UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

CAMADA DE COMUNICAÇÃO PARA UM SISTEMA GERÊNCIA DE GRADES COMPUTACIONAIS

TRABALHO DE GRADUAÇÃO

Rodrigo Exterckötter Tjäder

Santa Maria, RS, Brasil

2010

CAMADA DE COMUNICAÇÃO PARA UM SISTEMA GERÊNCIA DE GRADES COMPUTACIONAIS

por

Rodrigo Exterckötter Tjäder

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Prof. Benhur de Oliveira Stein

Trabalho de Graduação N. 294

Santa Maria, RS, Brasil

2010

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

CAMADA DE COMUNICAÇÃO PARA UM SISTEMA GERÊNCIA DE GRADES COMPUTACIONAIS

elaborado por
Rodrigo Exterckötter Tjäder

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Prof. Benhur de Oliveira Stein
(Presidente/Orientador)

Profª Andrea Schwertner Charão (UFSM)

Prof. Raul Ceretta Nunes (UFSM)

Santa Maria, 07 de janeiro de 2010.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

CAMADA DE COMUNICAÇÃO PARA UM SISTEMA GERÊNCIA DE GRADES COMPUTACIONAIS
Autor: Rodrigo Exterckötter Tjäder
Orientador: Prof. Benhur de Oliveira Stein
Local e data da defesa: Santa Maria, 07 de janeiro de 2010.

Vários problemas existentes atualmente requerem uma grande quantidade de poder computacional para serem resolvidos. Usando grades computacionais, entidades distintas podem cooperar para a resolução desses problemas. Para auxiliar nessa cooperação, existem ferramentas para gerenciar grades computacionais. No entanto, grades computacionais são inerentemente dinâmicas, com os nós que as compõem mudando regularmente, e heterogêneas, compostas por nós com capacidades distintas. Nesse contexto, está sendo desenvolvido no LSC (Laboratório de Sistemas de Computação da UFSM) um sistema didático de gerência dinâmica de trabalhos em grades computacionais com suporte a grades heterogêneas, permitindo uma reorganização do trabalho durante sua execução conforme necessário de acordo com fatores como carga das máquinas e entrada e saída de nós na grade, visando assim manter a carga de trabalhos bem distribuída, evitando sobrecarga em uma parte da grade e ociosidade em outra. O presente trabalho propõe o desenvolvimento de uma ferramenta que ofereça uma camada base de comunicação a ser utilizada na implementação de tal sistema de gerência de grades computacionais.

Palavras-chave: Grades computacionais; comunicação.

ABSTRACT

Trabalho de Graduação
Undergraduate Program in Computer Science
Universidade Federal de Santa Maria

COMMUNICATION LAYER FOR A COMPUTING GRID MANAGEMENT SYSTEM

Author: Rodrigo Exterckötter Tjäder
Advisor: Prof. Benhur de Oliveira Stein

Several problems require a large amount of computing power in order to be solved. Using grid computing, separate entities can cooperate to solve these problems. To help that cooperation there are middlewares that manage computational grids. However, a computational grid is inherently dynamic, as its nodes change continuously, and heterogeneous, composed by nodes with different abilities. In this context, a dynamic job management system for heterogeneous computational grids is being developed at LSC, allowing a rescheduling of the jobs during their execution, as needed according to factors such as the current load of the machines, and the joining and leaving of nodes in the grid, therefore keeping the workload well spread across the grid, avoiding overloading some nodes while others are idle. This work proposes the creation of a base communication layer to be used in the implementation of such a computational grid management system.

Keywords: Grid computing, communication.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

ASCII	American Standard Code for Information Interchange
BOINC	Berkeley Open Infrastructure for Network Computing
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
LSC	Laboratório de Sistemas de Computação
MPI	Message Passing Interface
RMI	Remote Method Invocation
SOAP	Simple Object Access Protocol
URI	Universal Resource Identifier

SUMÁRIO

1 INTRODUÇÃO

A computação em grade consiste geralmente na combinação de vários recursos computacionais independentes para a resolução de um problema cuja solução seria impraticável com um número pequeno de máquinas. A utilização de computação em grade permite um compartilhamento de recursos computacionais entre várias entidades, fazendo com que elas tenham ao seu dispor um poder computacional muito maior do que poderiam ter acesso usando seus recursos individualmente.

