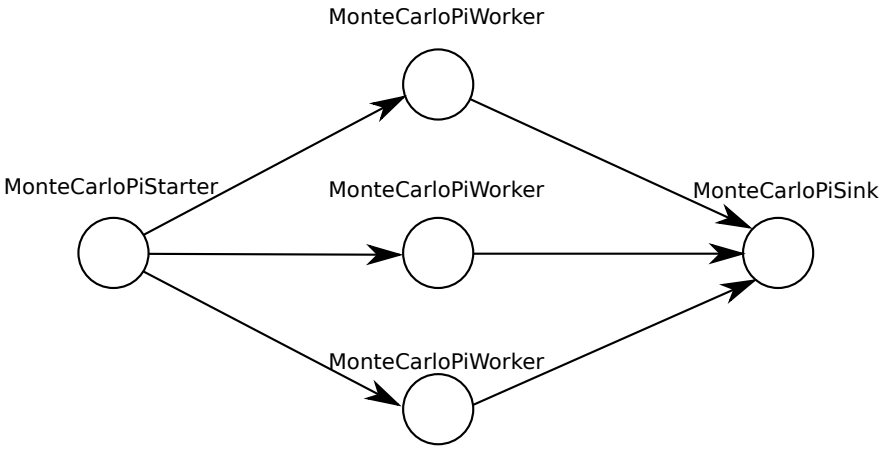

Figura 5.3: *Workflow* para simulação de Monte Carlo.

Para calcular o valor de π na grade, foram definidos os serviços representados na figura 5.3. O serviço *MonteCarloPiStarter* recebe como parâmetros o número total de iterações, as informações sobre as dimensões do círculo e quantidade de instâncias de *MonteCarloPiWorker* que devem ser invocadas. O número total de iterações é dividido pela quantidade de instâncias de *MonteCarloPiWorker* para definir quantas iterações cada instância de *MonteCarloPiWorker* irá executar. Em seguida, *MonteCarloPiStarter* realiza uma invocação para cada instância de *MonteCarloPiStarter* necessária, informando as informações do círculo e a quantidade de iterações. Após executarem a quantidade de iterações designada, as instâncias de *MonteCarloPiWorker* elegem um coordenador que encaminha todos os valores calculados para o serviço *MonteCarloPiSink*, que por sua vez realiza o cálculo final de $\pi = 4 * M/N$.

Pela própria definição do problema, percebe-se que *MonteCarloPiWorker* possui maior complexidade computacional, pois será responsável pela geração dos números aleatórios através de várias iterações. *MonteCarloPiStarter* e *MonteCarloPiSink* realizam cálculos ariméticos simples, sem nenhum iteração envolvida. Entre *MonteCarloPiStarter* e *MonteCarloPiSink*, *MonteCarloPiStarter* realiza maior esforço computacional, não pelos cálculos efetuados, mas sim por ser responsável pela criação das conexões necessárias para a criação das instâncias de *MonteCarloPiWorker*.

Em relação às transferências de dados, os serviços da simulação de Monte Carlo trocam mensagens simples. Os dados transferidos se resumem ao envio de valores numéricos relativos ao número de iterações e dos valores aleatórios obtidos. No *testbed* descrito na seção 5.4, a tecnologia de interconexão adotada permite que essas mensagens sejam transferidas em questão de milisegundos. Desta forma, considera-se que o tempo gasto na transferência de dados na execução das simulações não representa uma fração significativa do tempo total de execução de um *workflow*.

5.2.2 Mecânica de Fluidos

Os aplicativos *NASA Supercomputing Parallel Benchmarks* (WEERATUNGA et al., 1994) são um conjunto de programas projetados para ajudar na avaliação do desempenho de sistemas computacionais de alto desempenho. Esse *benchmark* é derivado de aplicações de mecânica de fluidos e consistem de cinco núcleos e três pseudo-aplicações. O objetivo da NASA ao desenvolver esse conjunto de testes era ter uma base comum que pudesse ser usada para avaliar soluções de alto desempenho fornecidas por fabricantes diversos.

As primeiras especificações do *benchmark* incluíam apenas descrições em pseudo-código dos algoritmos que deveriam implementados por aqueles interessados em avaliar os sistemas. Nas versões posteriores, foram incluídos códigos fontes funcionais em tecnologias padronizadas como MPI e OpenMP. A última versão, NPB 3.0, inclui também uma versão na linguagem Java. Uma versão para grades computacionais baseada na submissão de tarefas também está disponível, com foco na versão 2.0 do *Globus Toolkit*, sem suporte a definição de serviços.

Para a definição do *workflow* para a aplicação utilizada neste estudo de caso, o primeiro passo foi escolher quais delas se encaixam no modelo *fork-join*. A classe de problemas ED (*Embarrassingly Distributed*) representa aplicações chamadas de análises de parâmetros, que são formadas por várias execuções diferentes de um mesmo programa, porém com parâmetros de entrada diferentes. Dentro dessa classe, o principal programa é um tratador de fluxo dinâmico

chamado de SP (*Solver Program*). A figura 5.4 mostra uma representação esquemática da execução do SP.

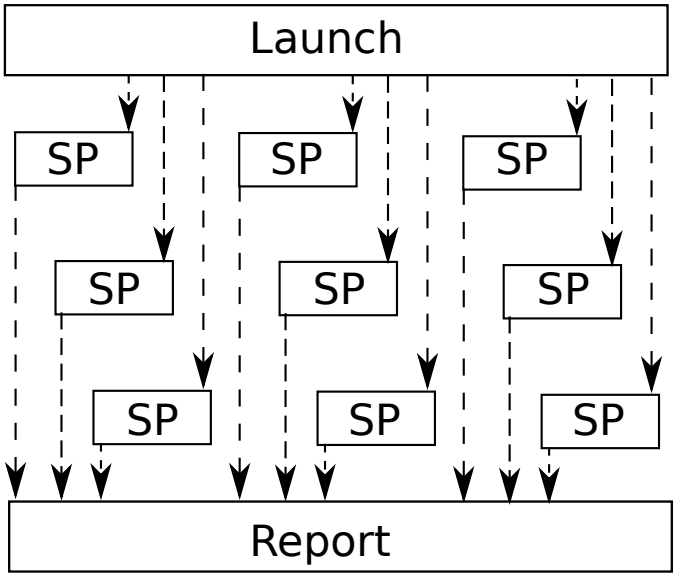

Figura 5.4: Execução do programa SP (WINJGAART; FUMKIN, 2002).

Na ausência de uma aplicação formada pela composição de serviços, a solução encontrada neste trabalho para aproveitar o *benchmark* na avaliação da proposta foi adaptar parte do código existente em serviços na grade. Por exemplo, a implementação do SP em Java consiste de uma parte serial (*Launch*, na figura 5.4) que inicializa várias versões de uma mesma *Thread*, porém com valores iniciais diferentes. Todas as *threads* são sincronizadas e retornam o valor para uma parte serial final, que consolida os resultados (*Report*). Os serviços *SPLaunch*, *SPWorker* e *SPReport* foram implementados adaptando parte do código Java já existente. *SPLaunch* e *SPReport* representam as seções seriais inicial e final, enquanto *SPWorker* encapsula os *threads* que executam a instância de SP. Desta forma, é possível modelar a aplicação em um *workflow* de composição de serviços no modelo *fork-join*.

Ao contrário da simulação de Monte Carlo, na aplicação SP as invocações intermediárias não apresentam complexidade superior em todos os casos. Execuções iniciais no *testbed*, descrito na seção 5.4, demonstraram que o serviço *SPLaunch* apresenta intervalos de execução com duração próxima aos apresentados pelo serviço *SPWorker*. A razão para tal é que os dados fornecidos pelo usuário precisam ser normalizados antes de serem submetidos aos serviços *SPWorker*, sendo que a normalização não é um processo trivial. Já o serviço *SPReport* somente anexa todos os resultados produzidos pelas instâncias de *SPWorker*, sendo seu tempo de execução menor que os dos serviços anteriores.

Os dados transferidos entre os serviços na simulação de mecânica de fluídos consis-

tem de matrizes de valores reais. De acordo com valores limites definidos no código fonte do *benchmark*, o valor máximo que uma matriz pode atingir fica na casa de 5 *megabytes*. Apesar dos intervalos de transferência serem maiores do que na simulação de Monte Carlo, ainda assim não ultrapassam a barreira dos segundos. Como pode ser observado na seção 5.5, os valores do tempo de execução para uma instância de *workflow* da aplicação estão na escala de minutos. Portanto, o tempo gasta na transferência de dados representa uma pequena fração do tempo total de execução de um *workflow*, podendo ser desconsiderados.

5.3 Carga de Trabalho

Um gerador de *workflows* foi implementado. Ele é responsável por submeter um conjunto variado de instâncias de *workflows* para a grade, em intervalos de tempo definidos. O objetivo é reproduzir a carga de trabalho de uma grade em pleno funcionamento. Essa carga de trabalho trata-se de um conjunto de 10 *workflows* que são submetidos periodicamente à grade. Os *workflows* são classificados nas categorias descritas na tabela 5.2.

Tabela 5.2: Classificação dos *workflows* da carga de trabalho.

Classe	Duração média da invocação	Variação	Quantidade
Baixa Carga	5 minutos	+- 3 minutos	5
Média Carga	15 minutos	+- 5 minutos	2
Alta Carga	30 minutos	+- 7 minutos	3

Um *workflow* de baixa carga significa uma instância em que todas as invocações que o compõem duram em média 5 minutos, com uma variação de 3 minutos para mais ou para menos. O impacto da variação é definido de maneira aleatória no momento da criação do *workflow*. O mesmo raciocínio se aplica para as demais classes. Os intervalos de tempo médio foram definidos de acordo com testes iniciais feitos com as aplicações definidas para a análise. A execução do *workflow* da simulação de Monte Carlo com uma instância intermediária de *MonteCarloPiWorker* em apenas uma máquina já fornece aproximações razoáveis de π quando o tempo de execução fica no intervalo entre 5 e 15 minutos. Portanto, os *workflows* de baixa e média carga são instâncias da simulação de Monte Carlo. No caso do exemplo de Monte Carlo, a duração da execução é controlada pela quantidade de iterações que a aplicação deve executar. Por exemplo, para um *workflow* de baixa carga (5 minutos), o número de iterações submetido fica em torno de 500.000. Já a execução da aplicação SP com as mesmas características precisou de em média 30 minutos para apresentar valores semelhantes aos relatados na documentação do *benchmark*, logo foi a escolha ideal para representar *workflows* de alta carga. A aplicação SP é configurada a partir de um arquivo texto com parâmetros para a simulação. De

maneira semelhante ao número de iterações na aplicação Monte Carlo, o arquivo de parâmetros da simulação SP foi configurado para oferecer o tempo de execução dos *workflows* de alta carga (30 minutos). A máquina utilizada para a medição de intervalos foi um *peer* com o processador Opteron(R) 248, cujos detalhes estão descritos na tabela 5.3.

