Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

Otimização do método de escalonamento de tarefas de uma ferramenta computadorizada para workflow científico distribuído no contexto da modelagem computacional

Felipe Rooke da Silva

JUIZ DE FORA
JULHO, 2014

Otimização do método de escalonamento de tarefas de uma ferramenta computadorizada para workflow científico distribuído no contexto da modelagem computacional

Felipe Rooke da Silva

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Ciro de Barros Barbosa

JUIZ DE FORA

JULHO, 2014

Otimização do método de escalonamento de tarefas de uma ferramenta computadorizada para workflow científico distribuído no contexto da modelagem computacional

Felipe Rooke da Silva

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Ciro de Barros Barbosa
Prof. Doutor em Ciência da Computação

Jairo Francisco de Souza
Prof. Doutor em Engenharia de Sistemas e Computação

Stenio Sã Rosário Furtado Soares
Prof. Doutor em Computação

JUIZ DE FORA

30 DE JULHO, 2014

Resumo

Cada vez mais a computação está presente no desenvolvimento científico e nesse contexto são utilizadas ferramentas para auxiliar os pesquisadores. Uma delas será tratada neste trabalho: o Sistema de *Workflow* Científico, *ad hoc*, denominado "*Scientific Workflow Automation in Distributed Environment*" (SWADE). O SWADE é um sistema, baseado em serviços web que permite ao pesquisador focar sua atenção na elaboração e execução de um *workflow*, minimizando a manipulação de conceitos técnicos relacionados ao ambiente computacional distribuído e heterogêneo. O SWADE é composto por vários elementos arquiteturais e este trabalho tem como objetivo modificar essa arquitetura afim de otimizar um elemento específico, o escalonador de tarefas. A arquitetura passou de descentralizada, para uma arquitetura híbrida com escalonador centralizado. Foram implementados também dois tipos de escalonadores, um utilizando uma abordagem "*First Come First Serve*" (FCFS) e outra utilizando o algoritmo "*Greedy Randomized Adaptive Search Procedure*" (GRASP). Através de testes de desempenho foi possível gerar evidências que apontam para uma melhoria significativa no tempo de execução total de um *workflow*, validando assim as soluções propostas.

Palavras-chave: GRASP, escalonador GRASP, e-science, *workflow* científico, escalonamento, escalonamento em *grid*, escalonador FCFS, sistema de *workflow* científico.

Abstract

Increasingly computation is present in scientific development nowadays and due to this importance new tools are appearing in the market to support the researchers. In this paper it will be discussed one of them: the Scientific Workflow system, ad hoc named "Scientific Workflow Automation in Distributed Environment" (SWADE). The SWADE is a system based in *web services* that allows researchers to focus more attention on the development and execution of scientific workflow, minimizing the manipulation of technical concepts related to distributed and heterogeneous computing environment. The SWADE is composed of several architectural elements and this paper aims to modify its design to optimize an specific element, the scheduler tasks. The decentralized architecture has been changed into a hybrid architecture with a centralized scheduler. Two types of schedulers have also been implemented, an approach using a "First Come First Serve" (FCFS) and another using the algorithm "Greedy Randomized Adaptive Search Procedure" (GRASP). Through performance tests it was possible to generate evidences pointing to a significant improvement in the total execution time of a workflow, validating, thus, the proposed solutions.

Keywords: GRASP, GRASP schedule, e-science, scientific workflow, scheduling, *grid* scheduling, FCFS scheduler, scientific workflow system.

Agradecimentos

Primeiramente aos meus pais e irmãos, por todo suporte, incentivo, amor e ensinamentos essenciais a formação do meu caráter. Sem eles nada seria possível.

A todos os meus parentes, pelo encorajamento e apoio.

A minha namorada e amigos pela paciência, companheirismo e compreensão, principalmente nesses últimos meses.

Aos professores e colegas de trabalho do Grupo de Educação Tutorial da Computação (GETComp) e do Centro de Educação a Distância (CEAD) da UFJF, pelas oportunidades e palavras de incentivo.

Ao professor Ciro Barbosa pela orientação, amizade e principalmente, pela paciência, sem a qual este trabalho não se realizaria.

Ao doutorando Aldemon, que muito me ajudou na realização deste trabalho.

Aos professores do Departamento de Ciência da Computação, pelos ensinamentos durante esses anos, que contribuiram muito para minha formação.

"Agir, eis a inteligência verdadeira. Serei o que quiser. Mas tenho que querer o que for. O êxito está em ter êxito, e não em ter condições de êxito. Condições de palácio tem qualquer terra larga, mas onde estará o palácio se não o fizerem ali?".

Fernando Pessoa

Sumário

Lista de Figuras

Lista de Algoritmos

Lista de Códigos

Lista de Abreviações

DCC	Departamento de Ciência da Computação
UFJF	Universidade Federal de Juiz de Fora
WC	*Workflow* Científico
SWC	Sistema de *Workflow* Científico
SWADE	Scientific *Workflow* Automation in Distributed Environment
SINAPAD	Sistema Nacional de Processamento de Alto Desempenho
SOAP	Simple Object Access Protocol
WSDL	Web Services Description Language
ETC	Expected Time to Compute
TPCC	Total Processor Cycle Comsumption
GIS	Grid Information System
FCFS	First Come First Serve
FIFO	First In First Out
GRASP	Greedy Randomized Adaptive Search Procedure
HC	Hill Climbing
SA	Simulated Annealing
TS	Tabu Search
SIGAR	System Information Gatherer and Reporter

1 Introdução

1.1 Contextualização

Cada vez mais a computação está presente no desenvolvimento científico e já pode ser considerada como o "terceiro pilar" da pesquisa científica (Taylor et al, 2006). Em variados campos da ciência, tais como ciências da terra, bioinformática, medicina e ciências sociais, existe um grande esforço computacional envolvido nas pesquisas e muitas vezes gerando volumes muito grandes de dados. Para tratar tal volume de dados, se faz necessária a exploração de recursos computacionais que vão desde paralelismo a uma arquitetura integrada de nós de redes, através de *clusters* e *grids*.

Nesse contexto, muitos trabalhos de pesquisas nas mais diversas áreas fazem uso de modelos computacionais para serem desenvolvidos. Para auxiliar esses trabalhos, pode-se fazer uso de um Sistema de *WorkFlow* Científico (SWC). "Dá-se o nome de *Workflow* Científico para o conjunto das tarefas que compõem um experimento em modelagem computacional" (Bonifácio, 2008). Segundo Altintas et al (2006), *workflow* científico é um processo automatizado que combina dados e processos em uma sequência de passos estruturados para implementar soluções computacionais para problemas científicos.

Tais *Workflows* Científicos (WC) vêm apresentando um crescimento muito grande nos últimos anos: a ciência torna-se cada vez mais dependente da análise de grandes conjuntos de dados e do uso de recursos distribuídos. O paradigma de programação de *workflow* é visto como um meio de gerenciar a complexidade na definição dessa análise. Essa gerência é realizada executando as computações necessárias sobre os recursos distribuídos, coletando informações sobre os resultados e fornecendo meios para gravar e reproduzir a análise científica (Taylor et al, 2006). Tais *workflows*, podem ser implementados através de um Sistema de *Workflow* Científico (SWC).

Sistemas de *workflow* científico ainda são pouco difundidos. Isso se deve, em parte, ao esforço necessário para realização do aporte tecnológico e conceitual envolvidos em sua utilização. Ferramentas dessa natureza disponibilizam um grande número de

funcionalidades, muitas delas fazendo uso de tecnologias desconhecidas para grande parte dos pesquisadores (Bonifácio, 2008).

Umas das vantagens de se utilizar SWC no contexto da modelagem computacional, segundo Bonifácio (2008), é a divisão do trabalho e o controle avançado de sua execução. Por divisão do trabalho entende-se a execução de tarefas em máquinas remotas ou até mesmo em clusters. O controle avançado da execução é o acompanhamento da computação da tarefa, com opções de visualização de variáveis, de pausa, resumo e parada da computação.

Um exemplo de SWC é o *Scientific Workflow Automation in Distributed Environment* (SWADE), que será utilizado como base para as implementações e experimentos deste trabalho. O SWADE é um sistema baseado em serviços web que permite ao pesquisador focar a atenção na elaboração e execução da heurística, minimizando a manipulação de conceitos técnicos do ambiente computacional distribuído e heterogêneo (Bonifácio et al, 2014).

A arquitetura da rede formada pelo SWADE é constituida de nós que se comportam tanto como clientes quanto como servidores. Cada nó contém uma arquitetura interna, formada por um servidor web java, um servidor SWADE, ações, arquivos públicos e arquivos de dados.

Em um fluxo resumido de funcionamento desse sistema, um cliente se conecta a um nó da rede formada pelo SWADE, fornece a ele a descrição de um *workflow* científico, arquivos caso sejam necessários e o endereço dos outros nós da rede. O mesmo nó que recebe as solicitações é responsável por interpretar o *workflow* científico. Para toda tarefa descrita pelo WC, é feito, pelo nó interpretador uma chamada a um escalonador (*scheduler*), que por sua vez decide para qual nó da rede o interpretador irá repassar a tarefa a ser executada. Repassada e executada a tarefa, o nó alvo retorna o resultado obtido para o interpretador e este da seguimento ao fluxo do *workflow*.

O *scheduler* é o componente responsável por definir em qual nó da rede determinada tarefa será executada. No SWADE essa decisão é baseada no algoritmo de escalonamento por chaveamento circular (*Round-Robin*). Segundo Tanenbaum (2007), a cada processo é atribuído um intervalo de tempo, o seu *quantum*, no qual ele é permitido

executar. Além disso, o escalonador só precisa manter uma lista de processos executáveis, quando um processo termina o seu *quantum*, ele é colocado no final da lista.

O algoritmo *Round-Robin* implementado pelo escalonador do SWADE mantém uma lista de endereços de todos os nós da rede e um ponteiro apontado para o último nó ao qual foi atribuído uma tarefa. À medida que o escalonador é acionado, é retornado o nó ao qual será atribuído uma tarefa e o ponteiro se desloca para o próximo nó. Quando o ponteiro chega ao final da lista, o próximo endereço a ser apontado será o primeiro.

O uso do algoritmo baseado no *Round-Robin*, pelo escalonador gera um desperdício de recursos. Num cenário de uso onde temos uma quantidade grande de nós e são atribuídas tarefas de maneira circular aos mesmos, se uma tarefa é executada quase que instantaneamente, ela liberaria recursos do nó onde se encontrava. Porém, só seria alocada outra tarefa ao nó livre quando o escalonador completasse seu ciclo e alcançasse novamente o mesmo.

Um outro problema encontrado é referente à arquitetura da rede formada pelo SWADE. Trata-se de uma arquitetura descentralizada. Como mencionado anteriormente, todo nó apresenta um servidor SWADE, logo podemos ter um cenário onde dois clientes distintos se conectam a dois nós da rede para a execução de dois *workflows*, "A" e "B". Assim que executados e suas tarefas distribuidas pelos nós, as tarefas do *workflow* "A" competiriam com as tarefas do *workflow* "B" em um mesmo nó. Ampliando este cenário para outro em que temos vários *workflows*, pode-se perceber uma perda de eficiência, já que a vazão, número de tarefas por hora que o sistema termina (Tanenbaum, 2007), diminui.

Apresentada estas informações, justifica-se uma análise e implementação de uma melhor forma de escalonamento, assim como uma arquitetura mais adequada. Desta maneira, o objetivo deste trabalho vai ao encontro desta necessidade e propõe tornar a maneira como o SWADE escalona suas tarefas mais eficiente, de forma que o número de tarefas terminadas por intervalo de tempo seja aprimorado. Para tais melhorias serem alcançadas foram elaborados os objetivos específicos a seguir:

- Implementar um algoritmo baseado em fila para escalonar as tarefas geradas pelo sistema de *workflow* científico. O uso desse escalonador diminui a quantidade de

nós ociosos durante a execução de um WC.