Sendo amplamente utilizada com fins científicos, é importante a existência de sistemas que gerenciem tais grades de recursos computacionais e os trabalhos a serem executados de forma eficiente. No entanto, esses sistemas são geralmente complexos, sendo difícil sua utilização como ferramenta didática.

Por outro lado, um ambiente de grade é intrinsecamente dinâmico, com o conjunto de nós da grade sendo continuamente alterado, e heterogêneo, sendo composto por nós que possuem capacidades diferentes.

A larga utilização de computação em grade torna necessário um melhor aproveitamento dos recursos compartilhados. Este trabalho pretende melhorar a utilização de tais recursos através de uma distribuição dinâmica de trabalhos nas máquinas, de forma a evitar sobrecarga de uma parte da grade e ociosidade em outra.

Para resolver esse problema, está sendo desenvolvido no LSC (Laboratório de Sistemas de Computação da UFSM) um sistema dinâmico de gerência de trabalhos em grades computacionais, que permita uma reorganização do trabalho após seu início conforme necessário de acordo com fatores como carga das máquinas e entrada e saída de nós na grade. A base para essa dinamicidade está no suporte a trabalhos maleáveis, que podem ser quebrados e redistribuídos. Esse sistema contará também com suporte a ambientes heterogêneos visando, por exemplo, suportar nós de cálculos utilizando computação em

placas gráficas.

Este trabalho se insere nesse projeto, e tem como objetivo desenvolver um sistema de comunicação para um gerenciador de trabalhos maleáveis em grades computacionais, que possa dividir e distribuir o trabalho da forma mais apropriada conforme o estado atual da grade. Esse sistema de comunicação deve permitir uma implementação dos componentes independente dos detalhes da comunicação, e uma fácil portabilidade entre mecanismos de comunicação e linguagens de programação diferentes.

Este trabalho está estruturado da seguinte forma: o capítulo 2 faz uma revisão sobre conceitos e tecnologias úteis para o entendimento do trabalho em geral; o capítulo 3 descreve o sistema de gerência no qual este trabalho está inserido, assim como o sistema de comunicação desenvolvido como objetivo deste trabalho e o capítulo 4 realiza a avaliação dos resultados. Por fim, o capítulo 5 conclui o trabalho, discutindo os resultados do que foi desenvolvido.

2 REVISÃO BIBLIOGRÁFICA

Atualmente vários problemas requerem uma quantidade muito grande de poder computacional para serem resolvidos. A simulação de problemas complexos é necessária para vários campos de pesquisa, tornando necessário o uso de processamento distribuído para resolver tais problemas em tempo hábil.

Uma das formas de se resolver esses problemas é usando programação paralela, como, por exemplo, possibilitado pela biblioteca MPI (Message Passing Interface), que fornece passagem de mensagens entre processadores ou máquinas distintas (SNIR et al., 1995).

Conforme pesquisas conjuntas foram sendo realizadas entre grupos separados geograficamente foi surgindo uma necessidade do compartilhamento dos recursos para a resolução de problemas complexos.

A computação em grade é uma forma de se obter o poder computacional para resolver esses problemas, consistindo da utilização de recursos computacionais de entidades distintas e geograficamente separadas para a realização de uma tarefa comum (FOSTER; KESSELMAN, 1999).

A grande heterogeneidade das máquinas e dos ambientes administrativos dos componentes de uma grade computacional inviabiliza soluções tradicionais de programação paralela como o MPI, que foram desenvolvidos para uso em ambientes mais homogêneos.

Para facilitar o uso de grades computacionais foram criados vários projetos visando auxiliar na manutenção das grades, como, por exemplo, o Globus Toolkit (FOSTER; KESSELMAN, 1997), desenvolvido pela Globus Alliance, e o BOINC (ANDERSON, 2004), desenvolvido na Universidade de Berkeley.

A seguir serão apresentados alguns detalhes a respeito desses *middlewares* para grades computacionais, focando particularmente na área de comunicação, que é o objetivo do presente trabalho.

2.1 Globus Toolkit

O Globus Toolkit vem sendo desenvolvido desde o final da década de 1990, e tem como objetivo a criação de ferramentas que permitam o uso de infraestruturas computacionais distribuídas para atender as demandas de "organizações virtuais" (FOSTER; KESSELMAN; TUECKE, 2001) científicas e comerciais (FOSTER, 2006).