O tamanho de cada *workflow* varia no intervalo $[3, N_{worker}]$, no qual N_{worker} é o número de *peers* que hospedam serviços correspondentes a instâncias intermediárias da aplicação do *workflow*. A decisão do tamanho é aleatória. Por exemplo, na simulação de Monte Carlo o serviço intermediário é o *MonteCarloPiWorker* e na aplicação SP o serviço é o *SPWorker*. Se um *workflow* de baixa carga for criado com tamanho 6, significa que ele possui uma invocação para *MonteCarloPiStarter*, outra para *MonteCarloPiSink* e 4 invocações para *MonteCarloPiWorker*.

Para definição do conjunto total de *workflows*, foi realizada uma simulação na grade protótipo, descrita na seção 5.4. O algoritmo de escalonamento utilizado decidiu qual *peer* utilizar para cada serviço de forma aleatória. No início da rodada da simulação, 10 *workflows* foram submetidos com a proporção entre as classes de acordo com a última coluna da tabela 5.2. Quando um *workflow* terminava a execução, uma nova instância da mesma classe era carregada e submetida à grade. Desta forma, em qualquer momento, a quantidade de *workflows* em execução na grade se mantinha a mesma, entretanto a carga variava, pois os *workflows* submetidos não foram idênticos. A carga do sistema variou entre uma situação na qual só existiam *workflows* de tamanho 3 na grade para outra na qual todos possuiam tamanho de N_{worker} equivalente à aplicação que representam. Essa variação foi importante, pois além de representar uma situação realista de uma grade, garante que em nenhum momento o ambiente apresentou carga nula. A simulação foi observada por um período de cinco horas, sendo que três filas foram definidas para cada classe de *workflows*. Quando um *workflow* atingia o final da execução, sua descrição era armazenada na fila da sua classe. O histórico de inserção nessas filas define o conjunto total de *workflows* que compõem a carga de trabalho. Os tamanhos das filas ficaram da seguinte forma: 248 instâncias de baixa carga, 27 de média carga e 14 de alta carga.

Para as simulações na análise posterior, a quantidade necessária para submeter os 10 primeiros *workflows* foi retirada das filas. À medida que cada *workflow* finalizava, uma nova instância era retirada da fila da mesma classe do *workflow* finalizado e submetida. Ao contrário do escalonamento original, que foi realizado de forma aleatória, na análise o escalonamento utilizado variou de acordo com a técnica em estudo.

A razão de dividir a carga de trabalho em *workflows* de cargas diferentes é reproduzir o processo evolutivo do desenvolvimento e utilização de *workflows*. O processo da descoberta científica possui natureza exploratória, sendo natural que várias tentativas sejam necessárias até

se consiga calibrar experimentos capazes de atingir os melhores resultados possíveis. Estudos recentes observam que na prática, mesmo já conhecendo a estrutura de seus experimentos, pesquisadores costumam submeter instâncias de *workflow* de curta duração quando desejam explorar novas alternativas para os parâmetros iniciais (OCI, 2007). Só depois de resultados iniciais positivos, é que instâncias de longa duração são submetidas. Por essa razão, a proporção apresentada na tabela 5.2 foi adotada.

5.4 Descrição do *Testbed*

Para realização da análise, foi estabelecida uma grade protótipo com o ambiente proposto em execução. Um conjunto de servidores foi configurado para representar os recursos disponíveis na grade. Cada servidor executa os serviços do ambiente, tornando-se assim um *peer* da grade, simulando um domínio administrativo distinto. A relação da configuração das máquinas está descrita na tabela 5.3.

Tabela 5.3: Recursos do protótipo de grade computacional.

Processador	Cache	Núcleos	Memória	Quantidade
Pentium(R) 4 3.00GHz	2 MB	1	1.5 GB	1
Opteron(R) 248 2.20GHz	1 MB	2	2.0 GB	4
Xeon(R) X5355 2.66GHz	4 MB	8	16.0 GB	3
Core(R)2 Duo E4600 2.40GHz	2 MB	2	2.0 GB	2

No total de 10 máquinas, cada uma foi configurada com a versão 4.2.1 do *Globus Toolkit* instalada. A arquitetura descrita na seção 5.1 foi implementada utilizando a plataforma *Globus*, na linguagem de programação Java, versão 1.6.0.20 da Sun. O sistema operacional utilizado foi o CentOS 5.4 com *kernel* versão 2.6.32.23. A rede de interconexão é a tecnologia *Fast Ethernet*, sendo que todas as máquinas estão em uma mesma rede local.

Uma questão crucial para a avaliação é definir como a rede *peer-to-peer* é formada. Para as duas aplicações consideradas, existem 6 serviços: *MonteCarloPiStarter*, *MonteCarloPiWorker*, *MonteCarloPiSink*, *SPLaunch*, *SPWorker* e *SPReport*. Esses serviços devem ser distribuídos pelas 10 máquinas. A primeira opção considerada foi alocar um *peer* para *MonteCarloPiStarter* e *SPLaunch*, outro para *MonteCarloPiSink* e *SPReport*, alocando os outros serviços de maneira igual nas máquinas restantes. Essa opção está descrita na figura 5.5.

Essa opção de isolar os serviços iniciais e finais não permite uma avaliação adequada da carga da grade. Apesar de não ocorrerem ao mesmo tempo e serem invocações que exigem uma quantidade menor de processamento, acumular todas as invocações iniciais das submissões

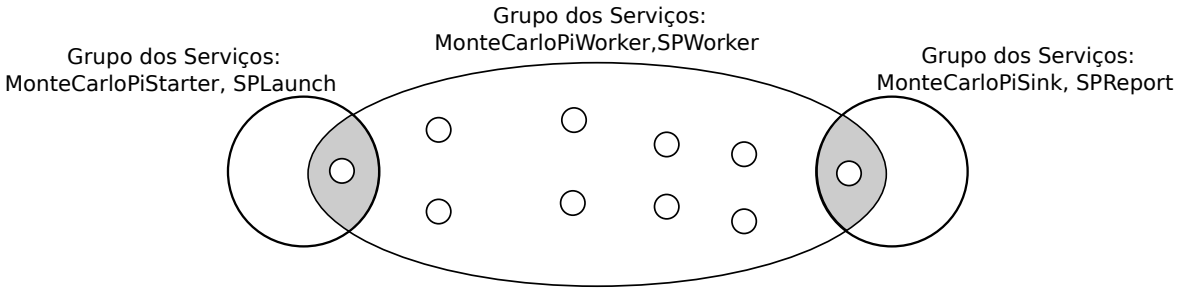

Figura 5.5: Primeira opção para alocação de serviços.

em um único *peer* causa desequilíbrio na distribuição de carga. Todo *workflow* submetido a grade invocaria o serviço inicial em um mesmo *peer*. Além disso, essa situação apresenta um gargalho como em arquiteturas centralizadas. Uma segunda opção foi projetada para evitar as limitações encontradas, sua estrutura está na figura 5.6.

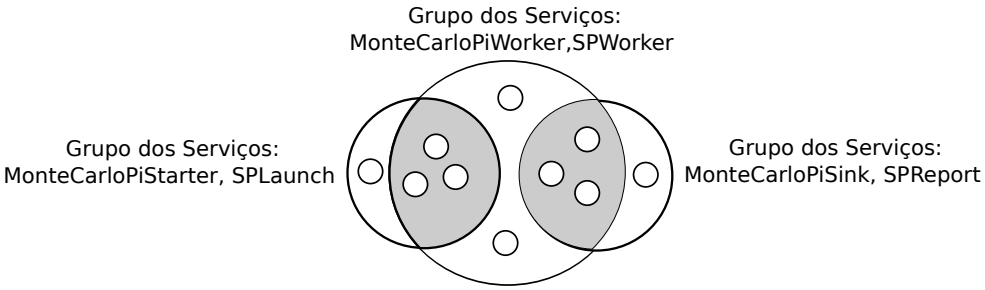

Figura 5.6: Segunda opção para alocação de serviços.

Na segunda configuração, os seis serviços estão distribuídos de maneira homogênea entre os *peers*. Não há problemas de escalabilidade e gargalos, além de corresponder a uma situação realista, pois em uma grade não é factível uma funcionalidade ser oferecida apenas por um domínio. Uma organização que oferecesse apenas uma funcionalidade teria pouco valor para seus usuários internos. Outro aspecto importante que essa configuração possui decorre da natureza das aplicações submetidas para execução. Tanto na aplicação de Monte Carlo quanto na simulação SP, os serviços que exigem mais computação são os intermediários, *MonteCarloPiWorker* e *SPWorker* respectivamente. Caso esses serviços só estivessem presentes em uma pequena quantidade de *peers* não seria possível observar a diferença no escalonamento entre as diferentes abordagens estudadas na seção 5.5. Os *peers* com os serviços *MonteCarloPiWorker* e *SPWorker* se tornariam sobrecarregados, apresentando valores uniformes para as métricas de QoS no decorrer da execução. Por essas razões, o protótipo para análise adota a organização da figura 5.6.

5.5 Resultados da Análise Comparativa

Dentre as soluções apresentadas na seção 3.4.2, apenas o SwinDew apresenta uma arquitetura *peer-to-peer* com suporte a escalonamento de *workflows* de serviços. Entretanto, o SwinDeW apenas considera o balanceamento de carga como métrica. Na seção 5.5.1, a proposta desta dissertação é comparada em relação ao SwinDeW em termos da distribuição de carga. Em seguida, requisitos de QoS são definidos e os *workflows* são analisados em relação ao cumprimento desses requisitos quando executados com o escalonamento do SwinDeW e com o da proposta. O objetivo é demonstrar que além de fornecer um melhor balanceamento de carga, a proposta também orienta a execução a cumprir os requisitos informados. O escalonamento da proposta foi implementado escalonando uma etapa por invocação finalizada.

Na seção 5.5.2, a proposta é comparada com um escalonador centralizado baseado na execução de algoritmos genéticos, como descrito na seção 3.4.1. A distribuição de carga é novamente considerada, além do comportamento das técnicas centralizada e descentralizada em relação ao impacto da definição de requisitos de QoS por parte do usuário.

Em ambas avaliações, são feitas análises tendo a distribuição de carga como métrica administrativa. O custo e o tempo de execução são as métricas do usuário. Esses requisitos são utilizados como métricas na maioria dos trabalhos considerados (PRODAN; FAHRINGER, 2005) (YU; BUYYA, 2006a). Outras métricas como segurança são também consideradas (PEIXOTO; SANTANA; SANTANA, 2009) (ANSELMI; ARDAGNA; CREMONESI, 2007), contudo, não há detalhamento sobre como calcular tais métricas em ambientes reais. Muitas vezes, as simulações utilizam valores arbitrários (ENGELBRECHT; BENKNER, 2009). Desta forma, visando permitir maior flexibilidade na escolha das soluções a serem comparadas, esse conjunto de métricas foi adotado na implementação.