- Implementar um algoritmo baseado em inteligência computacional para escalonar as tarefas geradas pelo sistema de *workflow* científico. Esse escalonador permite um melhor aproveitamento dos recursos da rede, combinando as necessidades das tarefas aos nós que melhor podem atende-la.

1.2 Estrutura do Trabalho

Este trabalho foi elaborado em seis capítulos. Este capítulo faz uma introdução ao tema e expõe brevemente os problemas e objetivos propostos. O Capítulo 2 apresenta a fundamentação teórica do tema abordando os principais conceitos relacionados às tecnologias envolvidas nesta monografia. O Capítulo 3 mostra as soluções propostas aos problemas destacados nos capítulos anteriores. O Capítulo 4 expõe os testes e seus critérios realizados, depois de aplicadas as soluções propostas no Capítulo 3. O Capítulo 5 expõe trabalhos futuros passíveis de realização e, por fim, o Capítulo 6 apresenta a conclusão desta monografia.

2 Fundamentação Teórica

Neste capítulo apresentaremos os principais conceitos relacionados a este trabalho. A Seção 2.1 faz uma contextualização sobre e-Science e em que contexto este trabalho está inserido. A Seção 2.2 apresenta uma visão geral sobre *workflows* científicos e sua importancia para a pesquisa científica. A Seção 2.3 aborda um contexto mais específico sobre SWC (Sistemas de *Workflow* Científico) e Computação em *Grid*. As seções seguintes, 2.4, 2.5, 2.6 e 2.8 estão relacionadas de maneira mais direta ao sistema adotado neste trabalho, que por sua vez está apresentado na seção 2.7.

2.1 E-Science

Sistemas computacionais tem se tornado importantes para pesquisa ciêntífica, dando suporte para todos os aspectos relacionados ao seu ciclo de vida. A comunidade científica tem utilizado os termos *e-Science* e *e-Research* para englobar o importante papel das tecnologias computacionais na pesquisa, colaboração, compartilhamento de dados e documentos, e uso de recursos para automatizar a execução e análise de dados de experimentos científicos (Parastatidis, 2009).

Cada vez mais as comunidades científicas percebem os benefícios que o uso de recursos computacionais, compartilhamento de dados e serviços têm sob seus trabalhos, além de contribuir para a construção de uma infraestrutura de dados e uma comunidade científica distribuida (Ludäscher et al, 2006).

Segundo Hey e Fox (2005), e-Science pode ser entendido como a infraestrutura que tem por objetivo permitir que cientistas e pesquisadores tenham acesso a dados primários distribuídos, utilizando acesso remoto aos mesmos, e além disso promover algo que vá além da infraestrutura computacional, ou seja, a "capacidade de acessar, mover, manipular e extrair dados é a exigência central dessas novas aplicações das ciências da colaboração".

Embora e-Science especificamente seja desenvolvido em sua maioria por progra-

mas britânicos, existem atividades similares ao redor do mundo, incluindo "*Cyberinfrastructure*" nos Estados Unidos e "*e-Infrastructure*" na Europa Continental (Hey e Fox , 2005). No Brasil, existem diversas infraestruturas sendo desenvolvidas e dentre elas podemos citar: SINAPAD[1] (Sistema Nacional de Processamento de Alto Desempenho) e GridUNESP[2].

A Computação em Grade ou *Grid* é uma tecnologia bastante relacionada às atividades de e-Science. Um *Grid* computacional gerencia o compartilhamento de recursos e soluções de problemas em organizações virtuais. A ideia principal deste conceito é a habilidade de negociar compartilhamento de recursos distribuídos em um conjunto de partes participantes, e assim usar os resultados desta união de recursos para algum propósito.

A computação em *grid* será tratada mais a frente na Seção 2.3.

2.2 Workflow Científico

Segundo Hollingsworth e Hampshire (1993), *workflow* consiste da automação de um processo de negócio, na sua totalidade ou parte, no qual documentos, informações ou tarefas são passadas de um participante para outro por nós, de acordo com um conjunto de regras procedurais.

Uma definição de *workflow* denota a execução controlada de múltipas tarefas em um ambiente de elementos processados de forma distribuída. *Workflows* representam um conjunto de atividades que podem ser executadas com suas relações interdependentes, suas entradas e suas saídas (Seffino et al, 1999).

Altintas et al (2006) define *workflow* científico como um processo automatizado que combina processos e dados em um conjunto de passos estruturados para implementar soluções computacionais para um problema científico.

Tecnologias de *workflows* científicos surgiram para permitir que os pesquisadores criem e executem experimentos dado um conjunto de serviços e recursos disponíveis em determinado ambiente. Linguagens e Ferramentas foram projetadas para capturar o fluxo de informações entre serviços (por exemplo, serviços de endereçamento e relações de

[1]https://www.lncc.br/sinapad/index.php
[2]http://www.unesp.br/portal#!/gridunesp/

entrada e saída) (Pignotti et al, 2011).

A modelagem de *workflows* científicos ainda é muito difícil, em contrapartida, existe um maior domínio na modelagem de *workflows* para negócios (*business workflows*). As características deste segundo tipo de *workflow* são bem diferentes das características do primeiro. A diferença mais notável entre os *workflows* científicos e de negócios é a escala. Quando comparados aos *workflows* científicos, os *workflows* para negócios normalmente são bem menores. Os *workflows* científicos podem envolver centenas de instâncias de serviços, que terão que ser modeladas. Além disso, os *workflows* científicos, na maioria das vezes, vão executar centenas de invocações de serviços básicos e, consequentemente, vão enviar centenas de mensagens, que serão trocadas entre os serviços. Os *workflows* para negócios, normalmente, operam em pequena escala (Taylor et al, 2006).

De uma maneira geral, ainda que a modelagem de *workflows* científicos possa ser muitas vezes complexa, existem softwares que têm como objetivo acelerar e facilitar tal procedimento, tais softwares são denominados Sistemas de *Workflow* Científico (SWC).

2.3 Sistemas de Workflow Científico e Grids

A Computação em *Grid* e tecnologias relacionadas, surgiram principalmente para satisfazer o aumento crescente da demanda da comunidade de computação científica por um maior poder de computação. Computadores distribuídos geograficamente, ligados através da Internet, são usados para criar super computadores virtuais com grande quantidade de recursos e capacidade de computação, capaz de resolver problemas complexos em menos tempo do que o conhecido antes (Xhafa e Abraham, 2010).

No contexto de *e-Science*, simulações que se utilizam de *grids* normalmente são as que envolvem sequencias de atividades como a busca por recursos, submissão de *jobs*, transferência de arquivos, análise e coleta de dados. É possível utilizar um serviço baseado em *workflow* para automatizar a coordenação entre tarefas sem que haja um controle humano. Assim, cada tarefa e código científico herdado pode se apresentar como um web service, simplificando o desenvolvimento e a manutenção do *workflow* (Yang et al, 2010).

A gestão do ciclo de vida de um SWC (criação, execução e monitoramento) não é uma tarefa trivial. Ela exige uma plataforma integrada que permita a criação de *workflows*

utilizando *web services*, gerenciamento da execução de tais *workflows*, interpretação das definições do processo e interação com outros *workflows* participantes.(Yang et al, 2010)

Segundo Lemos (2004), um SWC deve atender a alguns requisitos, são eles:

- Incluir os processos, dados e recursos normalmente usados e oferecer mecanismos de extensibilidade para acomodar novos processos, dados e recursos. Entre os processos estão os programas de análise de bioinformática e de domínios específicos, programas que filtram os resultados de outros programas e mecanismos de controle de execução do *workflow*.

- Ajudar os usuários na definição e redefinição do *workflow*. A redefinição é importante quando os resultados finais não forem considerados úteis ou interessantes pelos pesquisadores. Para tanto, o sistema deve permitir a definição de propriedades dos processos, dados e recursos de tal forma a facilitar a escolha dos mais adequados para cada caso exposto pelo pesquisador.

- Oferecer ferramentas para validar o *workflow* definido pelo pesquisador. Durante a validação, o sistema deve verificar se as entradas e saídas definidas pelos usuários para cada processo do *workflow* são coerentes, além de incluir, caso seja necessário, processos que façam conversão nos formatos dos dados e processos que verifiquem se os resultados gerados pelos programas são esperados ou não.

- Ser capaz de otimizar e executar o *workflow* definido pelo pesquisador, de acordo com a arquitetura que está sendo utilizada. A execução do *workflow* pode ser monitorada pelo pesquisador e deve permitir a intervenção do pesquisador. A intervenção é necessária caso o pesquisador queira avaliar os resultados intermediários para decidir se continua ou não a execução, ou para fazer alguma modificação na definição das próximas atividades do *workflow*.

- Oferecer agendamento da execução do *workflow*. O pesquisador pode desejar executar o *workflow* uma única vez em um determinado dia ou de tempos em tempos, como diariamente ou semanalmente.

- Armazenar metadados sobre os *workflows*. Metadado é comumente definido como

"dado sobre dado", ou ainda, de forma mais completa, como uma informação sobre o dado que permite o acesso e gerenciamento deste dado de maneira eficiente e inteligente. O sistema deve ser capaz de gerar estes metadados automaticamente, na medida do possível, e oferecer mecanismos para o usuário consultá-los e atualizá-los, quando for o caso. Os metadados ajudarão os pesquisadores a definir novos *workflows*, usar resultados gerados em execuções anteriores de *workflows* como entrada de novas execuções de *workflows* e minerar os resultados produzidos por execuções de *workflows* para descobrir informação.

Existem atualmente inúmeros SWCs disponíveis para uso, tais como: Kepler[3], Triana[4] e Taverna[5]

A ferramenta Kepler é um SWC desenvolvido pela Universidade da Califórnia, Berkeley nos EUA, em 2005. Trata-se de uma ferramenta desenvolvida em Java e de código aberto, construída a partir do sistema Ptolemy[6], que é um ambiente para estudos de modelos computacionais, desenvolvido pela mesma instituição.

Outra ferramenta é o sistema Triana, mantido pela Universidade de Cardiff no Reino Unido. Essa ferramenta é de código aberto e desenvolvida utilizando a linguagem Java. Triana permite que seus usuários montem programas a partir de um conjunto de blocos de construçao, que são selecionados para compor um MWC na área de trabalho, para o qual os blocos são arrastados.

O sistema Taverna foi criado pelo projeto myGrid[7], do Reino Unido. Esse sistema foi planejado para trabalhar principalmente com projetos científicos do domínio de bioinformática, e teve sua primeira versão de testes disponibilizada em junho de 2003. O Taverna é um sistema de código aberto e escrito utilizando linguagem Java.

[3]https://kepler-project.org/Wiki.jsp?page=KeplerProvenanceFramework
[4]http://www.trianacode.org/
[5]http://www.taverna.org.uk/
[6]http://ptolemy.eecs.berkeley.edu/ptolemyII/
[7]http://www.mygrid.org.uk/

2.4 Web Services

Web Service é uma tecnologia de computação distribuida, que a indústria de TI está tentando definir para o desenvolvimento de blocos menos rigidos de acoplamento, em sistemas distribuídos, com base em princípios de serviços orientados a arquitetura (Atkinson et al, 2005). *Web Services* interagem trocando mensagens em formato SOAP (*Simple Object Access Protocol*), enquanto a interface para troca de mensagens que é implementada por essas interações é descrita através de WSDL (*Web Services Description Language*) e outros formatos de metadados.

Neste trabalho a utilização de *web services* apoiará a comunicação entre os elementos arquiteturais do sistema, além de permitir o acoplamento de novas ações.