Para isso, o Globus oferece três conjuntos de componentes: implementações de serviços, que fornecem recursos importantes de infraestrutura, como gerência de execução, acesso e movimento de dados, gerência de réplicas, monitoria e descoberta, entre outros; contenedores, usados para integrar serviços desenvolvidos pelo usuário, fornecendo recursos como segurança, gerência de estados e outros mecanismos frequentemente usados na construção de serviços; e bibliotecas de clientes, que permitem o desenvolvimento de clientes que usem operações oferecidas pelos serviços das duas categorias anteriores (FOSTER, 2006).

Esses serviços usam largamente Web Services (BOOTH et al., 2004) para definir suas interfaces e estruturar seus componentes. A utilização de Web Services facilita o desenvolvimento de arquiteturas orientadas a serviços, como é o caso do Globus, oferecendo diversos recursos e extensões para aspectos como segurança e gerência de serviços, entre outros.

Por fazer um uso extensivo de Web Services, a comunicação entre os componentes é realizada utilizando o *framework* SOAP de comunicação (GUDGIN et al., 2007), adotado pelos Web Services.

Por fazer uso de vários padrões como Web Services e SOAP, e ter sua arquitetura dividida em vários serviços, o Globus é um sistema bastante robusto e adaptável a vários usos distintos, mas acaba por ser demasiado complexo para ser usado eficientemente em um ambiente didático.

2.2 BOINC

O BOINC (Berkeley Open Infrastructure for Network Computing) tem suas origens no projeto SETI@home (ANDERSON et al., 2002), criado em 1999, e visa fazer uso de computação com recursos públicos, isto é, usando ciclos computacionais de computadores de voluntários para resolver problemas que grupos pequenos de pesquisadores não teriam capacidade para resolver independentemente (ANDERSON, 2004).

Para atingir esse objetivo, o BOINC é organizado na forma de projetos autônomos, que têm servidores independentes, mas ainda assim permitindo que os voluntários participem de vários projetos simultaneamente.

Um projeto deve ter um ou mais servidores, com os quais os participantes se comunicam. Os servidores incluem os servidores de escalonamento, que distribui unidades de trabalho entre os voluntários e recebendo as respostas, e os servidores de dados, que fornecem os dados necessários para os cálculos e recebem os dados das respostas.

Os voluntários participam de um projeto executando um cliente, que é composto de vários componentes que se comunicam através de chamadas remotas de procedimentos, e se comunicam com os servidores utilizando o protocolo HTTP.

Por ser baseado em computação voluntária, o BOINC precisa se preocupar com fatores como autenticação e verificação dos resultados, o que adiciona algumas complexidades, que, unidas à diferença de foco comparado à computação em grade tradicional, complica o uso do BOINC para motivos didáticos.

3 DESCRIÇÃO DA FERRAMENTA

Neste capítulo é detalhado o processo de desenvolvimento do sistema de comunicação para um gerente de grades computacionais. O *software* desenvolvido visa fornecer uma camada base de comunicação para um sistema de distribuição de trabalhos maleáveis em uma grade computacional heterogênea e dinâmica. Inicialmente é descrito esse sistema, que é chamado Tuxur, e posteriormente é descrita em mais detalhes a camada de comunicação implementada para o Tuxur.

3.1 Descrição do Tuxur

Neste trabalho foi desenvolvida uma camada de comunicação para um sistema de gerência de grades computacionais. Para a identificação dos requisitos para essa camada de comunicação, é necessária uma análise da arquitetura desse sistema de gerência. Nesta seção é descrito o Tuxur, que é o sistema de gerência de grades para o qual foi desenvolvida a camada de comunicação, listando suas necessidades e requisitos.

Este sistema está sendo desenvolvido com o objetivo de experimentar com distribuição de trabalhos maleáveis, isto é, distribuição de trabalhos que podem ser quebrados em partes menores em uma grade heterogênea e dinâmica.

Para isso, foi projetado um sistema organizado em árvore composto por gerentes e trabalhadores. Além disso, ainda há um componente chamado de lançador, que é responsável por inicializar e estabelecer a comunicação com novos gerentes e trabalhadores da grade. Essa divisão visa separar as tarefas de execução das tarefas de manutenção da infra-estrutura, como descoberta, lançamento e conexão de nós de computação.