Para a análise da distribuição de carga entre os *peers*, a carga de trabalho é submetida à grade com cada *workflow* sendo escalonado de acordo com a técnica em estudo. Durante a execução da carga de trabalho, informações sobre o estado de cada *peer* são recuperadas através do recurso *peerStatus* no serviço *Execution Management Service*. Dentre as informações recebidas, está a carga de cada *peer*. A carga do *peer* é definida como o tamanho médio da fila de espera de processos prontos para a execução no sistema operacional, sendo esse valor dividido pelo número de núcleos de processamento. Um estudo comparativo realizado em (FERRARI; ZHOU, 1988) afirma que o tamanho da fila é uma métrica melhor para o balanceamento de carga do que a taxa de utilização da CPU. No *testbed*, cada *peer* que representa um domínio consiste de apenas uma máquina. Entretanto, se cada *peer* fosse ponto de entrada para *clusters*,

o ambiente Ganglia (MASSIE; CHUN; CULLER, 2004) permite que tal métrica seja calculada com precisão para um grupo de máquinas.

Em relação aos requisitos de QoS do usuário, os dados necessários para análise são obtidos através da submissão paralela à carga de trabalho de um outro conjunto de *workflows*. Por exemplo, para estudar o tempo de execução dos *workflows* em uma grade com escalonamento baseado no SwinDew, 10 *workflows* de cada classe são submetidos em sequência após um intervalo fixo do início da submissão da carga de trabalho. A carga de trabalho também é submetida utilizando o escalonamento em estudo, no caso o SwinDew. O *peer* que recebe a submissão é escolhido de maneira aleatória.

Para obter o valor da métrica de tempo de execução de um serviço hospedado em um determinado *peer*, o recurso *serviceTimes* anexado ao serviço *Execution Management Service* é recuperado. Nele estão armazenadas médias do tempo de execução para cada serviço do *peer*. O próprio serviço *Execution Management Service* se responsabiliza de monitorar cada instância de *Invocator Service* e adicionar à média cada valor de nova invocação concluída.

A métrica do custo de execução para cada serviço é calculada no momento de ativação do *peer*, também pelo serviço *Execution Management Service*. O custo de execução é calculado de acordo com a equação 5.1.

$$C(i,j) = (Ncores_j * Clock_j + M_j) * P_i \tag{5.1}$$

O valor de $C(i,j)$ corresponde ao custo por minuto da invocação o serviço i no *peer* j. $Ncores_j$ representa a quantidade de núcleos de processamento do *peer*. A medida da frequência de processamento de cada núcleo é dada por $Clock_j$, em *gigahertz*. Já M_j é a quantidade de memória principal em *gigabytes*. P_i é o preço base do serviço i, definido no acordo de formação da grade. A tabela 5.4 apresenta os preços para a grade *testbed*, definidos de acordo com a complexidade de cada serviço. Todo serviço tem seu preço base associado, sendo que o valor final da invocação de um serviço em um *peer* é calculando levando em consideração esse valor inicial e o poder computacional do *peer*.

A medida que as notificações sobre invocações chegam ao processo cliente, o mesmo recupera as informações sobre o preço do serviço base e a configuração do *peer* que executou a invocação. O custo total é dado pela equação 5.2, na qual $T(i,j)$ é o intervalo de tempo em minutos que foi necessário para a execução da invocação do serviço i no *peer* j.

Tabela 5.4: Preço base do serviços.

Serviço	Preço (unidade monetária/minuto)
MonteCarloPiStarter	1
MonteCarloPiWorker	2
MonteCarloPiSink	1
SPLaunch	5
SPWorker	10
SPReport	3

$$\text{Custo Final} = \sum_{i=0}^{n} C(i,j) * T(i,j), \text{ sendo } (i,j) \text{ uma etapa do } \textit{workflow}. \tag{5.2}$$

5.5.1 SwinDew

Na comparação entre a proposta e o SwinDew, a primeira análise é apresentada na figura 5.7. Nela, está apresentado o comportamento da solução com o parâmetro $M1$ variando durante a submissão da carga de trabalho. Para cada valor do parâmetro, a carga de trabalho foi submetida usando a técnica da proposta. A mesma carga de trabalho também foi submetida usando a estratégia do SwinDew. De acordo com a seção 4.4, o parâmetro $M1$ permite controlar a relevância dos requisitos administrativos. Como no caso, o único requisito considerado é a carga, o valor de $M1$ determina o quanto o sistema deve priozirar o balanceamento de carga. Nessa primeira análise, os outros parâmetros estão configurados de forma a não interferirem no escalonamento. O objetivo é analisar apenas o impacto da estratégia dinâmica apresentada na seção 4.4.1.

Observando apenas os resultados da proposta, é possível perceber que o valor do parâmetro tem impacto direto na variância da carga entre os *peers*. Quando o parâmetro assume valores valores mais restritos como $M1 = 0.2$, a diferença entre a carga média dos *peers* não ultrapassa o valor de 1.0, além das curvas apresentarem um comportamento ordenado. Já quando o parâmetro é relaxado para $M1 = 0.8$, a variância sempre fica acima de 1.0, sendo que a curva apresenta desníveis acentuados.

Em relação ao SwinDew, a distribuição de carga da proposta é mais equilibrada quando o valor de $M1$ for aproximadamente menor ou igual a 0.5. Também na figura 5.7 está a média da carga para a solução proposta e o SwinDew, sendo que alguns valores para o parâmetro de $M1$ foram retirados visando manter a legibilidade do gráfico. Mesmo com valores extremos para $M1$, os valores da carga média se restringem à valores na faixa $[0.6, 1.1]$, sendo que o SwinDew também apresenta valores nessa faixa. A utilização de qualquer técnica não altera a carga média

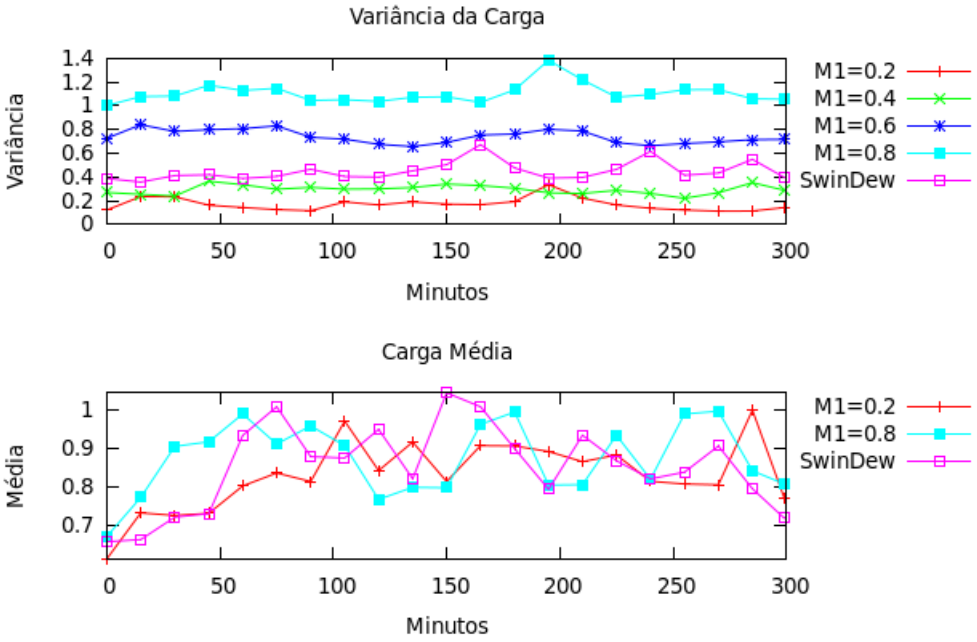

Figura 5.7: Variância e carga média dos *peers* para $M1$ variando, $M2 = 1.0$ e $p_{tempo} = p_{custo}$.

do sistema, portanto o *overhead* de utilização de qualquer uma delas é semelhante.

As figuras 5.8, 5.9 e 5.10 apresentam o comportamento da solução proposta e do SwinDew em relação à métrica do tempo de execução. As três classes de *workflows* são avaliadas, com o tamanho do *workflow* variando de 3 até 10. Novamente, se um *workflow* tem tamanho 10, significa que ele possui uma invocação inicial (*MonteCarloPiStarter* ou *SPLaunch*), sete invocações intermediárias (*MonteCarloPiStarter* ou *SPLaunch*) e mais um invocação final (*MonteCarloPiSink* ou *SPReport*). A valor de $M1$ foi fixado em 0.5 para fornecer um balanceamento de carga semelhante ao SwinDew. Os pesos para as métricas foram fixados em $p_{custo} = 0.2$ e $p_{tempo} = 0.8$, valorizando o tempo de execução em detrimento ao custo. O valor de $M2$ varia, para permitir a análise do impacto desse parâmetro na aplicação dos pesos das métricas de usuário.

Foram submetidas 10 amostras para cada tamanho, sendo que cada amostra além do mesmo tamanho possuem a mesma configuração de complexidade nas invocações. Nesse conjunto de amostrar, o intervalo de confiança foi calculado com índice $ic = 95$.

Para os *workflows* de baixa carga (figura 5.8), os valores para diferentes configurações de $M2$ são próximos, sendo que em dois pontos (6 e 8) apresentam valores iguais. O SwinDew apresenta resultados piores, e como não considera as métricas, a curva tem comportamento errático e intervalos de confiança amplos. Na figura 5.9, os *workflows* de carga média apresentam maior distância entre as curvas com parâmetros diferentes para a proposta. O SwinDew mantêm

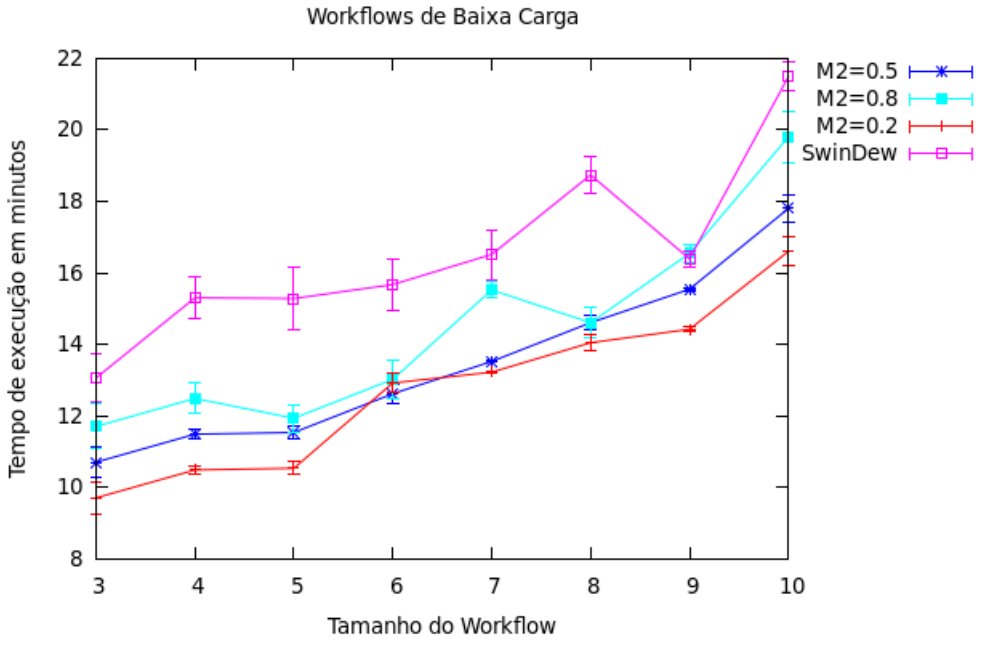

Figura 5.8: Tempo de execução para *workflows* com baixa carga e $M1 = 0.5$, $p_{custo} = 0.2$ e $p_{tempo} = 0.8$.