A Figura 2.1 apresenta a estrutura lógica de um web service. Quando uma mensagem SOAP chega em um web service, ela é tratada pela primeira vez pela lógica de processamento de mensagens do serviço, que transforma mensagens SOAP em algo coisa mais tangível para a aplicação (como objetos de um domínio específico). Uma vez que a mensagem foi consumida, seu conteúdo é processado pela lógica da aplicação, fazendo uso de recursos disponíveis para o serviço. Tipicamente, algumas respostas são geradas como um *feedback* através de uma ou mais mensagens (Hey e Fox , 2005).

Ao encapsular os recursos internos dentro do serviço e fornecer uma camada de lógica de aplicação entre esses recursos e os consumidores, os proprietários do serviço são livres para evoluir a sua estrutura interna ao longo do tempo (por exemplo, para melhorar o seu desempenho ou confiabilidade), sem fazer alterações no padrão de troca de mensagens que são usados pelos consumidores de serviço existentes. Isso incentiva o acoplamento flexível entre consumidores e prestadores de serviços, o que é importante em uma computação *inter-enterprise*, já que nenhuma parte está no completo controle de todas as partes de uma aplicação distribuida. No entanto, o acoplamento flexivel não significa que a funcionalidade das aplicações fica comprometida, desde que o conjunto existente de especificações de *Web Services* emergentes, permita construtores (*buiders*) de aplicações distribuidas para interações complexas entre serviços (Hey e Fox , 2005).

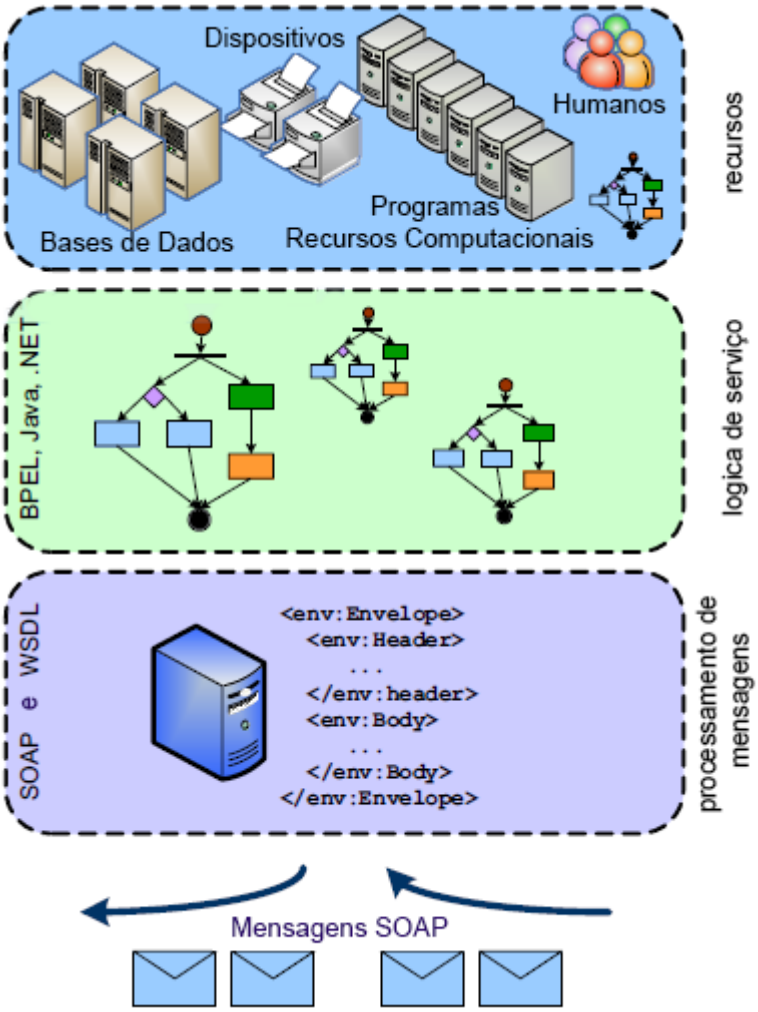

Figura 2.1: Arquitetura de um Web Service (Atkinson et al, 2005)

2.5 Web Services e Grids

Segundo Ludäscher et al (2006), Estados Unidos, Ásia e Europa estão investindo massivamente em projetos de e-science, as siglas chegam a milhões de Dólares e Euros.

Para sustentar toda essa atividade é preciso um conjunto de padrões de serviços de *middleware* que habilitam a coordenação e colaboração de recursos distribuídos. Este conjunto de serviços de *middleware* é denominado *Grid*. Existe toda uma comunidade de cientistas e engenheiros (tanto na academia quanto na indústria), todos se preparando para fazer aplicações científicas e comerciais no contexto de e-Science, uma realidade. Porém existe um problema: o ritmo lento dos processos de padronização de *Web Services*.

Segundo Hey e Fox (2005), existem mais de 60 propostas de padronização de *web services* para e-science, mas ainda não há nenhum prioritário.

Com o avanço das tecnologias de web service neste contexto, arquitetos de *Grids* serão capazes de produzir ferramentas, documentações e materiais educacionais para que a comunidade de *web services* possa construir aplicações sem a necessidade de criar um conjunto paralelo de soluções. Isso permitirá que a comunidade científica se concentre na construção de serviços em alto nível para um determinado domínio de aplicação, enquanto a responsabilidade pela infraestrutura do *Grid* é deixada para a indústria de TI (Hey e Fox , 2005).

Em termos gerais o uso de *web services* em *Grids* e SWC, torna a resolução de problemas e a implementação de novos recursos, mais flexível, uma vez que é possível criar e integar qualquer serviço inexistente.

2.6 Grid Scheduler

Embora muitos tipos de recursos possam ser compartilhados e utilizados em um sistema de *Grid*, usualmente eles são acessados por meio de uma aplicação em execução. Normalmente, uma aplicação é usada para definir uma parte do trabalho em alto nível. Um cenário típico é o seguinte: uma aplicação pode gerar vários jobs, que por sua vez podem ser compostos por subtarefas: o sistema de *Grid* é responsável por enviar cada subtarefa a um recurso para que seja resolvida. Em um simples cenário, é o usuário quem escolhe a máquina mais adequada para executar a sua aplicação ou subtarefas. No entanto, geralmente, os sistemas de *Grid* devem dispor de um escalonador (*scheduler*) que busca de maneira automática e eficiente a máquina mais apropriada para executar o código das tarefas (Xhafa e Abraham, 2010).

Segundo Xhafa e Abraham (2010), problemas relacionados ao escalonamento são bastante estudados nas comunidades de pesquisa. No entanto, quando aplicados a configurações de *Grid*, existem várias características que tornam tais problemas mais complexos do que em sistemas distribuídos convencionais. Dentre elas podemos citar:

- A estrutura dinâmica de um *Grid*. Um *Grid* está sujeito a várias mudanças, seja pelo simples desligamento de um de seus nós, pela perda de conexão de rede ou até pela mudança de sistema operacional. O escalonador deve estar preparado para tais

mudanças.

- A grande heterogeneidade de recursos. Um *Grid* pode ser formado por uma quantidade de nós muito heterogênea, que pode ir desde um laptop, um super computador, a um dispositivo móvel.

- A grande heterogeneidade dos jobs. Enquanto um job pode requerer recursos intensivos, outros podem simplesmente executar uma tarefa atômica.

- A grande heterogeneidade das conexões de rede. Os recursos de um *Grid* podem estar interconectados por diferentes redes, com diferentes latencias e tempos de resposta.

- Existência de políticas locais nos recursos. Companhias proprietárias de determinado recurso computacional podem impor regras sob o mesmo.

- Os requisitos de recursos para um job. Alguns jobs necessitam de determinada combinação de recursos para serem executados.

- A aplicação do *Grid* em larga escala.

- Segurança. Esta não é uma caracteristica presente em um escalonador clássico, porém é importante em um escalonador de *Grid*. Os jobs devem cumprir alguns requisitos de segurança assim como os nós que o executam.

Devido a versatilidade de escalonadores em ambientes de *Grid*, é preciso considerar diferentes modelos computacionais. Esses, permitem formalizar, implementar e avaliar diferentes algoritmos de escalonamento. Alguns dos modelos importantes neste contexto são:

- *Expected time to compute model* (ETC)

  Neste modelo (Ali et al, 2000), presume-se a existência de uma estimativa ou previsão da carga de cada tarefa (por exemplo em milhões de instruções), capacidade de computação de cada recurso (por exemplo em milhões de instruções por segundo), e uma estimativa de prioridade de carga dos recursos. Além disso, uma matriz

de Tempo Esperado de Computação (ETC) de dimensões: número de tarefas por número de máquinas, onde cada posição $ETC[t][m]$ indica o tempo esperado para computar uma tarefa t em um recurso m, também é considerada conhecida e computável neste modelo. No mais simples dos casos, a entrada $ETC[t][m]$ pode ser calculada dividindo-se a carga de trabalho da tarefa t pela capacidade de computação do recurso m. Essa formulação é geralmente viável , uma vez que é possível conhecer a capacidade de computação dos recursos, enquanto a necessidade de computação das tarefas pode ser adiquirida a partir de especificações fornecidas pelo usuário, por histórico de dados ou por previsões (Hotovy, 1996).

- *Total processor cycle comsumption model* (TPCC)

  Apesar de suas propriedades interessantes, o modelo ETC tem uma limitação, a capacidade de computação dos recursos é assumida inalterada durante a computação das tarefas. Esta limitação torna-se mais evidente quando consideramos sistemas de *Grid* em que não só os recursos tem diferentes capacidades de computação, mas também eles podem mudar ao longo do tempo devido a sobrecargas. A velocidade de computação poderia ser assumida constante apenas por períodos curtos tempo. Para remediar essa situação, Fujimoto e Hagihara (2003) introduziram o Consumo Total de Ciclo de Processamento (TPCC). O consumo total de ciclo é definido como o número total de instrições que os recursos do *Grid* podem concluir a partir do início da execução do escalonamento até sua conclusão. Tal como no modelo ETC, a carga de tarefas é expressa em número de instruções; e a capacidade de computação dos recursos é expressa em número de instruções por unidade de tempo. Além disso, o consumo total do poder de computação do *Grid* é medido. Claramente, esse modelo leva em conta que os recursos podem mudar sua velocidade de computação ao longo do tempo, como acontece em sistemas de grande escala cuja carga de trabalho é, em geral, imprevisível.

- *Grid information system model* (GIS)

  Os modelos de computação para escalonadores de *Grid* apresentados permitem uma descrição precisa da instância do problema, no entanto, eles baseiam-se em previsões,

distribuições ou simulações. Atualmente, outros modelos de escalonadores de *Grid* são desenvolvidos a partir de uma perspectiva de alto nível. No modelo *Grid Information System* (GIS) o escalonador usa arquivos de descrição de tarefas e arquivos de descrição de recursos, bem como informações sobre o estado dos recursos (uso de CPU, número de tarefas executadas por recuso), fornecido pelo GIS. O escalonador calcula a melhor correspondência das tarefas aos recursos com base em informações atualizadas da carga de trabalho dos recursos. Este modelo é mais realista para ambientes de *Grid* e é especilmente adequado para a implementação de heuristicas simples como *First-Come First-Served*, *Earlíest Deadline First*, *Shortest job First*, etc. No Capítulo 3, Seções 3.2 e 3.3 serão apresentados dois escalonadores os quais fazem uso do modelo GIS para seu funcionamento.