A seguir, está detalhado um pouco do funcionamento de cada componente deste sistema.

3.1.1 Trabalhador

O trabalhador é o componente que é responsável por computar os trabalhos e devolver os resultados a seu chefe. Para evitar que um trabalhador fique ocioso, ele tem uma fila de trabalhos a computar.

O trabalhador é também responsável por manter seu gerente informado sobre sua capacidade de cálculo e sobre o estado de sua fila.

3.1.2 Gerente

O gerente é o componente que gerencia um conjunto de trabalhadores, distribuindo trabalhos para serem computados por eles.

Cada gerente tem um outro gerente como seu chefe, exceto pelo gerente raiz, de forma que a grade é estruturada como uma árvore. Todo gerente é visto pelo seu chefe como um trabalhador com capacidade de trabalho maior, pois para o chefe é indiferente a forma como um trabalho será realizado.

Um certo nó da grade pode ter uma quantidade distinta de gerentes e trabalhadores: um *cluster* pode conter apenas um gerente mas um trabalhador para cada processador, enquanto uma máquina utilizando programação em placas gráficas pode operar somente como trabalhador.

Para realizar a distribuição dos trabalhos, o gerente coleta informações como a sua capacidade, que é a capacidade do seu nó mais a capacidade dos seus trabalhadores e a carga atual do seu nó e dos seus filhos. Usando essas informações, o gerente deve manter as filas dos trabalhadores que gerencia suficientemente cheias e equilibradas. Esse equilíbrio é baseado em um tempo de autonomia alvo, de forma que cada trabalhador deve ter trabalho suficiente para um certo tempo, sem a necessidade de contato com seu chefe. Dessa forma, evita-se o excesso de volume de comunicações.

3.1.3 Lançador

O lançador é responsável pela descoberta e lançamento de novos nós, isto é, a execução dos gerentes e trabalhadores e o estabelecimento de um canal de comunicação entre eles.

Quando um novo gerente ou trabalhador deve ser adicionado na grade, o lançador executa o gerente ou trabalhador, e então informa ao gerente que será o chefe dele como

ele deve entrar em contato com o novo nó.

Para realizar tal tarefa ele deve saber características dos nós, para garantir que possa estabelecer um canal de comunicação entre eles, inserindo uma passarela para tradução entre sistemas de comunicação diferentes entre eles, se necessário, e para lançar uma versão correta do programa de acordo com a arquitetura do nó.

3.1.4 Trabalho

Um trabalho no Tuxur deve ser maleável. O usuário do Tuxur deve desenvolver seu trabalho de forma que ele obedeça uma interface pré-definida, que contém formas de se obter o tamanho do trabalho, isto é, o tempo computacional aproximado necessário para sua computação, e define formas para que o trabalho seja quebrado em trabalhos menores e reagrupe as respostas desses trabalhos para gerar a resposta completa, e também uma forma de serialização, para permitir que o trabalho seja enviado através da rede.

3.2 Comunicação Entre os Componentes do Tuxur

Esta seção contém uma visão de alto nível no componente de comunicação, listando as necessidades e decisões necessárias para atender aos requisitos de comunicação do Tuxur.

Devido aos requisitos de comunicação do sistema se optou por manter a comunicação o mais simples possível, baseada na troca de mensagens pré-definidas entre os componentes. Também há um caráter de experimentação no sistema, o que elimina inicialmente preocupações com segurança e integridade.

Essa simplicidade do nível baixo de comunicação deve permitir que essas características possam ser adicionadas mais tarde com facilidade, além de tornar a comunicação facilmente portável e expansível, e simplifica a implementação de passarelas para traduzir a comunicação entre nós com sistemas de comunicação diferentes.

A seguir serão descritos os cenários identificados nos quais é necessária a comunicação, assim como as mensagens escolhidas para fornecer tal comunicação.

3.2.1 Lançamento

O lançador é responsável por lançar e estabelecer a comunicação entre um novo gerente ou trabalhador e a parte já existente da grade. Para tal, ele realiza o procedimento a seguir:

1. O lançador executa o programa que deve ser lançado, passando como argumento um identificador para o novo nó e uma descrição de como ele deve se comunicar com o lançador.