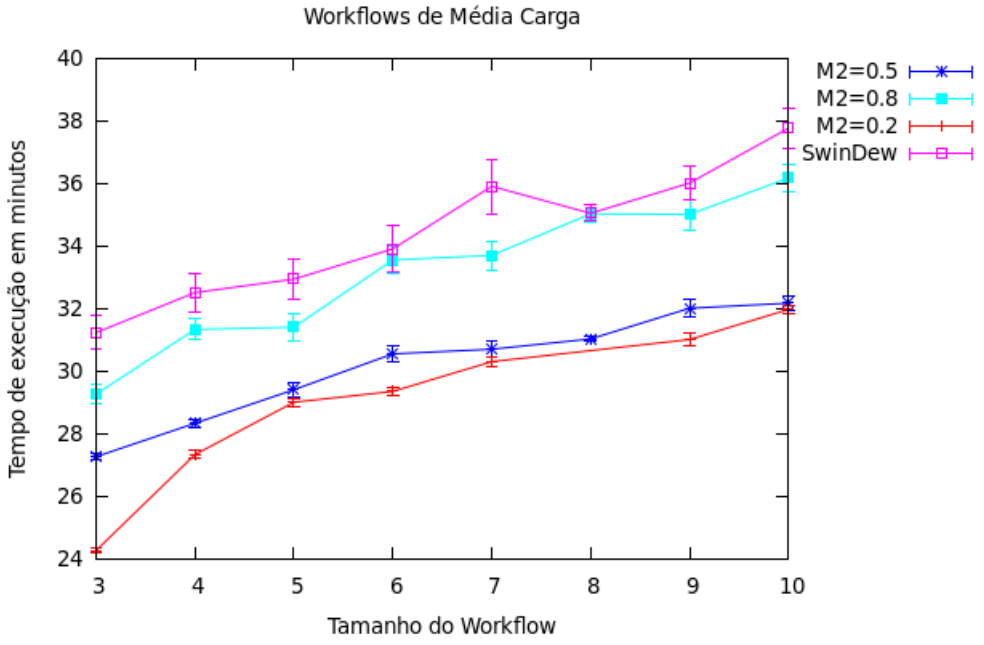

Figura 5.9: Tempo de execução para *workflows* com média carga e $M1 = 0.5$, $p_{custo} = 0.2$ e $p_{tempo} = 0.8$.

a curva irregular, entretanto há uma aproximação da curva com valor de $M2 = 0.8$.

Os *workflows* de alta carga (figura 5.10) exibem as curvas correspondentes à proposta com regularidade entre si, como valores sempre crescentes com o tamanho do *workflow*. O

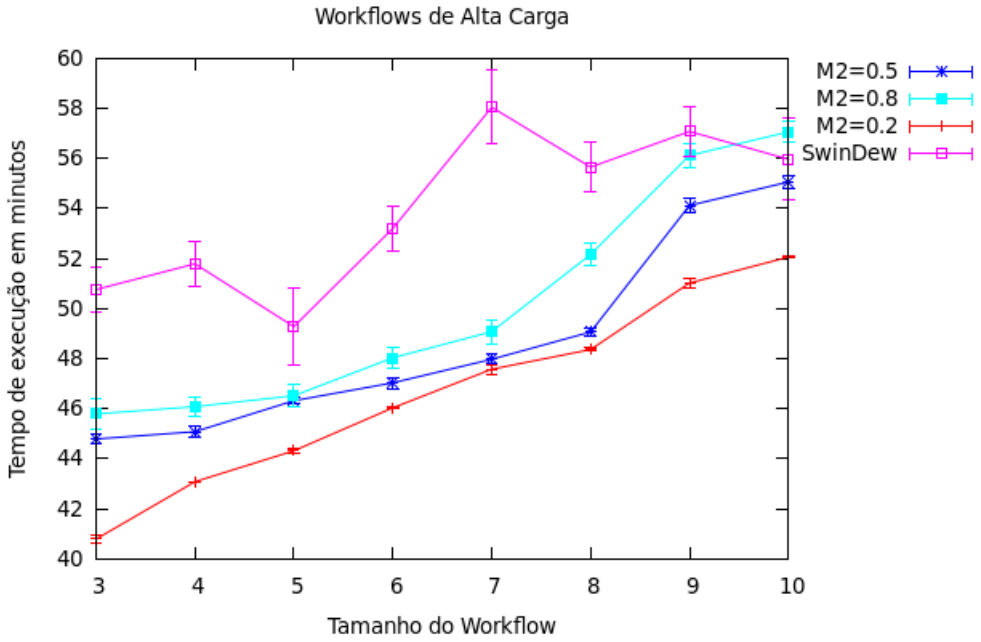

Figura 5.10: Tempo de execução para *workflows* com alta carga e $M1 = 0.5$, $p_{custo} = 0.2$ e $p_{tempo} = 0.8$.

comportamento irregular do SwinDew é acentuado.

O comportamento dos gráficos em relação ao tempo de execução pode ser explicado pela técnica de escalonamento utilizada e devido ao tipo de *workflow* aplicado. Como o SwinDew não leva em consideração as métricas de QoS, é natural que apresente um comportamento irregular em todas as avaliações. A natureza do do *workflow* explica porque as execuções de maior carga apresentam comportamento mais próximo do idealizado. A medida que a carga do *workflow* aumenta, o tempo de invocação de cada etapa também aumenta. Quanto maior o tempo de invocação das etapas, maior a eficiência adquirida pela realização do escalonamento dinâmico com informações mais recentes do estado da grade, como explicado na seção 4.4.1. Por exemplo, considere um *workflow fork-join* cuja primeira invocação tenha duração de 3 minutos. Se o escalonamento de todas as etapas for feito no momento da submissão do *workflow*, a diferença entre o momento da escolha do *peer* a realização de uma invocação intermediária será também de 3 minutos. Durante um intervalo com essa amplitude, a probabilidade do estado da grade variar a ponto de invalidar a escolha do *peer* é muito pequena. Portanto, a eficiência da técnica é acentuada quando o tempo de execução das invocações aumenta. No caso em estudo, *workflows* de alta carga possuem tempo de execução com duração média de 30 minutos, ou seja, 10% da duração total da carga de trabalho. Logo, é na sua execução que a prioridade dos requisitos é mais acentuada.

As figuras 5.11, 5.12 e 5.13 apresentam o comportamento da solução proposta e do

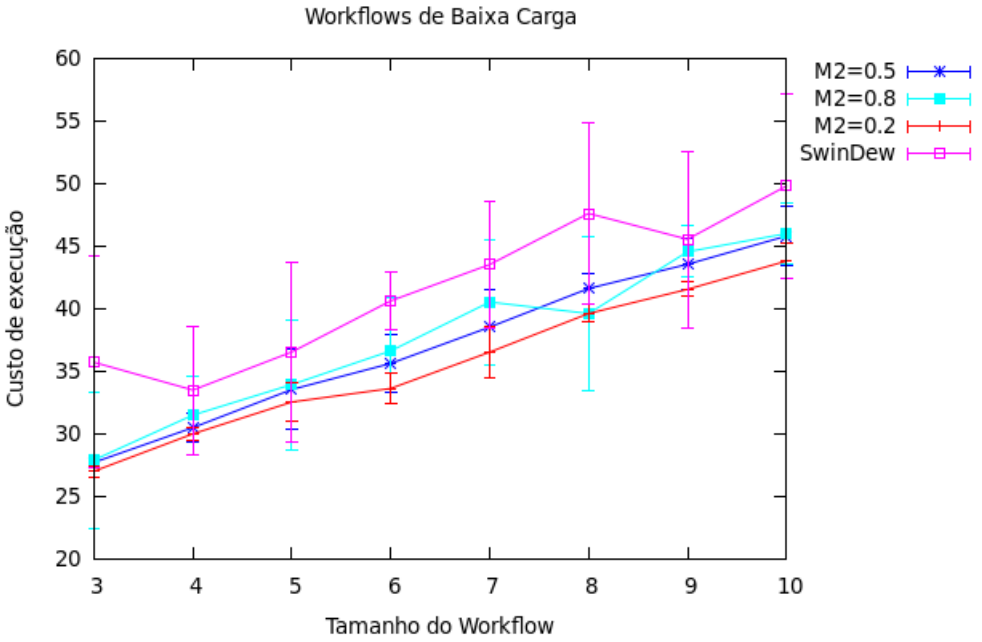

Figura 5.11: Custo de execução para *workflows* com baixa carga e $M1 = 0.5$, $p_{custo} = 0.8$ e $p_{tempo} = 0.2$.

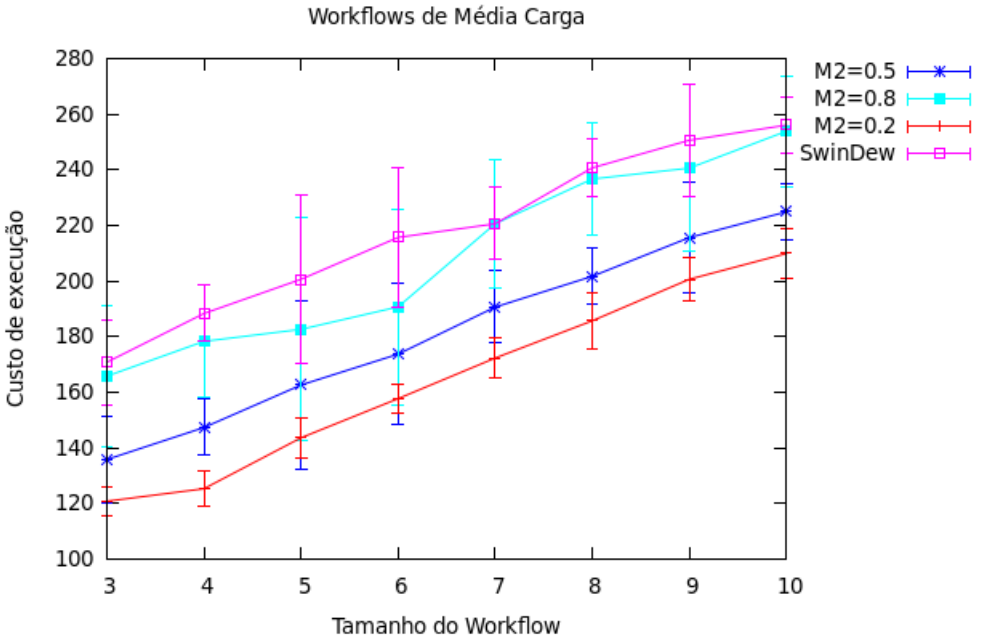

Figura 5.12: Custo de execução para *workflows* com média carga e $M1 = 0.5$, $p_{custo} = 0.8$ e $p_{tempo} = 0.2$.