Apresentados os modelos, vale resaltar que existem critérios de avaliação e otimização utilizados para aferir a produtividade de um sistema de *Grid*. Um dos critérios de otimização mais populares e amplamente estudado é a minimização do *makespan*. *Makespan* é um indicador da produtividade geral do sistema de *Grid*: pequenos valores de *makespan* significam que o escalonador está fornecendo um bom e eficiente planejamento das tarefas aos recursos. Outro critério importante é a otimização do *flowtime*, que se refere ao tempo de resposta das tarefas em execução. Minimizar o valor do *flowtime* implica reduzir o tempo de resposta médio do sistema. Essencialmente, o que se quer é maximizar o rendimento do *Grid* e, ao mesmo tempo, obter um escalonamento que ofereça um QoS aceitável.

Makespan indica o tempo em que a última tarefa termina e flowtime é a soma dos tempos de finalização de todas as tarefas. Formalmente eles podem ser definidos como:

- Minimização do *makespan*: $min_{S_t \in Sched} \{max_{j \in Tasks} F_j\}$
- Minimização do *flowtime*: $min_{S_t \in Sched} \left\{ \sum_{j \in Tasks} F_j \right\}$

Onde F_j é o tempo em que a tarefa j termina sua execução e $Sched$ é o conjunto de todas as possibilidades de escalonamento.

Existem várias abordagens, heurísticas e algoritmos para o escalonamento de tarefas, contudo, neste trabalho serão adotados apenas três deles, que são:

- *Round Robin*

  Round Robin, é um dos algoritmos mais simples de agendamento de processos em um sistema operacional, que atribui frações de tempo para cada processo em partes iguais e de forma circular, manipulando todos os processos sem prioridades (Tanenbaum, 2007). Aplicado ao contexto do escalonamento de tarefas em um *grid*, o algoritmo Round Robin, atribui tarefas de forma cíclica aos nós que compõem o *grid*.

- *First Come First Serve* (FCFS)

  First Come First Serve ou também conhecido como *First In First Out* (FIFO) é um dos escalonadores mais simples e fundamentais (Azmi et al, 2011). Um escalonador FCFS tem a seguinte política: A primeira tarefa a ser escalonada é a primeira que foi inserida na fila. Usando este algoritmo, as tarefas são bufferizadas e enviadas para os nós pela ordem de chegada.

- Abordagem meta-heurística.

  A Busca local é uma família de métodos que exploram um espaço de soluções começando por uma solução inicial, e construindo um caminho para um espaço de soluções durante um processo de busca. Métodos nesta família incluem *Greedy Randomized Adaptive Search Procedure* (GRASP) , *Hill Climbing* (HC), *Simulated Annealing* (SA) e *Tabu Search* (TS), entre outros. Simples métodos de busca local são de interesse pois produzem uma solução viável de certa qualidade dentro de um curto espaço de tempo. Para a realização deste trabalho foi dado um enfoque maior no algoritmo GRASP.

  Segundo Festa e Resende (2002) GRASP é um algoritmo comumente aplicado a problemas de otimização combinatória. Como diversos métodos construtivos, a aplicação do GRASP, consiste em criar uma solução inicial e depois efetuar uma busca local para melhorar a qualidade da solução. O pseudocódigo 1 exemplifica melhor o funcionamento do algoritmo.

Algoritmo 1: Pseudocódigo do algoritmo GRASP

Entrada: α (fator de aleatoriedade)

Saída: S_{best}

```
1  início
2      S_best ← constroiSolucaoAleatoria();
3      enquanto ¬ condicaoDeParada() faça
4          S_candidate ← construtorRandomicoGuloso(α);
5          S_candidate ← buscaLocal(S_candidate);
6          se custo(S_candidate) < custo(S_best) então
7              S_best ← S_candidate;
8          fim se
9      fim enqto
10 fim
```

2.7 SWADE

O SWADE é um sistema de *workflow* científico distribuído, *ad hoc*, que tem como principais características sua facilidade de disponibilizar recursos novos e fazer um amplo uso dos recursos existentes no ambiente em que está inserido. Além dessas, ele permite também uma fácil inserção de um novo nó de processamento no sistema, aumentando o poder de processamento da ferramenta de forma escalável.

Possui uma arquitetura híbrida, onde vários nós de processamento compõem o sistema, executando funcionalidades servidoras e comunicando entre si para execução das tarefas dos *workflows*. Os nós também executam atividades clientes, que permitem a interação do usuário com o sistema, através de funcionalidades como: criar e monitorar a execução de experimentos; inserção e remoção de novos nós no grupo de servidores; entre outros. (Bonifácio et al, 2014).

Além dos recursos de construção de *workflows*, tais como: sequenciamento de ações, iteração, execução condicional, execução paralela, passagem de parâmetros, variáveis e recursão, esse middleware permite que novas ações possam ser disponibilizadas para composição de *workflows* de maneira flexível. Tais ações podem ser de natureza variada, como

por exemplo: um comando de sistema operacional (Windows®ou Linux); um programa executável (Java®ou C); ferramentas de terceiros que possam ser executadas na linha de comando (Matlab®ou Cast3M); serviços web; entre outras, adicionadas sem necessidade de recompilar o sistema (Bonifácio et al, 2014).

Por se tratar de um sistema distribuído, o SWADE apresenta a seus usuários uma abstração quanto a complexidade tecnológica envolvida em seu funcionamento. A arquitetura do SWADE, tal como apresentado na Figura 2.2, trata-se de uma arquitetura descentralizada, onde os nós executam o núcleo funcional do SWADE e estão todos aptos a receberem requisições de usuários e também de outros nós. Como descrito na seção 1.1 a utilização desta aquitetura, da forma como está, acarreta problemas principalmente relacionados a concorrência e desempenho, esses problemas serão abordados na seção 3.1.

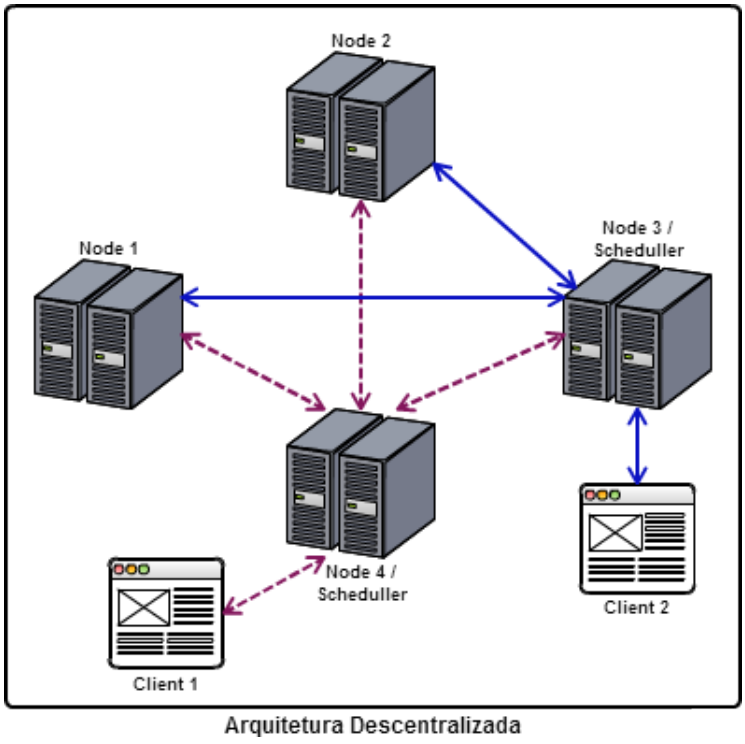

Figura 2.2: Arquitetura descentralizada da rede formada pelo SWADE

Para exemplificar melhor esta aquitetura, um cenário de utilização pode ser observado na Figura 2.3. O cenário começa com um usuário fazendo uma requisição http **(1)** para um nó SWADE. Nesta requisição estão contidas a descrição do WC a ser executado e os arquivos utilizados por esse. Recebida a requisição o nó SWADE começa a interpretar as tarefas descritas no WC e as envia a um escalonador **(2)**, esse por sua vez define

onde aquelas serão executadas, localmente ou externamente, e informa ao interpretador. Caso seja definido que a execução da tarefa será local, o interpretador a envia ao *Core* do nó para que ele a execute **(3)**. Caso seja uma execução externa, o interpretador envia a tarefa ao SwadeService, um web service, do nó definido pelo escalonador, utilizando o protocolo SOAP **(4)**. O SwadeService repassa a tarefa ao *core* do nó em que está alojado **(5)** e ao final de sua execução retorna a resposta ao interpretador. Terminada a execução do *workflow*, o Interpretador retorna o resultado do WC ao usuário **(6)**.

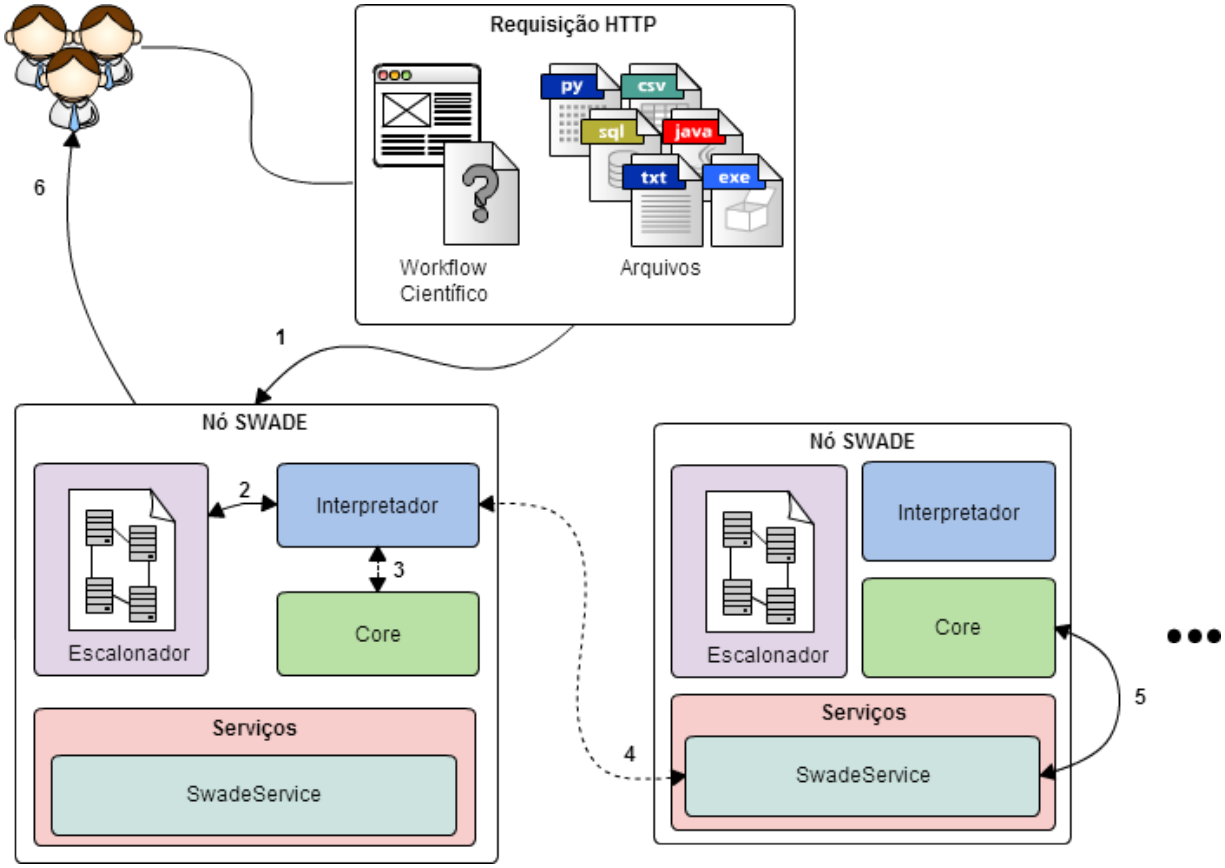

Figura 2.3: Arquitetura detalhada dos nós da rede formada pelo SWADE

2.8 SIGAR API

Para a implementação do escalonador GRASP, realizada neste trabalho e descrita na seção 3.3, a busca por informações referentes ao estado atual dos recursos dos nós da rede é necessária. Para esta obtenção foi utilizada uma biblioteca de código aberto denominada SIGAR. *Hyperic's System Information Gatherer and Reporter* (SIGAR) é uma API independente de plataforma usada para a coleta de dados do sistema operacional. O núcleo

da API está implementado em linguagem C, contudo a API provê interfaces implementadas para a utilização em Java. Com essa biblioteca é possível coletar informações em tempo de execução sobre a infraestrutura em que está sendo executada, independente de arquitetura ou plataforma (Hyperic, 2014).