2. O novo gerente ou trabalhador envia uma mensagem para o lançador contendo o seu identificador e uma descrição do seu canal de comunicação.

3. O lançador envia uma mensagem para o gerente que será o chefe do novo trabalhador contendo o identificador e a descrição do canal de comunicação do novo trabalhador.

4. O gerente chefe envia uma mensagem para o novo trabalhador contendo o identificador e a descrição do canal de comunicação do chefe.

Nesse procedimento podem ser identificadas três mensagens: uma enviada do novo trabalhador para o lançador, uma do lançador para o chefe do novo trabalhador e uma do chefe para o novo trabalhador.

A primeira mensagem, enviada do novo trabalhador ou gerente para o lançador, chamada $NODE\_STARTED$, contém o identificador do trabalhador e a descrição do seu canal de comunicação.

A mensagem enviada do lançador para o chefe, chamada NEW_WORKER, também contém o identificador do trabalhador e a descrição do seu canal de comunicação.

A última mensagem, enviada do gerente chefe para o novo trabalhador, chamada $BOSS\_CONTACT$, e também contém o identificador do chefe e a descrição do seu canal de comunicação.

Na figura 3.1 pode ser visto um exemplo da adição de um novo nó a uma grade. No primeiro momento, o lançador executa o programa de trabalhador no novo nó, passando um canal de comunicação com o lançador como argumento. Após isso, o novo trabalhador envia uma mensagem $NODE\_STARTED$ para o lançador, com a descrição do canal de comunicação que o chefe deverá usar para se comunicar com ele. Então o lançador envia uma mensagem NEW_WORKER para o chefe, que envia uma mensagem $BOSS\_CONTACT$ através do canal recebido do lançador e com isso a comunicação é dada como estabelecida.

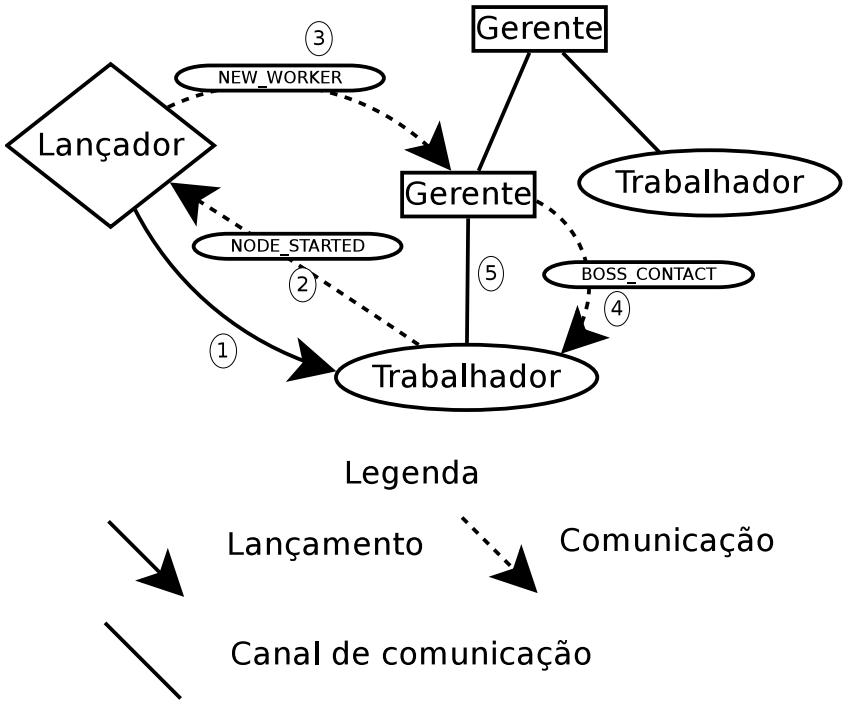

Figura 3.1: Exemplo do procedimento de lançamento

3.2.2 Propagação de estado

Para que um gerente possa distribuir trabalhos de forma adequada ele precisa saber do estado dos seus trabalhadores. Tal estado é composto por capacidade, que é uma medida de quanto trabalho o trabalhador consegue realizar por unidade de tempo, e carga de trabalho, que indica a quantidade de trabalho que o trabalhador ainda tem por realizar.