SwinDew em relação à métrica do custo de execução. Assim como no caso do tempo de execução, quanto maior a carga do *workflow*, melhor a eficiência da técnica. A explicação se dá pelas mesmas razões já citadas. Entretanto, os intervalos de confiança são mais amplos no caso dessa métrica. A explicação dessa diferença se dá pela heterogeneidade do *testbed* da grade. Como o

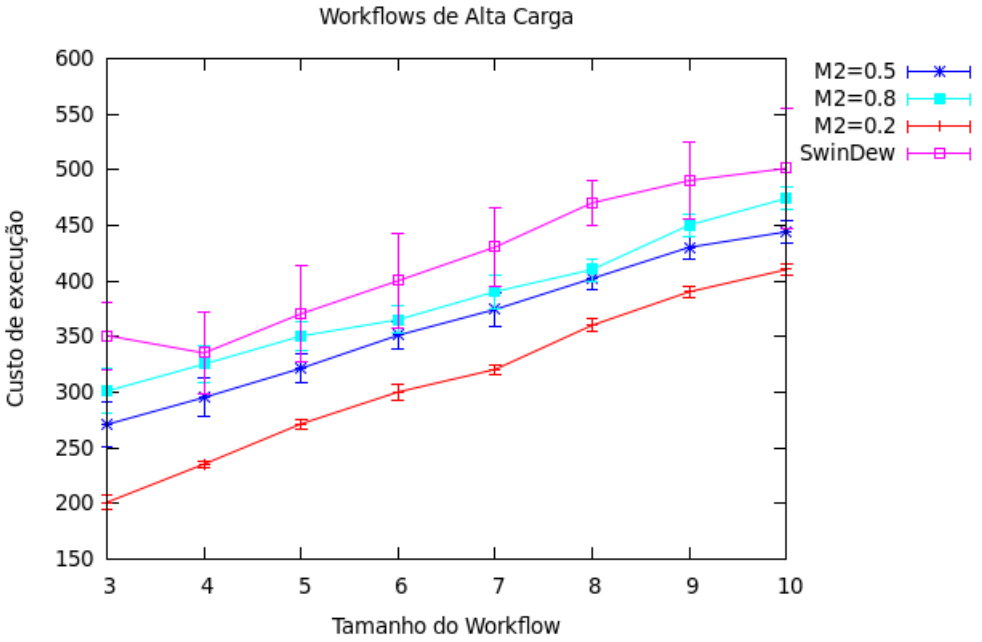

Figura 5.13: Custo de execução para *workflows* com alta carga e $M1 = 0.5$, $p_{custo} = 0.8$ e $p_{tempo} = 0.2$.

preço de um serviço de um *peer* é definido pela configuração de *hardware* do mesmo (equação 5.2), há grandes variações de custo devido à variedade de configurações das máquinas.

Considere como exemplo, uma máquina com 8 núcleos de processamento. Da maneira que a carga é definida, para que a máquina apresente carga de valor 1.0 é preciso que existam 8 processos na fila de espera para execução. Portanto, a probabilidade de passar pela primeira avaliação (aplicação do parâmetro $M1$, na seção 4.4.2) é maior do que a de um *peer* com uma quantidade menor de núcleos. Desta forma, situações nas quais na segunda fase o conjunto de *peers* candidados seja composto na sua maioria por máquinas com boa configuração são comuns. O custo dos serviços nessas máquinas é alto, mas se existir pelo menos uma máquina com equipamento inferior, ela terá maior chance de ser escolhida em um cenário no qual o peso tem maior prioridade. A amplitude do intervalo demonstra que a existência dessa máquina com capacidade interior é comum.

A amplitude dos intervalos não invalida a proposta. Apesar de representar um fator de imprevisibilidade na previsão dos resultados, a posição das curvas comprova que os pesos informados pelos usuários e os parâmetros determinados pela administração da grade possuem impacto determinante na execução do *workflow*, representando de maneira correta os desejos daqueles que os definiram.

As figuras 5.14 e 5.15 avaliam o impacto do peso informado diretamente pelo usuá-

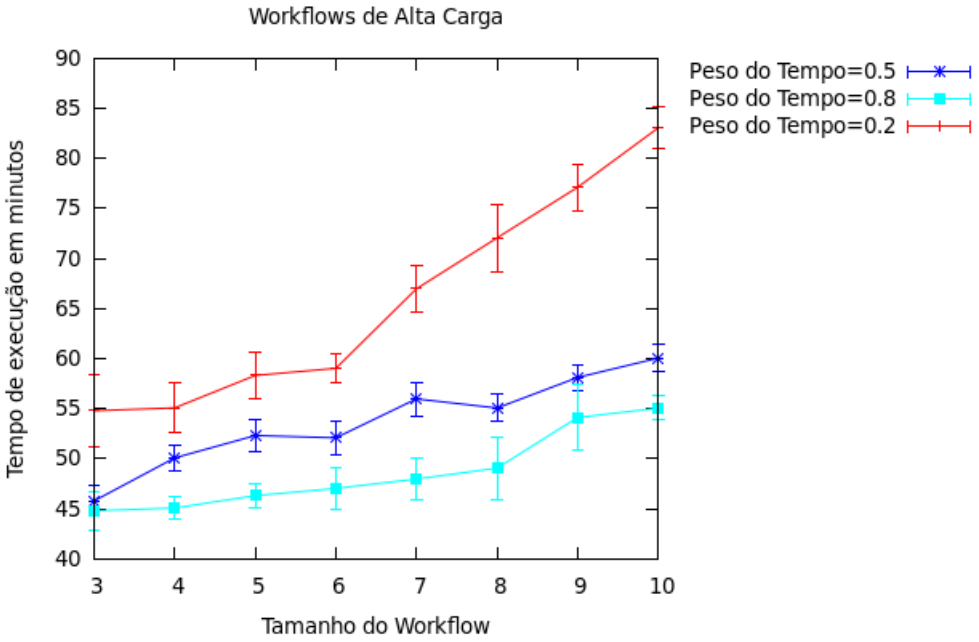

Figura 5.14: Tempo de Execução para $M1 = 0.5$ e $M2 = 0.5$

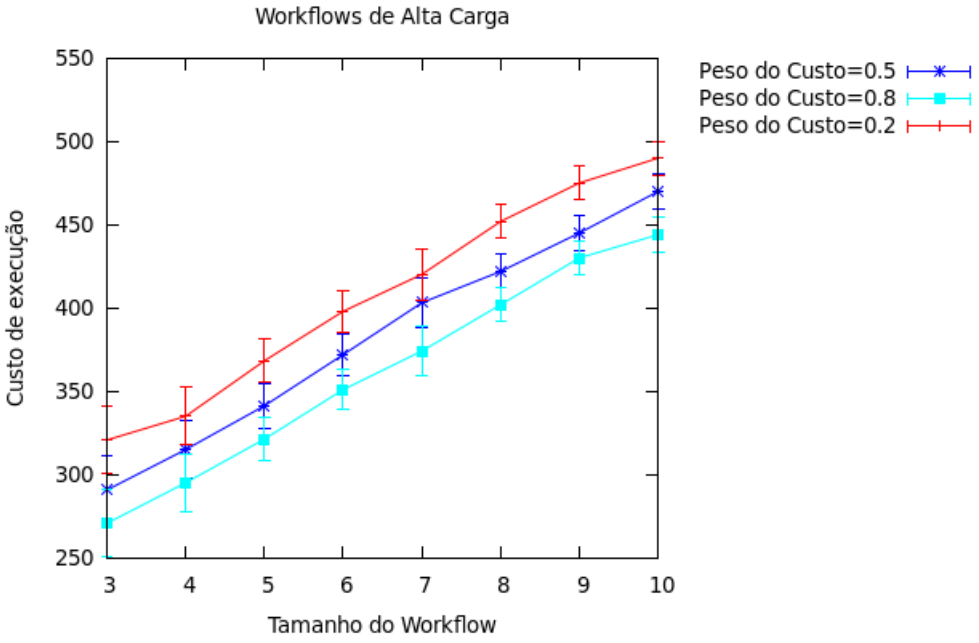

Figura 5.15: Custo de Execução para $M1 = 0.5$ e $M2 = 0.5$

rio para *workflows* de alta carga. Todos os parâmetros administrativos são fixados em valores intermediários, variando apenas a proporção entre os pesos informados pelo usuário. A técnica utilizada pelo SwinDew não é avaliada justamente por não considerar os pesos de QoS informados pelo usuário.

Na figura 5.14 estão os tempos de execução, com o valor da prioridade do tempo

variando. Neste caso, o valor do peso do custo é o complemento de 1 do peso do tempo de execução. É perceptível a influência dos pesos na orientação do escalonamento. Mesmo com intervalos de confiança relevantes, na maioria dos casos não há intersecção dos mesmos. Já na figura 5.15, a eficácia da técnica também é comprovada, apesar dos intervalos de confiança mais amplos, devido a razões já explicadas pela maneira que o preço é definido.

5.5.2 Algoritmo Genético

Para a comparação com um escalonador que utiliza um algoritmo genético, foram implementadas as estruturas de dados e funções descritas na seção 3.4.1. Os parâmetros utilizados estão na tabela 5.5 e foram definidos de acordo com estudos experimentais presentes em (YU; BUYYA, 2006a).

Tabela 5.5: Parâmetros iniciais para o escalonador com algoritmos genéticos.

Parâmetro	Valor/Tipo
Tamanho da população	10
Geração Máxima	100
Probabilidade de *crossover*	0.9
Probabilidade de Mutação	0.5
Esquema de seleção	Categoria Elitista
Indivíduos Iniciais	Geração aleatória

Além das configurações iniciais, foi estabelecida uma alteração no serviço *Execution Management Service* para permitir a reserva de recursos. A alteração foi necessária pois as outras soluções existentes que adotam algoritmos genéticos trabalham com reserva de recursos. Assim para realizar uma comparação realista com as técnicas já existentes, o mesmo foi feito na proposta dessa dissertação.

O escalonador genético foi implementado usando a mesma configuração de máquina virtual Java usada nas outras simulações. Foi utilizado o *framework* JGAP para construção de soluções baseadas em algoritmos genéticos (MEFFERT et al., 2005). Ao contrário das simulações descentralizadas, nas quais o escalonador escolhe um *peer* aleatório para submissão de um *workflow*, nos algoritmos uma máquina foi escolhida para submeter e controlar a execução. No caso, a máquina escolhida foi o Pentium 4 da tabela 5.3.