SIGAR oferece suporte para Linux, FreeBSD, Windows, Solaris, AIX, HP-UX e Mac OSX através de uma variedade de versões e arquiteturas. Utilizando esta API é possível monitorar recursos como:

- Memória do sistema, swap, cpu, carga média, tempo de atividade, logins;
- Memória por processo, cpu, arquivos abertos;
- Detecção e métricas do sistema de arquivos;
- Detecção de interfaces de rede, suas configurações e métricas;

3 Soluções Propostas

Este capítulo apresenta as soluções adotadas para a resolução dos problemas ligados a arquitetura e a melhoria dos métodos de escalonamento propostos para a ferramenta SWADE.

3.1 Arquitetura Centralizada

Como visto na seção 2.7, o sistema SWADE apresenta uma arquitetura descentralizada. Um dos problemas decorrente dessa arquitetura, pode ser observado na Figura 2.2 onde existem dois clientes executando dois WCs distintos. Da forma como está implementada, um nó não toma conhecimento do que está sendo executado nos outros nós da rede. O escalonador de um nó, como visto na Figura 2.3, não toma conhecimento da carga do nó ao qual será enviada uma tarefa, assim esse pode alocar mais tarefas ao nó de destino podendo sobrecarrega-lo.

O escalonador utilizado nesta arquitetura adota um algoritmo de escalonamento baseado no *Round Robin*. A Figura 2.2 será utilizada como exemplo afim de demonstrar o funcionamento desse algoritmo. O *Scheduler* presente no *Node 4* inicia seu funcionamento a partir de uma lista com n tarefas que podem ser alocadas para $m = 4$ nós. A primeira tarefa será escalonada para o *Node 1*, a segunda para o *Node 2*, a terceira para o *Node 3*, a quarta para o *Node 4*, a quinta tarefa será escalonada para o *Node 1* novamente, independentemente se este acabou a execução da tarefa alocada no ciclo anterior. As $n - 5$ demais tarefas serão escalonadas da mesma forma até a lista de tarefas se reduzir a 0.

Em um cenário onde existem vários clientes executando cada um, um *workflow* distinto, existe uma sobrecarga muito grande dos nós da rede, resultando em perda de desempenho e aumento do *Makespan* da execução do WC.

Para a resolução deste problema foi realizada uma mudança na arquitetura do sistema, tornando essa uma arquitetura centralizada. A rigor a arquitetura continua

híbrida, contudo a abordagem de escalonamento passa a ser centralizada. Como pode ser observado na Figura 3.1, existe um escalonador global presente no *Node 4* responsável por escalonar todas as tarefas que necessitem de execução externa, da rede. Dessa forma qualquer novo cliente ou WC inserido na rede terão suas tarefas submetidas ao escalonador central e esse irá eleger qual o nó responsável por executa-las.

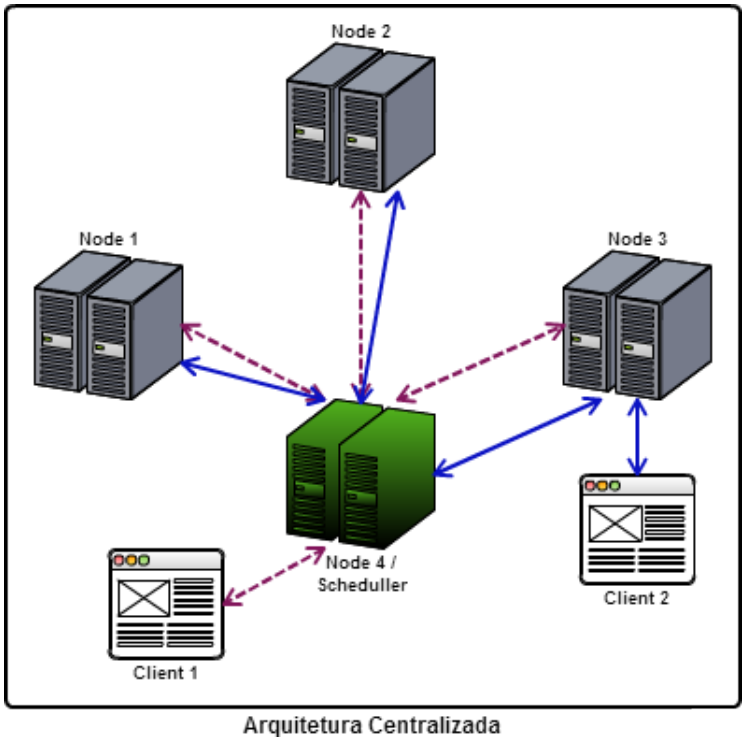

Figura 3.1: Arquitetura centralizada da rede formada pelo SWADE

A Figura 3.2 apresenta a arquitetura centralizada mais detalhadamente. Um cenário de execução utilizando essa nova aquitetura se inicia com um cliente fazendo uma requisição http **(1)**, que contém uma descrição de um WC e arquivos necessários a sua execução, a um nó SWADE. O interpretador do nó, assim que recebida a requisição, começa a interpretar as tarefas descritas no WC e as envia ao escalonador local **(2)**. O escalonador local decide se a tarefa será executada internamente ou externamente. Caso seja decidido que a execução será local, essa decisão é informada ao interpretador e este envia a tarefa ao *Core* para ser executada **(3)**. Caso o escalonador local decida que a tarefa deva ser executada externamente, é feito uma chamada **(4)**, utilizando o protocolo SOAP, a um escalonador global (SwadeServiceSchediling). Esse escalonador tem conhecimento sobre todas as tarefas, de execução externa, ativas na rede. Desta forma é possível

controlar a sobrecarga dos nós. O SwadeServiceScheduler retorna para o escalonador local qual nó executará a tarefa **(4)**, e esse repassa essa informação ao Interpretador **(2)**. O interpretador, utilizando SOAP novamente, envia a tarefa ao SwadeService do nó indicado pelo escalonador **(5)**, que a repassa ao *Core* **(7)** para execução, e também uma mensagem ao SwadeServiceScheduling **(6)**, informando que a tarefa iniciou sua execução. Terminada a execução da tarefa, é retornado seu resultado ao interpretador **(5)** e este por sua vez informa, utilizando SOAP, ao escalonador global **(6)**, o fim da execução da mesma. Desta forma, o escalonador global pode liberar o nó anteriormente escalonado para uma nova alocação.

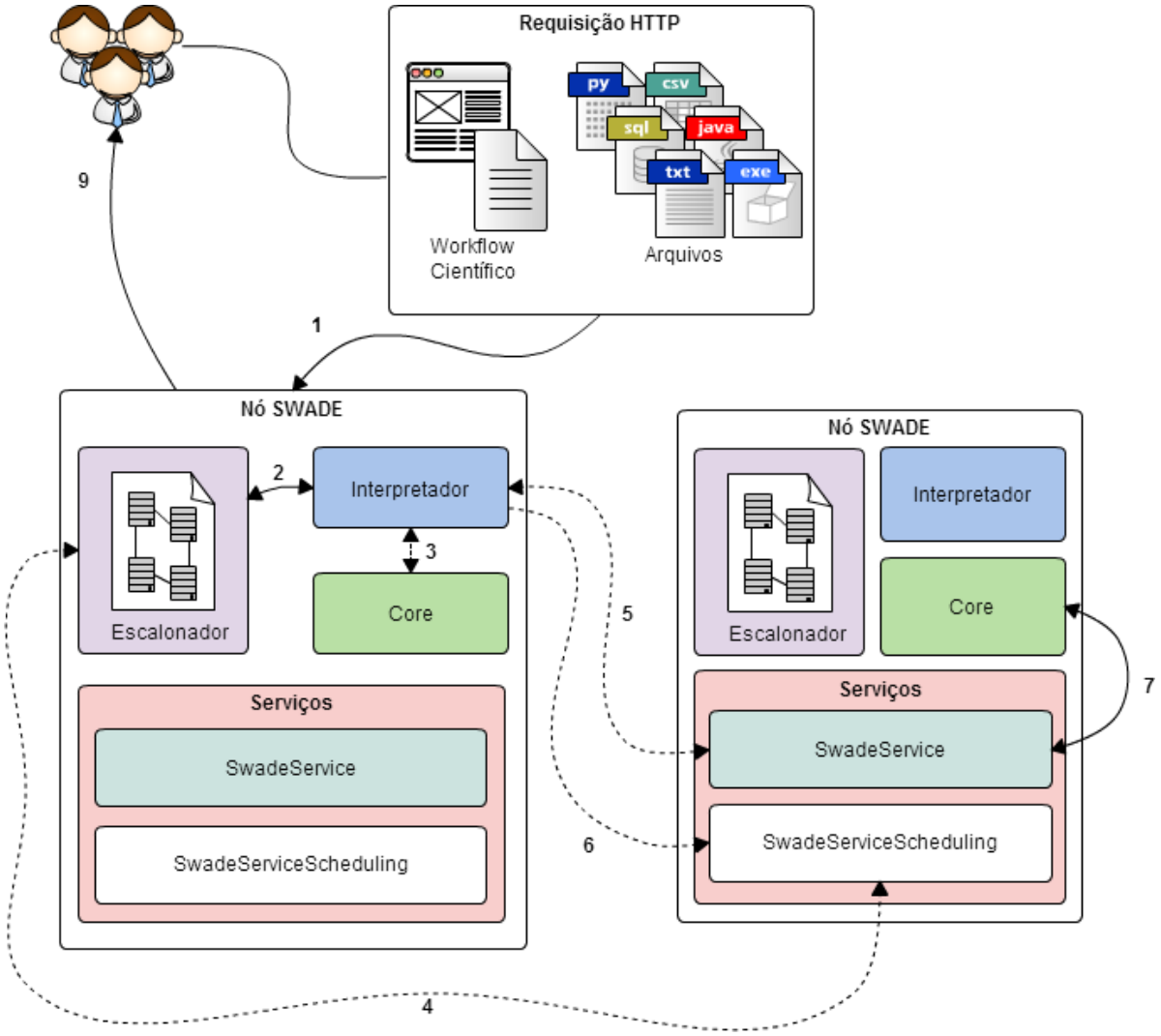

Figura 3.2: Arquitetura centralizada, detalhada dos nós da rede formada pelo SWADE

Esta arquitetura desenvolvida permite a implementação de novas abordagens de escalonamento, uma vez que se tem um controle maior do que está sendo executado na

rede. Além disso, para a implementação de novos escalonadores, basta implementar a interface implementada pelo escalonador global. As próximas seções tratarão de dois novos métodos de escalonamento implementados no sistema SWADE.

3.2 Escalonador FCFS

Como visto na seção 2.6, a política adotada por um escalonador FCFS é: a primeira tarefa a ser escalonada é a primeira que foi inserida na fila. Para o funcionamento de um escalonador FCFS, são necessárias uma lista N de nós de execução e uma lista T de tarefas a serem executadas. A partir da arquitetura já centralizada do sistema SWADE, o escalonador FCFS estaria presente tanto nos escalonadores locais quanto no escalonador global (representado pelo SwadeServiceScheduling na Figura 3.2). A diferença do escalonador local para o global é que o local não tem uma lista de nós.