Para que essa informação seja propagada através da grade foi criada uma mensagem chamada *NODE_STATUS*, que deve ser enviada por todo trabalhador para seu gerente regularmente, e deve conter o identificador do trabalhador, a sua carga de trabalho e a sua capacidade.

3.2.3 Distribuição de trabalhos

Usando as informações de estados os gerentes devem distribuir os trabalhos conforme adequado através da grade. Para que isso possa ser feito, o gerente deve poder enviar um trabalho a ser realizado para um de seus trabalhadores, e os trabalhadores devem poder enviar respostas de trabalhos para seus chefes.

Para isso foram criadas duas mensagens. A primeira, chamada *NEW_JOB*, é enviada de um gerente a um trabalhador para entregar um trabalho a ser realizado por esse tra-

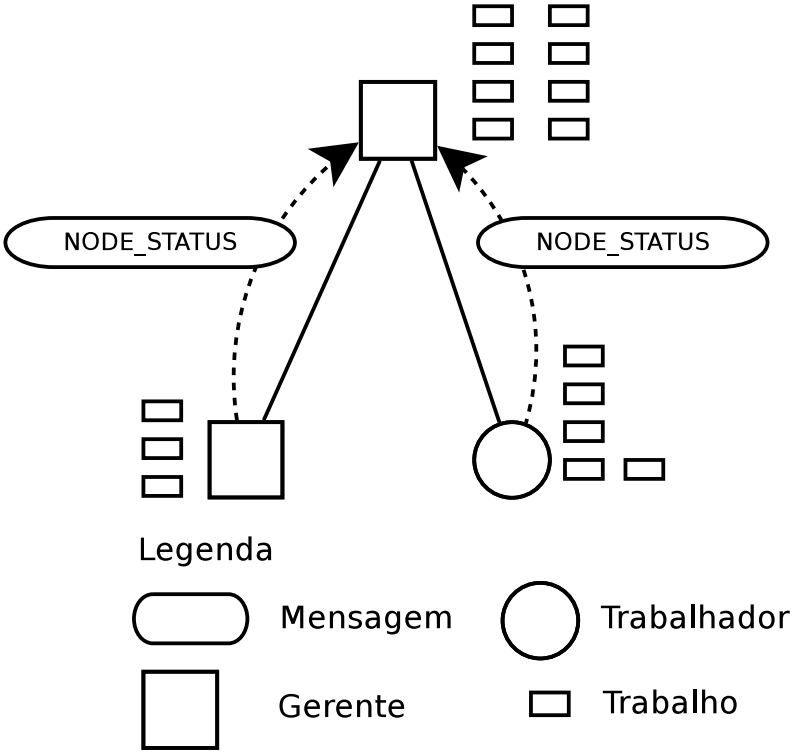

Figura 3.2: Exemplo de distribuição de trabalho, parte 1

balhador. Essa mensagem deve conter o identificador do trabalho e os dados necessários para reconstruir o trabalho no trabalhador.

A segunda mensagem, chamada *JOB_RESULT*, é enviada de um trabalhador para seu chefe para entregar a resposta de um trabalho que foi computado. Ela deve conter o identificador do trabalhador, o identificador do trabalho, e os dados do resultado da computação.

Nas figuras 3.2, 3.3 e 3.4 é representado um exemplo de um cenário de uso das mensagens *NODE_STATUS* e *NEW_JOB*. Na figura 3.2, os trabalhadores enviam seu estado para o gerente, que decide que um deles está com uma carga de trabalho muito baixa e envia um novo trabalho para ele, conforme visto na figura 3.3. Na figura 3.4, o trabalhador recebeu o trabalho e o adicionou na sua fila de trabalhos.

3.2.4 Mensagens

Nesta seção serão revistos os tipos de mensagens criados baseado nos cenários de comunicação descritos, bem como detalhados os campos que compõem cada tipo de mensagem.