A primeira análise feita é apresentada na figura 5.16. Apesar dos algoritmos genéticos não considerarem a carga como requisito, é importante analisá-la porque a solução proposta leva em consideração a distribuição de carga. O que se pode observar é que a utilização de algoritmos genéticos para submeter a carga de trabalho leva à uma situação de desequilíbrio da

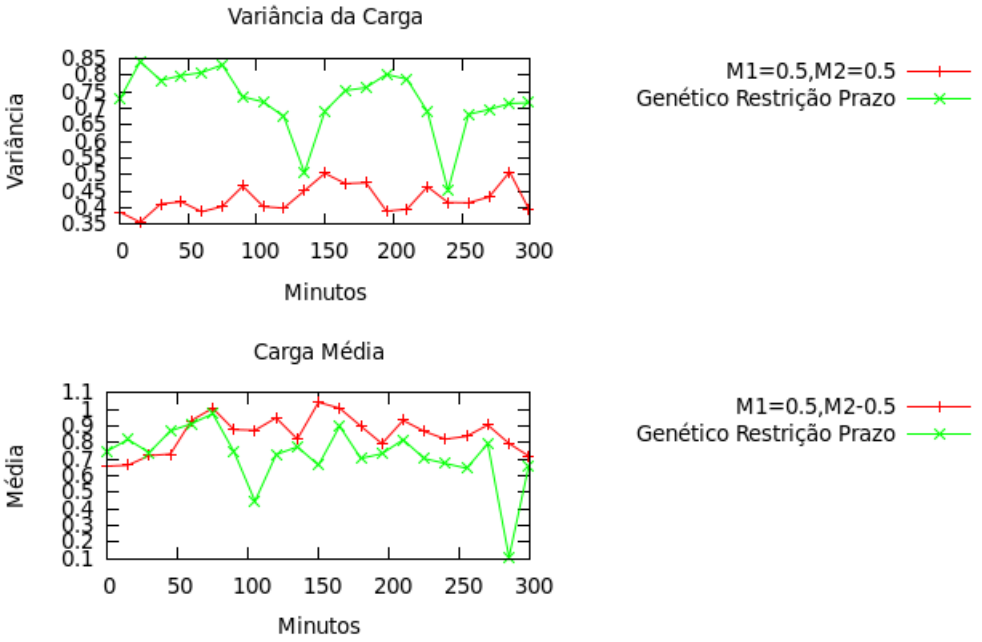

Figura 5.16: Variância e carga média dos *peers* para $M1 = 0.5$, $M2 = 0.5$ e $p_{tempo} = 2 * p_{custo}$.

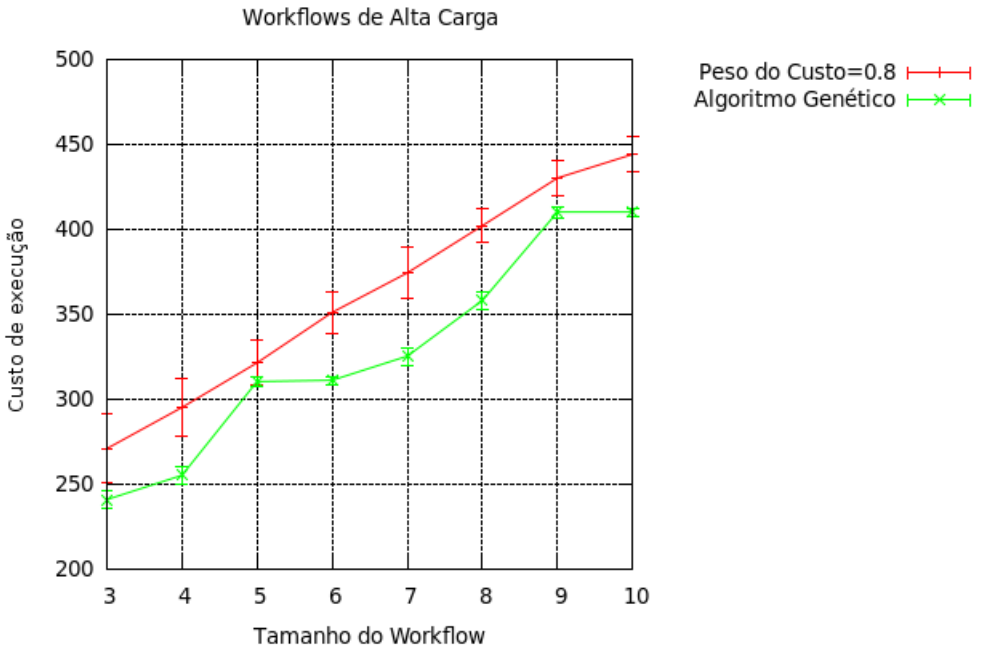

Figura 5.17: Custo de Execução em relação ao algoritmo genético para $M1 = 0.5$ e $M2 = 0.5$

carga. Entretanto, como a carga de trabalho é a mesma, a carga média apresenta comportamento semelhante.

Para analisar o custo de execução, primeiro é obtido um valor para cada tamanho de *workflow* a partir da execução da técnica proposta nesta dissertação. Esse valor é submetido ao escalonador genético como restrição orçamentária para a execução dos *workflows*. O que se

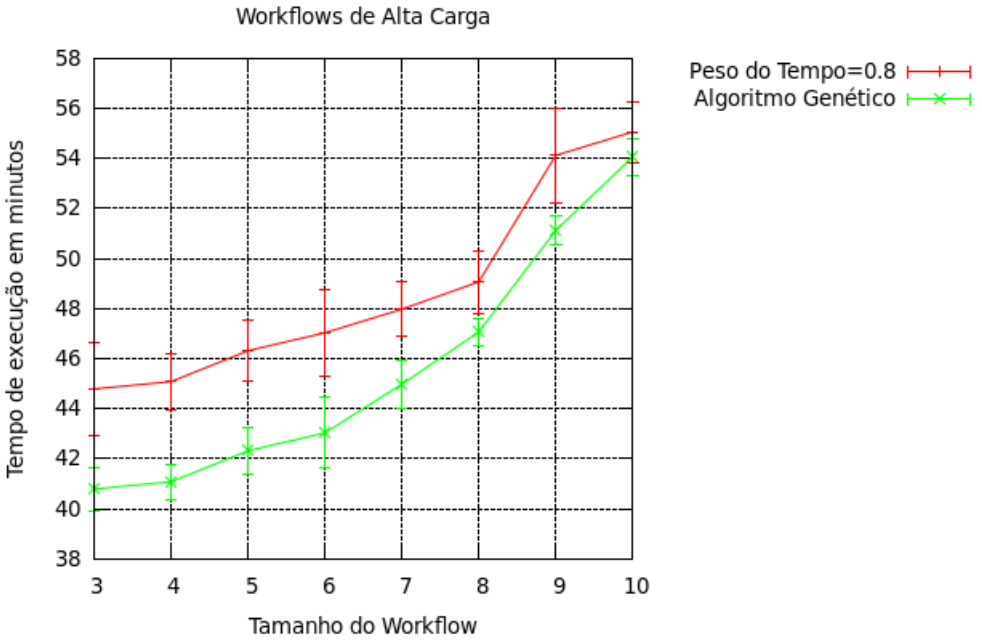

Figura 5.18: Tempo de Execução em relação ao algoritmo genético para $M1 = 0.5$ e $M2 = 0.5$

observa na figura 5.17 é que a solução proposta nesta dissertação consegue apresentar valores próximos aos fornecidos pelo escalonador genético mesmo sem contar com informações globais sobre todos os *peers* da grade.

Usando como restrição de prazo os valores encontrados pela técnica proposta, a solução proposta nesta dissertação também consegue valores próximos aos fornecidos pelo escalonador genético, como pode ser comprovado na figura 5.18. Por questão de simplicidade, os gráficos para valores de baixa e média carga foram omitidos, por não apresentarem variação em relação aos de alta carga. Também é importante notar que em ambas as métricas, os intervalos de confiança foram menores para o escalonador genético.

A razão para o escalonador centralizado encontrar soluções melhores do que as restrições impostas é maneira com que o *framework* JGAP trata a população inicial. No trabalho descrito na seção 3.4.1, a população inicial é determinada de forma aleatória. Os autores do *framework* utilizado afirmam que o poder computacional dos computadores modernos permitem a utilização de técnicas mais complexas na determinação da população inicial (MEFFERT et al., 2005). No caso do JGAP, é determinado por padrão que todo problema submetido para resolução tem sua população inicial criada através do uso de algoritmo guloso. Desta forma, as soluções iniciais são aproximações melhores do que indivíduos aleatórios. A execução posterior do algoritmo genético sobre essas soluções reforça a convergência para as partições do espaço de solução mais próximas do ótimo, retornando assim soluções de alta qualidade. Entretanto, a aplicação dos algoritmos utilizados pelo JGAP é pouco escalável, sendo que a medida

que o número de *peers* aumenta, o tempo de execução cresce de maneira não polinomial. A proposta desta dissertação consegue fornecer resultados próximos utilizando uma arquitetura descentralizada, evitando assim gargalos de desempenho.

5.6 Conclusão

Neste capítulo, apresentamos a arquitetura da implementação do ambiente descrito no capítulo 4. Para analisar a implementação, um conjunto de aplicações foi definido para representar os *workflows* submetidos à grade, além da carga de trabalho formada por tais aplicações e uma grade *testbed* para analisar o comportamento das técnicas consideradas.

O que pode ser observado na análise é que a técnica proposta nessa dissertação consegue estender a funcionalidade da estratégia do SwinDew com suporte a requisitos de QoS. Além disso, o escalonamento dinâmico utilizando a informação mais recente do estado da grade fornece um balanceamento de carga maior, comprovando o aspecto adaptativo da estratégia.

Já em relação à solução centralizada, a proposta continua apresentando melhor distribuição de carga. Porém, em relação ao cumprimento dos requisitos de QoS do usuário, os algoritmos genéticos oferecem melhores alternativas. Devido a limitações relativas a quantidade de recursos presentes no *testbed*, não foi possível demonstrar que a solução centralizada possui escalabilidade limitada em relação a proposta descentralizada.

6 Conclusões e Trabalhos Futuros

Esta dissertação propôs uma solução que permite aos administradores de grades computacionais estabelecerem um ambiente para execução *peer-to-peer* de *workflows*. O ambiente apresenta um alto nível de descentralização, fornecendo ao usuário a abstração da orquestração de serviços em *workflows* e possibilitando ao mesmo influenciar no escalonamento dos serviços ao informar prioridades entre as métricas de QoS. O foco deste trabalho são as aplicações científicas em execução nas grades computacionais. A seção 6.1 apresenta as principais contribuições e a seção 6.2 descreve os principais trabalhos futuros.