O funcionamento desse escalonador se dá tal como apresentado no Algoritimo 2. Sua lista de nós é obtida logo quando o sistema SWADE tem início ou quando um nó é removido ou acrescentado a rede. Cada solicitação enviada ao escalonador global inicia uma nova *thread* de escalonamento, que concorrerá com outras solicitações. Assim, é preciso um semáforo no processo de escalonamento. Ao conseguir direito a executar o escalonamento, é preciso verificar se o processo iniciado por solicitações anteriores já não atendeu a solicitação corrente. Após o escalonamento de uma tarefa, é armazenado em uma estrutura qual o nó a executará, e internamente este nó é marcado como ocupado, e nenhuma tarefa é atribuída a ele até que seu estado mude. Ao final da execução dessa tarefa, o interpretador informa ao escalonador global seu término, e esse por sua vez, remove a mesma de sua estrutura, liberando o nó correspondente para alocação.

Algoritmo 2: Pseudocódigo do escalonador FCFS

Entrada: *tarefa*

Saída: *ipDoNoQueExecutaraATarefa*

```
1  início
2      se existeNós() então
3          adicionaÀListaDeTarefas(tarefa);
4          repita
5              se tarefaNãoFoiAlocada(tarefa) então
6                  lock();
7                  //Confirma se não foi alocada por outro processo
8                  se tarefaNãoFoiAlocada(tarefa) então
9                      N ← getNósDesocupados();
10                     T ← getListaDeTarefas();
11                     para cada tarefa t de T faça
12                         se tarefaNãoFoiAlocada(t) então
13                             n ← removePrimeiroNó(N);
14                             alocaTarefaAoNó(t,n);
15                             se tamanho(N)=0 então  break;
16                         fim se
17                     fim para cada
18                 fim se
19                 unlock();
20             senão
21                 alocacaoFoiBemSucedida ← verdadeiro;
22                 ipDoNoQueExecutaraATarefa ← ipDoNóAlocado(tarefa);
23             fim se
24         até alocacaoFoiBemSucedida;
25     senão
26         ipDoNoQueExecutaraATarefa ← ipLocal();
27     fim se
28 fim
```

3.3 Escalonador GRASP

O ambiente de execução do SWADE muitas vezes pode ser heterogêneo, apresentando diferentes configurações nas máquinas que executam os nós. Essa heterogeneidade pode ser prejudicial ao tempo de execução total de um WC, visto que tarefas custosas podem ser atribuídas a nós que não tem tanta capacidade em resolve-las. Pensando nisso, um ambiente ideal de execução seria aquele em que uma tarefa que necessite mais de um determinado recurso fosse executada em um nó que tenha mais desse mesmo recurso.

A ideia por trás do escalonador que utiliza o algoritmo GRASP é justamente fazer o casamento entre tarefas e nós. Como visto na seção 2.6 a aplicação do GRASP, consiste em criar uma solução inicial e depois efetuar buscas locais para melhorar a qualidade da solução.

3.3.1 Algoritmo Grasp

Para o desenvolvimento desse algoritmo, foi definido um conjunto T de tarefas e um conjunto M de máquinas ou nós. Cada máquina M_n apresenta em sua estrutura quatro atributos, são eles memória $M_{n_{memoria}}$, velocidade da cpu (em Mhz) $M_{n_{cpu}}$, quantidade de núcleos $M_{n_{nucleos}}$ e porcentagem de cpu livre $M_{n_{cpuLivre}}$. Esses atributos são classificados com uma pontuação que varia de 0 a 10 e são proporcionais aos dados coletados da máquina. Assim como as máquinas, as tarefas também apresentam estrutura semelhante. Cada tarefa T_n apresenta em sua estrutura os mesmos atributos das máquinas, $T_{n_{memoria}}$, $T_{n_{cpu}}$, $T_{n_{nucleos}}$ e $T_{n_{cpuLivre}}$. A diferença é que para as tarefas, esses atributos indicam o quanto daquele recurso é importante para sua execução. Apresentadas estas estruturas, foi definido um cálculo para representar a afinidade entre uma tarefa e uma máquina, como pode ser visto na Figura 3.3.

$$\begin{aligned} A_{n,m} =& (M_{n_{memoria}} \times T_{m_{memoria}}) + \\ & (M_{n_{nucleos}} \times T_{m_{nucleos}}) + \\ & (M_{n_{cpuLivre}} \times T_{m_{cpuLivre}}) + \\ & (M_{n_{cpu}} \times T_{m_{cpu}}) \end{aligned}$$

Figura 3.3: Cálculo da afinidade entre a máquina n e a tarefa m

A representação da instância do problema contém uma matriz C de tamanho $m \times n$ contendo a afinidade de cada tarefa em cada máquina, como representado na Figura 3.4. A Solução do Algoritmo, por sua vez, é representada por um vetor S, com n posições, que contém, para cada posição $S[i]$ a máquina que executará a tarefa i. Esse vetor S será avaliado constantemente afim de se maximizar a afinidade total da solução.

$$C = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & \cdots & A_{1,n-1} & A_{1,n} \\ A_{2,1} & A_{2,2} & A_{2,3} & \cdots & A_{2,n-1} & A_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ A_{m-1,1} & A_{m-1,2} & A_{m-1,3} & \cdots & A_{m-1,n-1} & A_{m-1,n} \\ A_{m,1} & A_{m,2} & A_{m,3} & \cdots & A_{m,n-1} & A_{m,n} \end{bmatrix}$$

Figura 3.4: Representação de uma instância com m tarefas e n máquinas

Para a contrução de uma solução inicial para o problema são realizados os seguintes passos:

- A partir da matriz C, é construído um vetor V, com a média das afinidades de cada tarefa e este é ordenado em ordem decrescente.
- Além disso, é feita uma ordenação decrescente no vetor de afinidades acumuladas das máquinas.
- É selecionada a tarefa de maior afinidade do vetor de médias e é colocada em uma das máquinas de maior afinidade acumulada. Para a escolha da máquina é selecionada uma porcentagem das máquinas com maior afinidade acumulada e dentre essas é escolhida uma aleatoriamente.
- Esse procedimento é repetido até que cada tarefa tenha sido alocada em uma das máquinas.

Para a busca local, foi considerado apenas um tipo de movimentação na solução afim de gerar uma vizinhança distinta. Para tal, escolhe-se uma tarefa no vetor S solução, e troca a máquina que a realiza por outra. O pseudocódigo 3 exemplifica melhor o funcionamento do algoritmo GRASP desenvolvido.

Algoritmo 3: Pseudocódigo do algoritmo GRASP implementado no SWADE

Entrada: $C_{Instancia}$, α(fator de aleatoriedade), *epocas*
Saída: S_{best}

```
1  início
2  |  i = 0;
3  |  enquanto i < epocas faça
4  |  |  S_candidate ← construtivo(C_Instancia,α);
5  |  |  S_candidate ← buscaLocal(S_candidate);
6  |  |  se avaliacao(S_candidate) > avaliacao(S_best) então
7  |  |  |  S_best ← S_candidate;
8  |  |  fim se
9  |  |  i++;
10 |  fim enqto
11 fim
```

3.3.2 Implementação no SWADE

Para a implementação do escalonador GRASP no SWADE, foi necessário o estudo de métodos para obtenção de informações referentes a infraestrutura da máquina onde um nó executa. Para esta obtenção foi utilizado a biblioteca SIGAR API descrita na seção 2.8. Essa biblioteca permite a obtenção dos parâmetros de máquina descritos na seção anterior: $M_{n_{memoria}}$, $M_{n_{cpu}}$, $M_{n_{nucleos}}$ e $M_{n_{cpuLivre}}$. Afim de que essas informações pudessem ser acessadas pelo escalonador, foi necessário também, realizar uma pequena modificação na arquitetura do sistema, adicionando mais um serviço aos nós, o SwadeServiceMachine. Esse serviço fornece de tempos em tempos os atributos mencionados, dessa forma os pesos dos nós sempre refletiriam sua situação atual. A Figura 3.5 exemplifica como ficou a nova arquitetura.

Na seção anterior foram descritos também os parâmetros de tarefa: $T_{n_{memoria}}$, $T_{n_{cpu}}$, $T_{n_{nucleos}}$ e $T_{n_{cpuLivre}}$. Esses parâmetros são inteiros e também variam de 0 a 10, contudo, para uma mesma tarefa a soma dos parâmetros não pode ultrapassar 10 pontos. Para a definição desses, foi criado um arquivo XML onde são descritos os pesos[8], ou parâmetros, para cada tipo de tarefa como no código 3.1.

O funcionamento do escalonador GRASP se dá tal como descrito no algoritmo 4. Um detalhe importante a ser observado é que como um nó não suporta mais de uma tarefa sendo executada por vez, foi necessária uma adaptação ao escalonador grasp, para

[8]Os pesos estão descritos na ordem ghz, núcleos, memória e processador

caso uma tarefa fosse atribuída a um nó já ocupado, o nó que iria executar esta tarefa seria o próximo nó disponível, de uma lista de nós livres.

```
<?xml version="1.0" encoding="UTF-8"?>
<Heights>
    <Height Action="a">0;9;0;1</Height>
    <Height Action="b">6;3;0;1</Height>
    <Height Action="c">4;0;5;1</Height>
    <Height Action="d">0;9;0;1</Height>
    <Height Action="debug">4;0;2;4</Height>
    <Height Action="dec">1;0;9;0</Height>
    <Height Action="div">1;1;6;2</Height>
    <Height Action="divspecial">0;5;1;4</Height>
    <Height Action="e">1;1;8;0</Height>
    <Height Action="elapsedtime">6;0;1;3</Height>
    <Height Action="error">5;1;0;4</Height>
    <Height Action="getdate">3;0;7;0</Height>
    <Height Action="geterraddress">6;2;0;2</Height>
    <Height Action="geterror">5;2;1;2</Height>
    <Height Action="getlasterror">4;0;0;6</Height>
    <Height Action="getoutaddress">0;1;0;9</Height>
    <Height Action="getresource">7;0;3;0</Height>
    <Height Action="getschedulerserver">7;1;0;2</Height>
    <Height Action="getstack">0;1;8;1</Height>
    <Height Action="greater">0;8;0;2</Height>
    <Height Action="greatere">5;1;2;2</Height>
    <Height Action="inc">3;1;2;4</Height>
    <Height Action="less">1;8;1;0</Height>
    <Height Action="lesse">4;3;3;0</Height>
    <Height Action="mod">5;3;1;1</Height>
    <Height Action="mult">1;1;0;8</Height>
    <Height Action="plainnumber">2;1;0;7</Height>
    <Height Action="pop">9;1;0;0</Height>
    <Height Action="print">0;2;6;2</Height>
    <Height Action="println">5;0;4;1</Height>
    <Height Action="push">3;4;1;2</Height>
    <Height Action="rand">1;0;4;5</Height>
    <Height Action="reseterror">1;3;5;1</Height>
    <Height Action="savefile">4;0;0;6</Height>
    <Height Action="savelocalfile">0;3;1;6</Height>
    <Height Action="script">4;2;1;3</Height>
    <Height Action="scriptfile">1;0;7;2</Height>
    <Height Action="sendmail">5;0;0;5</Height>
    <Height Action="setresource">4;1;5;0</Height>
    <Height Action="setschedulerserver">6;1;1;2</Height>
    <Height Action="setstack">3;0;5;2</Height>
    <Height Action="shift">5;3;0;2</Height>
    <Height Action="simds">0;1;6;3</Height>
    <Height Action="sleep">1;9;0;0</Height>
    <Height Action="starttime">0;2;3;5</Height>
    <Height Action="unshift">0;4;5;1</Height>
    <Height Action="waitbackground">0;3;7;0</Height>
</Heights>
```

Código 3.1: Arquivo XML que descreve tarefas e respectivos pesos.