NODE_STARTED Mensagem enviada de um trabalhador ou gerente recém criado para

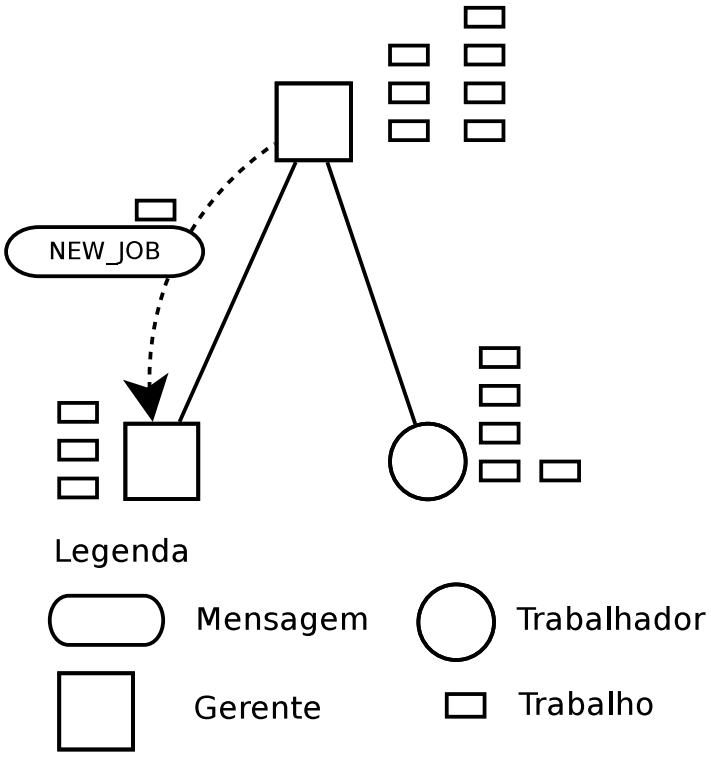

Figura 3.3: Exemplo de distribuição de trabalho, parte 2

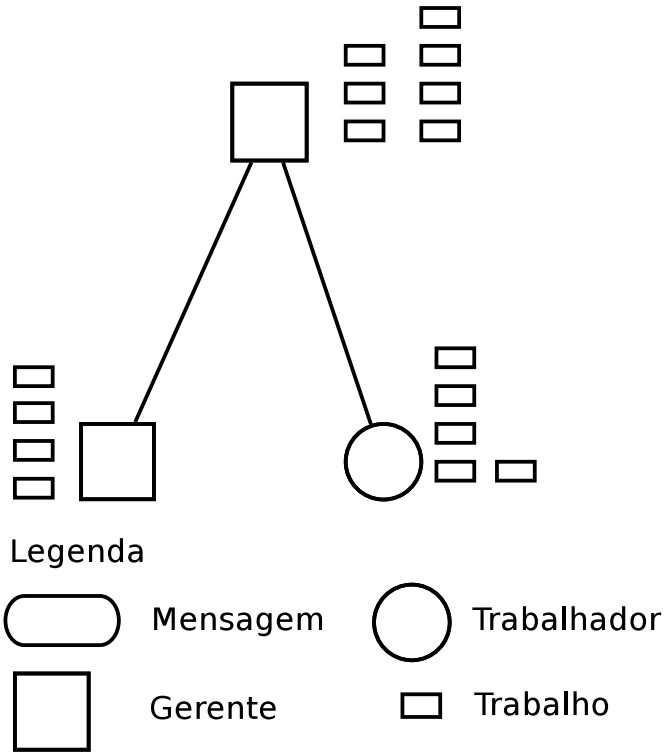

Figura 3.4: Exemplo de distribuição de trabalho, parte 3

o lançador. Contém a identificação do novo nó e a descrição do canal com a qual deve-se comunicar o chefe desse nó para estabelecer a conexão.

NEW_WORKER Mensagem enviada pelo lançador para um gerente já existente, para informar da existência de um novo trabalhador. Contém a identificação do nó trabalhador e a descrição do canal a ser aberto para comunicação com ele.

BOSS_CONTACT Mensagem enviada pelo chefe para um novo trabalhador, para estabelecer a comunicação. Contém a identificação do trabalhador.

NEW_JOB Mensagem enviada pelo chefe a um trabalhador com um novo trabalho a ser executado. Contém uma identificação do trabalho e os dados necessários para a computação do mesmo.

NODE_STATUS Mensagem enviada por um trabalhador para seu chefe, para informar seu estado. Identifica o trabalhador e seu estado, definido por dois inteiros: a carga de trabalho ainda por ser executado e a capacidade do trabalhador.