6.1 Contribuições

As seguintes contribuições relacionadas a execução de *workflows* em grades computacionais foram realizadas no desenvolvimento desta dissertação:

- A definição de uma estratégia *peer-to-peer* de melhor esforço (*best effort*) para o escalonamento de *workflow* em grades computacionais, descrita nas seções 4.4.1 e 4.4.2. Essa estratégia se beneficia da arquitetura de escalonamento com alto nível de descentralização e realiza uma tomada de decisão local a cada etapa do *workflow*, maximizando o cumprimento dos requisitos de QoS considerando informações recentes do estado da grade na avaliação de cada etapa. O aspecto de melhor esforço classifica a estratégia como adequada para as grades comunitárias, classificação detalhada na seção 2.2.

- O projeto e implementação de uma arquitetura para a execução *peer-to-peer* de *workflows* baseados na orquestração de serviços desenvolvida utilizando os paradigmas do *Globus Toolkit* e do WSRF (FOSTER, 2006). Essa arquitetura é descrita na seção 5.1. A implementação inclui um critério para formação de grupos de *peers* além de algoritmos para manutenção dos mesmos. O protótipo final apresenta funcionalidade completa para a execução de *workflows*, fato verificado pela utilização do ambiente na análise de aplicações reais.

- A análise comparativa da proposta em relação a soluções já existentes. Para efeito de comparação, na análise foi escolhida uma técnica de escalonamento centralizado tradicional baseada em algoritmos genéticos e outra heurística *peer-to-peer* que apresenta alto nível de descentralização e foi implementada pelos desenvolvedores do SwinDew. A análise é detalhada na seção 5.5.

Em geral, as contribuições apresentadas nessa dissertação promovem o avanço no estado da arte da execução descentralizada de *workflows* em grades. A técnica proposta pelos desenvolvedores do SwinDew foi aprimorada para considerar requisitos de QoS e informações atualizadas do estado da grade, com avanços comprovados em relação à estratégia original. O algoritmo genético ainda apresenta melhores resultados para o cumprimento das métricas, porém sua arquitetura é centralizada e não considera o balanceamento de carga do sistema.

6.2 Trabalhos Futuros

Dois trabalhos futuros principais são considerados. Primeiro está o estudo de como tornar mais intuitiva a definição dos pesos das métricas de QoS por parte do usuário. O segundo trabalho é o estudo de outros tipos de algoritmos evolutivos além dos genéticos, visando adaptá-los para execução descentralizada.

Para aprimorar a definição de métricas por parte do usuário e permitir que as informações da grade sejam traduzidas com maior facilidade para métricas administrativas, considera-se a utilização de tecnologias de *web* semântica para armazenamento dos meta-dados da grade. Já existem trabalhos consideram tal possibilidade (YAN; YANG; SHEN, 2005), entretanto a abordagem utilizada é limitada à submissão de tarefas. Com a adoção das tecnologias de serviços *web* semânticos e da descrição semântica de QoS para esses serviços (TONDELLO; SIQUEIRA, 2008), acredita-se que o nível de autonomia da grade seja aprimorado, além de facilitar o uso por parte de usuários não especialistas. Também espera-se que com informações estruturadas sobre a grade através de um formalismo robusto com a lógica por trás da *web* semântica, seja possível definir novas técnicas para representar quantativamente as métricas de qualidade de serviço levantadas na seção 2.3.

Algoritmos evolutivos são um campo de pesquisa fértil, apresentando várias técnicas para a resolução de problemas de otimização multi-objetivos (YU; GEN, 2010). O problema do escalonamento de *workflows* em grades computacionais se encaixa nessa classe de problemas. Algoritmos genéticos são as versões mais simples desse tipo de algoritmos, sendo que apresentaram bons resultados na avaliação. Portanto, é importante investigar a possibilidade de se

adaptar algoritmos evolutivos para arquiteturas descentralizadas, oferecendo um bom desempenho sem a presença de gargalos.

Além dos dois trabalhos futuros principais apresentados, outro trabalho importante para a usabilidade do sistema está em estudo. Trata-se de adaptar um ambiente visual de composição de *workflows* para submissão de instâncias ao ambiente de execução desenvolvido neste trabalho. Uma das opções levantadas nessa dissertação é o ambiente Triana (MATTHEW et al., 2003). Além de facilitar a utilização pelos usuários, o Triana ainda apresenta módulos de segurança e tolerância a falhas que complementariam o ambiente de execução apresentado nesta dissertação.

Por último, uma análise mais extensa é considerada para avaliar a escalabilidade do sistema e compará-lo com outras soluções. Em relação a escalabilidade, é interessante avaliar a proposta em um parque computacional mais amplo, já que um número maior de máquinas também permite a criação de *workflows* mais extensos. Considerando outras soluções, um cenário com mudanças na grade pode ser avaliado aplicando tanto a técnica de intercalar negociação com invocação do ambiente proposto quanto a arquitetura baseada em eventos do ASKALON (seção 3.4.2.4). O objetivo dessa nova análise seria avaliar qual opção é capaz de se adaptar com mais rapidez à alterações no contexto da grade.

Referências Bibliográficas

AHULLO, J. P.; LOPEZ, P. G. PlanetSim: an extensible framework for overlay network and services simulations. In: *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008. p. 1–1. ISBN 978-963-9799-20-2.

AKON, M. et al. Exchanging Peers to Establish P2P Networks. *Handbook of Peer-to-peer Networking*, Springer US, v. 1, n. 1, p. 143–165, 2010. ISSN 978-0-387-09750-3.

ANDERSON, D. P. et al. Seti@home: an experiment in public-resource computing. *Commun. ACM*, ACM, New York, NY, USA, v. 45, n. 11, p. 56–61, 2002. ISSN 0001-0782.

ANSELMI, J.; ARDAGNA, D.; CREMONESI, P. A QoS-based selection approach of autonomic grid services. In: *SOCP '07: Proceedings of the 2007 workshop on Service-oriented computing performance: aspects, issues, and approaches*. New York, NY, USA: ACM, 2007. p. 1–8. ISBN 978-1-59593-717-9.

ARAUJO, E. et al. The SegHidro experience: using the grid to empower a hydro-meteorological scientific network. *Proceedings of First International Conference on e-Science and Grid Computing*, 2005.

BARGA, R.; GANNON, D. Scientific versus Business Workflows. *Workflows for e-Science, Springer*, 2007.

BERNHOLDT, D. et al. The Earth System Grid: Supporting the next generation of climate modeling research. *Proceedings of the IEEE*, p. 485–495, 2005.

BERRIMAN, G. B. et al. Workflows for e-Science. In: ____. [S.l.]: Springer US, 2007. cap. Generating Complex Astronomy Workflows, p. 19–38.

BRANDES, U. et al. GraphML Primer. *Disponível em http://graphml.graphdrawing.org/primer/graphml-primer.html, último acesso 03/2010*, 2005.

BUYYA, R.; MURSHED, M. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, v. 14, n. 13, p. 1175 – 1220, 2002.

CARDOSO, A. R. *Architecture Basée sur les Reséaux Programmables et es Réseaux Pair-à-pair*. Paris, FR: École Doctorale d'Informatique, Telécommunications et Électronique de Paris, 2007.

CIRNE, W. et al. Labs of the World, Unite!!! *Journal of Grid Computing, Springer*, p. 225–246, 2006.

CIRNE, W.; BRASILEIRO, F.; PARANHOS, D. Trading Cycles for information: Using replication to schedule bag-of-tasks applications on computational Grids. *Proceedings of Euro-par'2003*, 2003.

CIRNE, W.; SANTOS-NETO, E. Grids Computacionais: da Computação de Alto Desempenho a Serviços sob Demanda. *Minicurso do 23º Simpósio Brasileiro de Redes de Computadores*, 2005.

CORCHO, O.; LOSADA, S.; BENJAMINS, R. Mediation: Bridging between Heterogeneous Web Service Systems. *Semantic Web Services: Concepts, Technologies and Applications, Springer*, 2007.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. Distributed Systems Concepts and Design. *Addison Wesley, Fourth Edition*, 2005.

DEELMAN, E. Looking into the Future of Workflows: The Challenges Ahead. *Workflows for e-Science, Springer*, 2007.

ENGELBRECHT, G.; BENKNER, S. A Service-Oriented Grid Environment with On-demand QoS Support. In: *SERVICES '09: Proceedings of the 2009 Congress on Services - I*. Washington, DC, USA: IEEE Computer Society, 2009. p. 147–150. ISBN 978-0-7695-3708-5.

FAHRINGER, T. et al. Askalon: A Grid Application Development and Computing Environment. In: *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2005. p. 122–131. ISBN 0-7803-9492-5.

FANNING, S. The Napster Protocol. 1999. Disponível em http://opennap.sourceforge.net/.

FERNANDEZ-BACA, D. Allocating Modules to Processors in a Distributed System. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 15, n. 11, p. 1427–1436, 1989. ISSN 0098-5589.

FERRARI, D.; ZHOU, S. An Empirical Investigation of Load Indices For Load Balancing Applications. In: *Performance '87, the 12th Internation Symposium On Computer Performance Modeling, Measurement, and Evaluation,*. Amsterdam, The Netherlands: North Holland, 1988. p. 512–528.

FOSTER, I. Globus Toolkit Version 4: Software for Service-Oriented Systems. *IFIP International Conference on Network and Parallel Computing*, p. 2–13, 2006.

FOSTER, I.; IAMNITCHI, A. On death, taxes, and the convergence of peer-to-peer and grid computing. In: *In 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03*. [S.l.: s.n.], 2003. p. 118–128.

FOSTER, I.; KESSELMAN, C. *The Grid: Blueprint for a New Computing Infrastructure, Chapter 2*. [S.l.]: Morgan-Kaufman, 1999.

FOSTER, I. et al. The Open Grid Services Architecture, version 1.0. *Global Grid Forum (GGF)*, 2005.

FROTA, A. M. C. *M-CODE: Um Modelo para Medição de Confidencialidade e Desempenho para Aplicações Móveis Seguras*. Ceará, BR: Dissertação de Mestrado do Departamento de Computação da Universidade Federal do Ceará, 2008.

GAMMA, E. et al. Design Patterns: Elements of Reusable Object-Oriented Software. *Addison-Wesley*, 1995.

GANNON, D. Component Architectures and Services: From Application Construction to Scientific Workflows. *Workflows for e-Science, Springer*, 2007.

GANNON, D. et al. Dynamic, Adaptative Workflows for Mesoscale Meteorology. *Workflows for e-Science, Springer*, 2007.

GIL, Y. From Data to Knowledge to Discoveries: Scientific Workflows and Artificial Intelligence. *Scientific Programming, Volume 16, Number 4, 2008*, 2008.

GIL, Y. et al. Wings for Pegasus: Creating Large-Scale Scientific Applications Using Semantic Representations of Computational Workflows. *Proceedings of the Nineteenth Conference on Innovative Applications of AI (IAAI-07)*, 2007.

GOBLE, C. A. et al. Enhancing Services and Applications with Knowledge and Semantics. *The Grid: Blueprint for a New Computing Infrastructure, 2nd ed., I. Foster and C. Kesselman, eds., Morgan Kaufmann*, 2003.

GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0201157675.