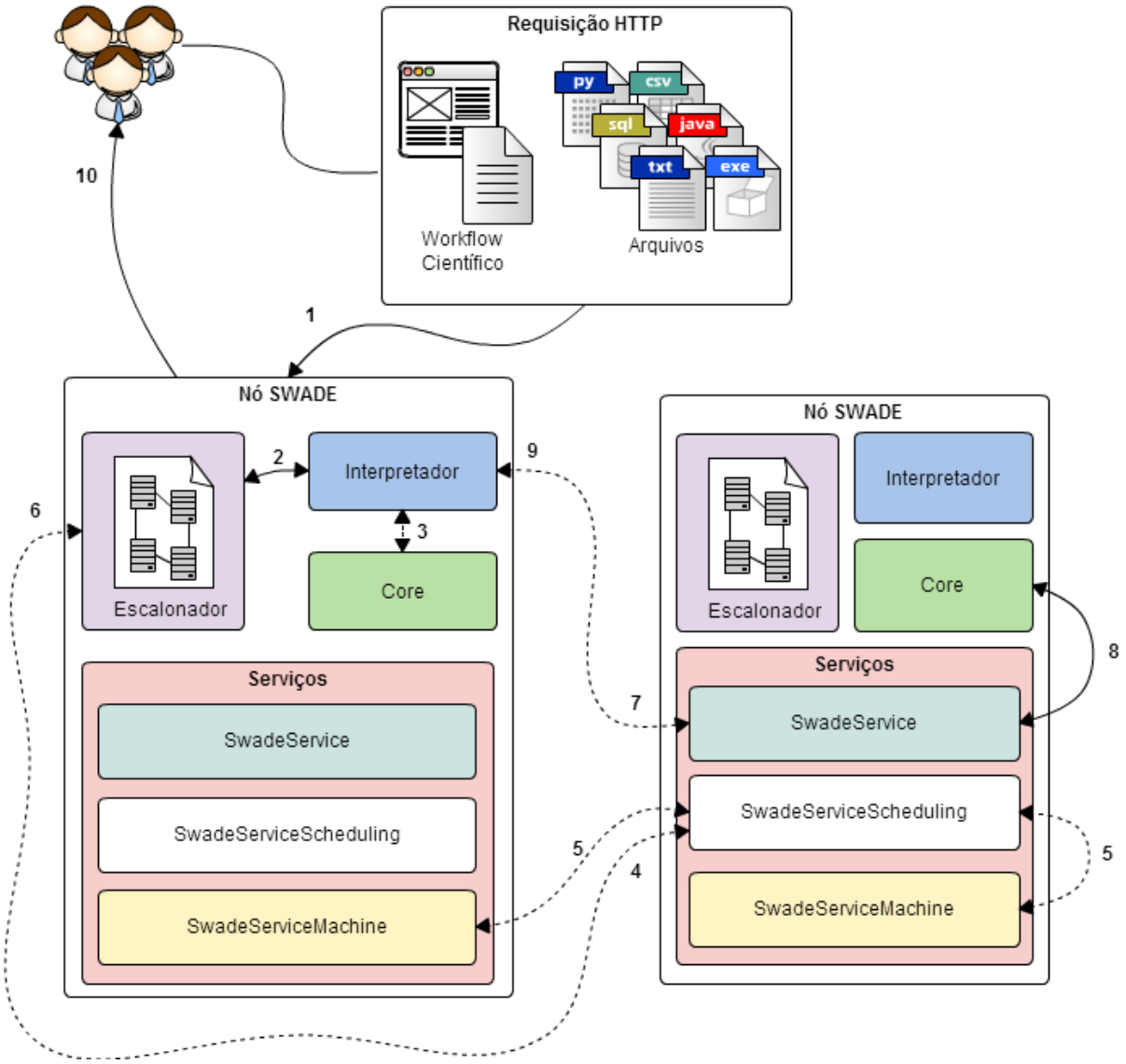

Figura 3.5: Arquitetura detalhada dos nós da rede formada pelo SWADE utilizando o escalonador GRASP

Algoritmo 4: Pseudocódigo do escalonador GRASP

```
    Entrada: tarefa
    Saída: ipDoNoQueExecutaraATarefa
 1  início
 2      se existeNós() então
 3          tarefaComPesos ← carregaPessosDaTarefa(tarefa);
 4          adicionaÀListaDeTarefas(tarefaComPesos);
 5          repita
 6              se tarefaNãoFoiAlocada(tarefa) então
 7                  lock();
 8                  //Confirma se não foi alocada por outro processo
 9                  se tarefaNãoFoiAlocada(tarefa) então
10                      N ← getNósDesocupadosComPesos();
11                      T ← getListaDeTarefasComPesos();
12                      A ← getMatrizDeAfinidade(N,T);
13                      S ← GRASP(A, α, numIteracoes);
14                      para i ←0 até tamanho(S) faça
15                          se nóEstáDesocupado(S[i]) então
16                              alocaTarefaAoNó(T[i],S[i]);
17                          senão
18                              N_d ← getNósDesocupados();
19                              n ← removePrimeiroNó(N_d);
20                              alocaTarefaAoNó(T[i],n);
21                          fim se
22                      fim para
23                  fim se
24                  unlock();
25              senão
26                  alocacaoFoiBemSucedida ← verdadeiro;
27                  ipDoNoQueExecutaraATarefa ← ipDoNóAlocado(tarefa);
28              fim se
29          até alocacaoFoiBemSucedida;
30      senão
31          ipDoNoQueExecutaraATarefa ← ipLocal();
32      fim se
33  fim
```

4 Avaliação da Solução Proposta

Com o objetivo de avaliar as abordagens de escalonamento implementadas, foram realizados testes observando o *makespan*, ou tempo de execução total, de um *workflow* científico quando executado utilizando cada um dos escalonadores.

O *workflow* definido, código 4.1, executa o sistema SimDS, trata-se de um simulador de dinâmica de sistemas, que recebe como parâmetros arquivos que descrevem o sistema de equações necessários a sua execução, uma constante δ de variação de tempo para cada passo de execução, um número de iterações e o método de simulação, Euler ou Runge-Kutta de quarta ordem. O WC executa o sistema SimDS 500 vezes passando 100 iterações como parâmetro, dessa forma é possível obter uma execução robusta com tarefas em uma granularidade adequada, permitindo uma melhor diferenciação do *makespan*.

```
1  main(){
2      em = "em_pp.txt" ;
3      um = "um_pp.txt" ;
4      limit = "500";
5      count = "0" ;
6      while (count < limit) {
7          count = inc(count, "1") ;
8          x=background("file=print(\"simds-\",count,\".txt\");
9                        rr=simds(em, um, file , \"0.01\", \"100\", \"1\");
10                       xx=appendResource(\"SimDS\", \"SimDS file \", count,
                             \" = \", rr, \"\n\");",
11                       "em", em,
12                       "um", um,
13                       "count", count) ;
14     } ;
15     waitBackground() ;
16     rr = getResource("SimDS");
17     println(rr);
18 }
```

Código 4.1: *Workflow* científico utilizado para casos de testes

Foram realizados três casos de teste, um teste com execução local, Figura 4.1, um teste utilizando cinco máquinas e escalonamento FCFS, Figura 4.2, e um teste utilizando cinco máquinas e escalonamento GRASP, Figura 4.3.

No teste local, o *makespan* da execução do *workflow* foi de aproximadamente 41 minutos. Para o teste com 5 máquinas e escalonamento FCFS, o *makespan* foi de aproxi-

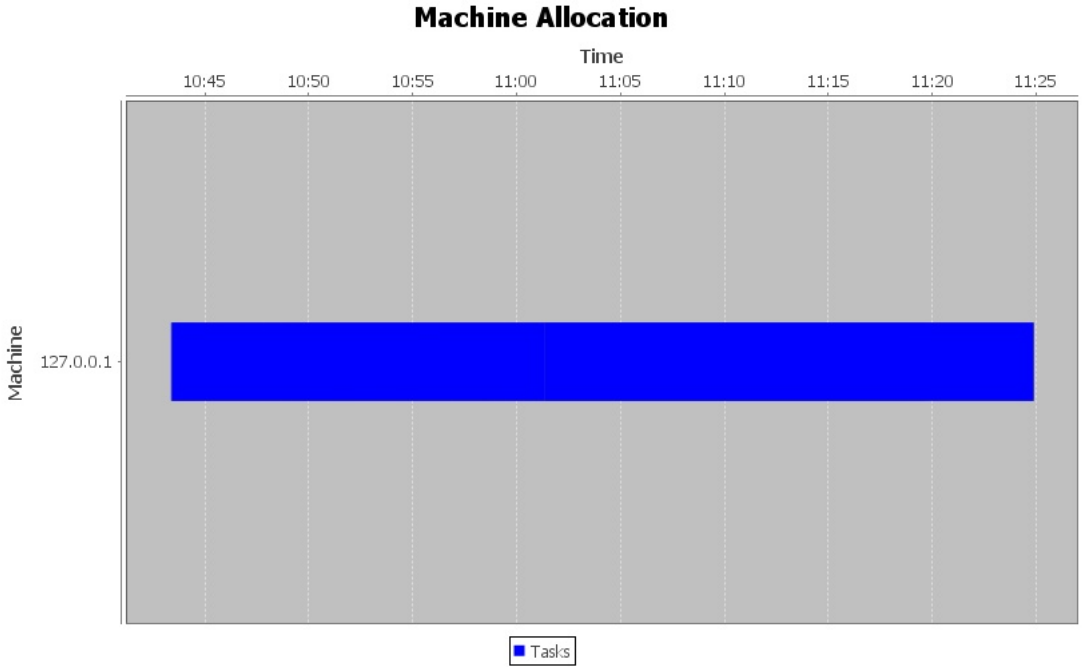

Figura 4.1: Execução de WC localmente

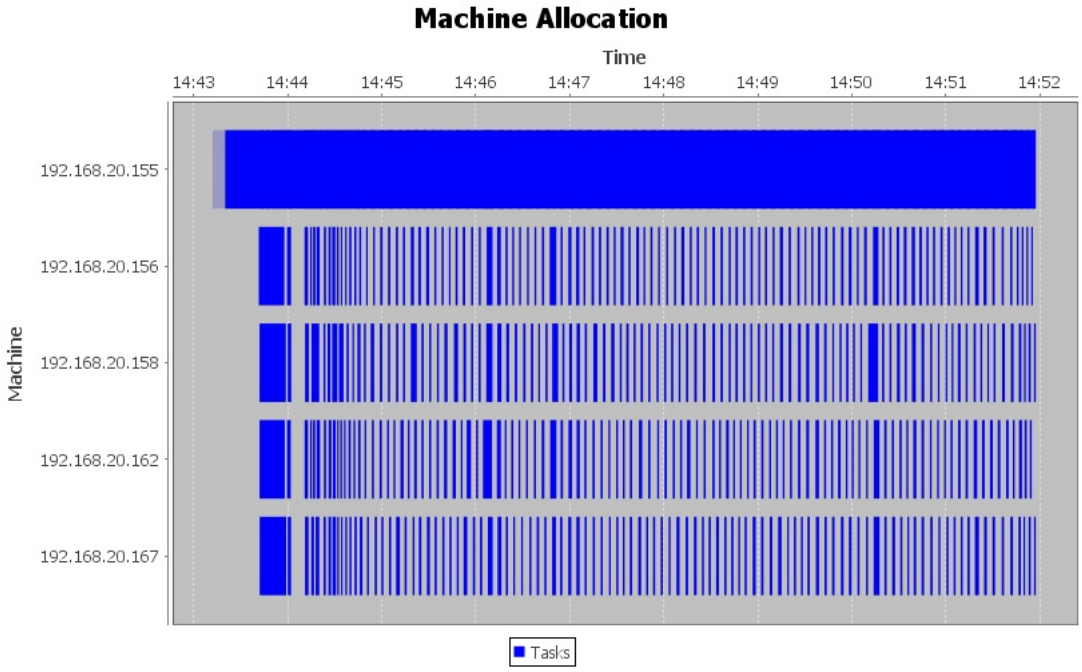

Figura 4.2: Execução de WC utilizando 5 máquinas e escalonamento FCFS

madamente 9 minutos, enquanto que para o teste com escalonamento GRASP o *makespan* foi de aproximadamente 5 minutos. É notável o ganho de desempenho entre a execução local e a paralela. Quando comparado o resultado da execução com escalonamento FCFS, com o resultado da execução com escalonador GRASP, é possível observar uma redução de aproximadamente 40% do tempo de execução total do WC. Esses resultados vão ao