HOLLINSWORTH, D. The Workflow Reference Model. *Workflow Management Coalition,TC00-1003*, 1994.

KON, F.; GOLDMAN, A. Grades Computacionais: Conceitos Fundamentais e Casos Concretos. *Jornada de Atualização em Informática do Congresso da Sociedade Brasileira de Computação*, 2008.

KRAUTER, K.; BUYYA, R.; MAHESWARAN, M. A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing. *Software: Practice and Experience*, 2002.

LAUSEN, H. et al. Description: Semantic Annotation for Web Services. *Semantic Web Services: Concepts, Technologies and Applications, Springer*, 2007.

LUA, E. K. et al. A survey and Comparison of Peer-to-Peer Overlay Network Schemes. *IEEE Communications Surveys and Tutorials*, v. 7, p. 72–93, 2005.

LUDWIG, S. A.; REYHANI, S. Selection Algorithm for Grid Services based on a Quality of Service Metric. *High Performance Computing Systems and Applications, Annual International Symposium on*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 13, 2007.

MADADHAIN, J.; FISHER, D.; NELSON, T. Java Universal Network/Graph Framework. *Disponível em http://jung.sourceforge.net/, último acesso 03/2010*, 2007.

MAECHLING, P. et al. SCEC Cybershake Workflows—Automating Probabilistic Seismic Hazard Analysis Calculations. *Workflows for e-Science, Springer*, 2007.

MANI, A.; NAGARAJAN, A. Understanding quality of service for Web services. *Disponível em http://www.ibm.com/developerworks/library/ws-quality.html, último acesso 09/2009*, 2002.

MARTINS, F. S. et al. Detecting misbehaving units on computational grids. *Concurrency and Computation: Practice and Experience*, v. 22, n. 3, p. 329–342, 2010.

MASSIE, M. L.; CHUN, B. N.; CULLER, D. E. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, v. 30, n. 7, p. 817 – 840, 2004. ISSN 0167-8191. Disponível em: <http://www.sciencedirect.com/science/article/B6V12-4CMHWWX-2/2/b6b44ba67c732867d1c3881c510b2953>.

MATTHEW, I. T. et al. Grid Enabling Applications Using Triana. In: *In Workshop on Grid Applications and Programming Tools*. [S.l.: s.n.], 2003.

MATTOSO, M. et al. Gerenciando Experimentos Científicos em Larga Escala. *Anais do XXVIII Congresso da SBC*, 2008.

MEFFERT, K. et al. Java Genetics Algorithm Package. *Disponível em http://jgap.sourceforge.net/, último acesso 06/2010*, 2005.

MORAIS, M.; CARDOSO, P.; GOMES, D. Controle dinâmico de QoS baseado em políticas. *XXVI Simpósio Brasileiro de Telecomunicações (SBrT 2008)*, 2008.

MURY, A. R.; SCHULZE, B.; GOMES, A. T. A. Task distribution models in grids: towards a profile-based approach. *Concurr. Comput. : Pract. Exper.*, John Wiley and Sons Ltd., Chichester, UK, v. 22, n. 3, p. 358–374, 2010. ISSN 1532-0626.

OAKS, S.; GONG, L. *JXTA in a Nutshell*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2002. ISBN 059600236X.

OCI Office of Cyberinfrastructure National Science Foundation's Cyberinfrastructure Council. Cyberinfrastructure Vision for 21st Century Discovery. *Disponível em http://www.nsf.gov/pubs/2007/nsf0728, último acesso 03/2009*, 2007.

OINN, T. et al. Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows. *Bioinformatics, Volume 20 , Number 17, Oxford University Press*, 2004.

OINN, T. et al. Taverna: Lessons in Creating a Workflow Environment for the Life Sciences. *Concurrency and Computation: Practice and Experience*, 2005.

OLIVEIRA, D. de et al. Uma Abordagem Semântica para Linhas de Experimentos Científicos Usando Ontologias. *III Workshop de e-Science em conjunto com o SBBD 2009*, 2009.

OPEN GROUP, S. W. G. of the. Definition of SOA. *Disponível em http://opengroup.org/projects/soa/doc.tpl?gdid=10632, último acesso 09/2009*, 2009.

ORACLE. Oracle Grid Computing. *Disponível em http://www.oracle.com/, último acesso 09/2009*, 2009.

PEIXOTO, M. L.; SANTANA, M. J.; SANTANA, R. H. A P2P Hierarchical Metascheduler to Obtain QoS in a Grid Economy Services. *Computational Science and Engineering, IEEE International Conference on*, IEEE Computer Society, Los Alamitos, CA, USA, v. 1, p. 292–297, 2009.

PEREIRA, W. M.; TRAVASSOS, G. H. Abordagem para concepção de experimentos científicos em larga escala suportados por workflows científicos. *III Workshop de e-Science em conjunto com o SBBD 2009*, 2009.

PREIST, C. Goals and Vision: Combining Web Services with Semantic Web Technology. *Semantic Web Services: Concepts, Technologies and Applications, Springer*, 2007.

PRODAN, R.; FAHRINGER, T. Dynamic scheduling of scientific workflow applications on the grid: a case study. In: *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2005. p. 687–694. ISBN 1-58113-964-0.

QIN, J.; FAHRINGER, T. A Novel Domain Oriented Approach for Scientific Grid Workflow Composition. In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC08)*. Austin, Texas, USA: IEEE Computer Society Press, 2008.

RANJAN, R.; RAHMAN, M.; BUYYA, R. A decentralized and Cooperative Workflow Scheduling Algorithm. In: *In Proceedings of 8 th IEEE International Symposium on Cluster Computing and the Grid*. [S.l.: s.n.], 2008.

ROURE, D. D.; JENNINGS, N. R.; SHADBOLT, N. R. The Semantic Grid: Past, Present, and Future. *Proceedings of the IEEE,Volume 93, Issue 3, March 2005*, 2005.

SCHWARZ, K.; BLAHA, P. Solid state calculations using WIEN2k. *Computational Materials Science*, v. 28, n. 2, p. 259 – 273, 2003. ISSN 0927-0256. Proceedings of the Symposium on Software Development for Process and Materials Design. Disponível em: <http://www.sciencedirect.com/science/article/B6TWM-49HDWCS-5/2/7b5a2c32cd298fc877bee1e5521205f5>.

SHIELDS, M. Control versus Data Driven Workflows. *Workflows for e-Science, Springer*, 2007.

SHONKWILER, R. W.; MENDIVIL, F. Explorations in monte carlo methods. In: ____. [S.l.]: Springer US, 2009. cap. Introduction to Monte Carlo Methods, p. 1–49.

SINAPAD. Projeto GradPAD. 2010. Disponível em http://www.lncc.br/sinapad/projetos.php.

SOTOMAYOR, B.; CHILDERS, L. Globus Toolkit 4: Programming Java Services. *The Elsevier Series in Grid Computing*, 2005.

TALUKDER, A. K. M. K. A.; KIRLEY, M.; BUYYA, R. Multiobjective differential evolution for scheduling workflow applications on global grids. *Concurr. Comput. : Pract. Exper.*, John Wiley and Sons Ltd., Chichester, UK, v. 21, n. 13, p. 1742–1756, 2009. ISSN 1532-0626.

TANENBAUM, A.; STEEN, M. V. Distributed Systems Principles and Paradigms. Prentice Hall, New York, NY, USA, p. 33–34, 2007.

TAYLOR, I.; SHIELDS, M.; WANG, I. Resource management for the Triana peer-to-peer services. Kluwer Academic Publishers, Norwell, MA, USA, p. 451–462, 2004.

TAYLOR, I. et al. The Triana Workflow Environment: Architecture and Applications. Springer, p. 320–339, 2007.

TONDELLO, G. F.; SIQUEIRA, F. The QoS-MO ontology for semantic QoS modeling. In: *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2008. p. 2336–2340. ISBN 978-1-59593-753-7.

VCG Virtual Community Grid. Portal de Submissão. 2010. Disponível em http://vcg.lncc.br/index.php, último acesso 07/2010.

VIANA, G. V. R. *Metaheurísticas e Programação Paralela em Otimização Combinatória*. Fortaleza, CE, BR: Edições UFC, 1998.

W3C Consortium. Soap Version 1.2 Part 0: Primer (second edition). *Disponível em http://www.w3.org/TR/2007/REC-soap12-part0-20070427/, último acesso 03/2009*, 2007.

W3C, W. Wide Web Consortium. Web Services Description Language. *Disponível em http://www.w3.org/TR/wsdl, último acesso 09/2009*, 2009.

W3C, W. Wide Web Consortium. Xml Schema. *Disponível em http://www.w3.org/XML/Schema, último acesso 09/2009*, 2009.

WEERATUNGA, D. et al. The NAS Parallel Benchmarks. *NAS Technical Report RNR-94-007*, NASA Ames Research Center, Moffett Field, CA, USA, 1994.

WFMC. Workflow Management Coalition. *Disponível em http://www.wfmc.org/, último acesso 09/2009*, 2009.

WINJGAART, R.; FUMKIN, M. NAS Grid Benchmarks Version 1. *NAS Technical Report NAS-02-005*, NASA Ames Research Center, Moffett Field, CA, USA, 2002.

YAN, J.; YANG, Y.; RAIKUNDALIA, G. Swindew, a P2P-based Decentralized Workflow Management System. *IEEE Transactions on Systems, Man, and Cybernetics Volume 36 Number 5*, 2006.

YAN, J.; YANG, Y.; SHEN, J. A P2P-based Service Flow System with Advanced Ontology-Based Service Profiles. *IEEE Transactions on Systems, Man, and Cybernetics Volume 36 Number 5*, 2005.

YAN, J.; YANG, Y.; SHEN, J. Swindew-s: Extending P2P Workflow Systems for Adaptive Composite Web Services. *Proceedings of the Australian Software Engineering Conference*, 2006.

YANG, Y. et al. Peer-to-peer Based Grid Workflow Runtime Environment of Swindew-g. *Third IEEE International Conference on e-Science and Grid Computing*, 2007.

YU, J.; BUYYA, R. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Sci. Program.*, IOS Press, Amsterdam, The Netherlands, The Netherlands, v. 14, n. 3,4, p. 217–230, 2006. ISSN 1058-9244.

YU, J.; BUYYA, R. A Taxonomy of Workflow Management Systems for Grid Computing. *Journal of Grid Computing - Springer*, 2006.

YU, J.; BUYYA, R.; THAM, C. K. Cost-based Scheduling of Workflow Application on Utility Grids. *Proceedings of the First IEEE International Conference on e-Science and Grid Computing*, 2005.

YU, X.; GEN, M. *Introduction to Evolutionary Algorithms*. London, UK: Springer-Verlag London, 2010.

ZHANG, Q.; LI, Z. Design of grid resource management system based on divided min-min scheduling algorithm. In: *ETCS '09: Proceedings of the 2009 First International Workshop on Education Technology and Computer Science*. Washington, DC, USA: IEEE Computer Society, 2009. p. 613–617. ISBN 978-0-7695-3557-9.