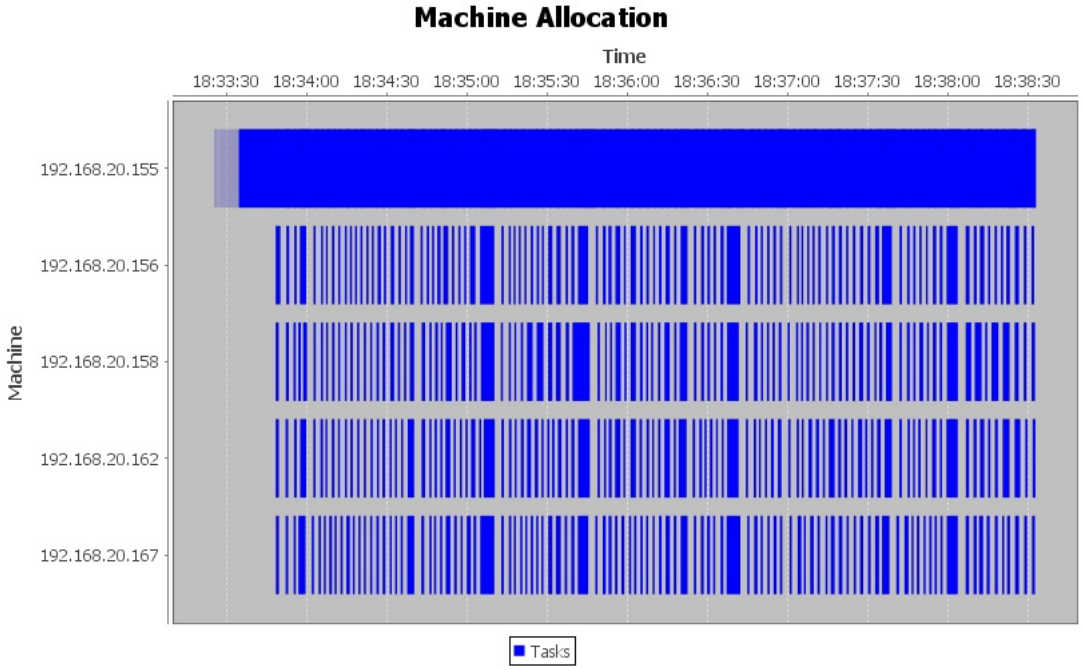

Figura 4.3: Execução de WC utilizando 5 máquinas e escalonamento GRASP

encontro dos resultados esperados, dando evidências que os escalonadores implementados proporcionam ganhos efetivos a execuçãode WCs.

5 Trabalhos Futuros

Durante a realização deste trabalho foram observados alguns pontos que fogem ao escopo desse ou que não foram implementados devido ao tempo, no entanto, vêm a contribuir com a evolução do sistema SWADE. A seguir estão listados os pontos identificados:

- Atualmente, existem padrões de *web services* utilizados em SWCs, um trabalho futuro seria adicionar tais padrões ao sistema SWADE.
- Existem outros tipos de buscas locais utilizando o algoritmo GRASP, estas poderiam ser avaliadas separadamente.
- Os escalonadores implementados não permitem a execução de mais de uma tarefa por nó, para um trabalho futuro esse controle poderia ser implementado.
- O escalonador GRASP, utiliza pesos nas tarefas e nos nós para efetuar suas escolhas de escalonamento. Os pesos dos nós são obtidos de acordo com as configurações das máquinas que os executam, já os pesos das tarefas são definidos por quem executa os WCs. Como trabalho futuro, um estudo poderia ser feito sobre qual a melhor configuração de pesos para cada tipo de tarefa.

6 Conclusão

Muitos trabalhos de pesquisas nas mais diversas áreas fazem uso de modelos computacionais e sistemas de *workflow* ciêntificos para seu desenvolvimento. O uso destes sistemas permitem aos pesquisadores a criação e execução de experimentos dado um conjunto de serviços e recursos disponíveis em um determinado ambiente.

Neste trabalho foi apresentado o sistema de *workflow* científico SWADE. Este sistema possuia uma arquitetura descentralizada onde vários nós de precessamento o compunham comunicando entre si para a execução das tarefas de um WC. Essa arquitetura é composta por vários elementos, contudo, esse trabalho teve como foco um elemento em específico, o escalonador de tarefas. O escalonador de tarefas inicialmente apresentado utilizava uma abordagem de escalonamento baseada no algoritmo *Round Robin*, esse escalonador gerava um desperdício de recursos.

Este trabalho propôs uma modificação na arquitetura do sistema SWADE afim de que o escalonador se tornasse centralizado, deste modo, todos os nós da rede formada pelo SWADE utilizariam um mesmo escalonador. Esperava-se com esta modificação um maior controle da execução das tarefas no sistema. Através dos testes realizados e implementações de novas heuristicas de escalonamento, foi possível constatar que esse controle foi alcançado, assim como as modificações arquiteturais.

Outra proposta deste trabalho foi a implementação de novas formas de escalonamento afim de diminuir a perda de recursos e o *makespan* da execução de WCs. Foram implementadas duas abordagens de escalonamento, a FCFS e a abordagem utilizando algoritmo GRASP. Após a implementação, ambas abordagens foram submetidas a testes, e o resultado desses foram ao encontro dos resultados esperados. Tanto o escalonador FCFS quanto o escalonador GRASP apresentaram uma melhora significativa na redução do tempo de execução de WCs. Um resultado que merece destaque é o escalonador GRASP que, segundo os testes realizados, quando comparado ao escalonador FCSF reduziu em 40% o *mekespan* da execução de um WC de prova.

Os resultados alcançados são promissores, contudo uma reflexão do ponto de vista

da escalabilidade deve ser feita. É possível que com o aumento da quantidade de nós do sistema a execução do algorítimo GRASP passe a representar um atrazo ao tempo de execução total de um WC. Além disso, com um volume grande de nós, o tráfego de rede para um escalonador centralizado deve ser considerado e tratado.

Como mencionado no capítulo 5, existem vários pontos que ainda podem ser explorados e evoluidos no sistema SWADE e portanto podem ser considerados trabalhos futuros.

Referências Bibliográficas

Ali, S.; Siegel, H.; Maheswaran, M. ; Hensgen, D. **Task execution time modeling for heterogeneous computing systems**. p. 185 – 199, 2000.

Altintas, I.; Barney, O. ; Jaeger-Frank, E. **Provenance collection support in the kepler scientific workflow system**. In: Moreau, L.; Foster, I., editors, Provenance and Annotation of Data, volume 4145, p. 118–132, San Diego Supercomputer Center, Universiy of California, San Diego, 9500 Gilman Drive, CA 92092-0505, 2006. Springer Berlin Heidelberg.

Atkinson, M.; DeRoure, D.; Dunlop, A.; Fox, G.; Henderson, P.; Hey, T.; Paton, N.; Newhouse, S.; Parastatidis, S.; Trefethen, A. ; others. **Web service grids: an evolutionary approach**. volume 17, p. 377–389. Wiley Online Library, 2005.

Azmi, Z.; Bakar, K. ; Shamsir, M. **Scheduling grid jobs using priority rule algorithms and gap filling techniques.** volume 37, p. 61–76, 2011.

Bonifácio, A. L. **Análise de ferramentas computadorizadas para suporte à modelagem computacional - estudo de caso no domínio de dinâmica dos corpos deformáveis**. In: Dissertação (Mestrado em Modelagem Computacional) - Universidade Federal de Juiz de Fora, Juiz de Fora, MG, Brazil, 2008.

Bonifácio, A. L.; Amaral, R. O.; Rooke, F.; Barbosa, C. B. ; Farage, M. C. **Implementação de uma heurística para ajuste de parâmetros de um modelo de mecânica computacional usando um sistema de workflow científico distribuído**. In: XI Simposio de Mecânica Computacional - II Encontro Mineiro de Modelagem Computacional, Juiz de Fora, MG, Brazil, 2014.

Festa, P.; Resende, M. G. **Grasp: An annotated bibliography**. In: Essays and surveys in metaheuristics, p. 325–367. Springer, 2002.

Fujimoto, N.; Hagihara, K. **Near-optimal dynamic task scheduling of precedence constrained coarse-grained tasks onto a computational grid**. In: in Proc. of ISPDC 2003, p. 80–87, 2003.

Hey, T.; Fox, G. **Special issue: Grids and web services for e-science**. volume 17, p. 317–322. John Wiley and Sons, Ltd., 2005.

Hollingsworth, D.; Hampshire, U. **Workflow management coalition the workflow reference model**. p. 68. Citeseer, 1993.

Hotovy, S. **Workload evolution on the Cornell Theory Center IBM SP2**. In: Feitelson, D. G.; Rudolph, L., editors, Job Scheduling Strategies for Parallel Processing, p. 27–40. Springer-Verlag, 1996. Lect. Notes Comput. Sci. vol. 1162.

Hyperic. **Sigar api (system information gather and reporter)**. In: http://www.hyperic.com/products/sigar, acesso em jun 2014.

Lemos, M. **Workflow para bioinformática**. In: Tese (Doutorado em Informática) – Programa de Pós-Graduação em Informática, Pontíficia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ, Brazil, 2004.

Ludäscher, B.; Altintas, I.; Berkley, C.; Higgins, D.; Jaeger, E.; Jones, M.; Lee, E. A.; Tao, J. ; Zhao, Y. **Scientific workflow management and the kepler system**. volume 18, p. 1039–1065. John Wiley and Sons, Ltd., 2006.

Parastatidis, S. **A platform for all that we know: creating a knowledge-driven research infrastructure.** In: Hey, T.; Tansley, S. ; Tolle, K. M., editors, The Fourth Paradigm, p. 165–172. Microsoft Research, 2009.

Pignotti, E.; Edwards, P.; Gotts, N. ; Polhill, G. **Enhancing workflow with a semantic description of scientific intent**. volume 9, p. 222 – 244, 2011.

Seffino, L. A.; Medeiros, C. B.; Rocha, J. V. ; Yi, B. **woodss — a spatial decision support system based on workflows**. volume 27, p. 105 – 123, 1999.

Tanenbaum, A. S. **Scheduling in interative sytems**. In: Modern operating systems 3rd, p. 93–96. Prentice Hall Press Upper Saddle River, NJ, USA, 2007.

Taylor, I. J.; Deelman, E. ; Gannon, D. B. **Workflows for e-science scientific workflows for grids**. Springer-Verlag New York, 2006.

Xhafa, F.; Abraham, A. **Computational models and heuristic methods for grid scheduling problems**. volume 26, p. 608–621. Elsevier, 2010.

Yang, X.; Bruin, R. P. ; Dove, M. T. **Developing an end-to-end scientific workflow**. volume 12, p. 52–61. IEEE, 2010.