JOB_RESULT Mensagem enviada por um trabalhador para seu chefe, contendo o resultado de um trabalho já calculado. Contém o identificador do nó, o identificador do trabalho e os dados da resposta.

Além dos campos acima descritos, todas as mensagens têm um campo para designar o tipo da mesma. Na tabela 3.1 estão identificados os campos de cada mensagem.

3.3 Implementação da Comunicação

Nesta seção serão descritos detalhes relativos à implementação da comunicação para o Tuxur, descrito nas seções 3.1 e 3.2.

A camada de comunicação tem como objetivo permitir a implementação dos componentes do Tuxur de uma forma simples e sem necessidade de preocupação com detalhes relativos à comunicação. Para isso, foi decidido que a comunicação seria exposta para os componentes na forma de chamadas remotas de procedimentos e métodos.

Como o sistema também tem a portabilidade como objetivo, deve ser evitado o uso de soluções específicas a uma linguagem, como o Java RMI. Para atingir a portabilidade necessária, foi decidido que a comunicação se daria através da troca de mensagens, compostas no nível mais baixo por vetores de *bytes*.

Tabela 3.1: Campos e tipos de dados das mensagens

Mensagem	Campos
NODE_STARTED	Tipo da mensagem
	Identificador do trabalhador
	Descrição do canal
NEW_WORKER	Tipo da mensagem
	Identificador do trabalhador
	Descrição do canal
BOSS_CONTACT	Tipo da mensagem
	Identificador do trabalhador
NEW_JOB	Tipo da mensagem
	Identificador do trabalho
	Dados do trabalho
NODE_STATUS	Tipo da mensagem
	Identificador do trabalhador
	Carga de trabalho
	Capacidade
JOB_RESULT	Tipo da mensagem
	Identificador do trabalhador
	Identificador do trabalho
	Dados da resposta

Também pode ser necessária a comunicação entre máquinas que não usam o mesmo protocolo de rede, de forma que seria necessária a implementação de uma passarela para traduzir as mensagens entre os dois protocolos. Uma comunicação simples e dividida em dois níveis facilita a implementação destas passarelas.

As próximas seções descrevem esses dois níveis de comunicação, inicialmente o canal de comunicação, que implementa a camada de portabilidade seguido pela implementação das chamadas remotas.

3.3.1 Canal de Comunicação

Para facilitar a implementação das interfaces de chamadas remotas, e permitir uma fácil portabilidade entre sistemas de rede distintos, o sistema de comunicação foi dividido em duas camadas, sendo que a mais baixa delas, o canal de comunicação, simplesmente abstrai os detalhes da rede, oferecendo uma interface de envio e recebimento de vetores de *bytes* a ser utilizada pela camada superior.

Neste trabalho, essa camada básica de comunicação foi implementada utilizando comunicação através de *sockets*, utilizando TCP. Esse canal básico de comunicação oferece,

além da comunicação simples com envio e recebimento de vetores de *bytes*, uma forma textual de descrever o canal de comunicação para que a conexão possa ser estabelecida pela outra ponta. Essa forma textual é uma URI da forma `socket://endereçoIP:porta` no caso da comunicação através de *sockets*.

É possível criar um canal em modo de escuta, em que ele fica aguardando uma conexão, ou, passando uma descrição de canal como argumento na criação do mesmo, de forma que ele conecte em um outro canal que está em modo de escuta.

Em Java, esse canal é representado por um objeto que possui construtores oferecendo as formas de criação descritas, e métodos para envio, recebimento e obtenção da descrição do canal.

Em C, ele é representado por uma estrutura que deve ser criada através de dois procedimentos de criação oferecendo os dois modos de criação descritos, e procedimentos para envio, recebimento e obtenção da descrição que recebem a estrutura como argumento.

3.3.2 Chamadas Remotas

Para isolar a implementação dos componentes da comunicação e facilitar o porte entre ambientes que usam sistemas de comunicação diferentes a comunicação foi exposta para os componentes através de chamadas remotas de métodos e procedimentos.

Para expor os campos das mensagens nas chamadas remotas são utilizados alguns tipos de dados, descritos a seguir. Para a codificação do tipo da mensagem se optou pela utilização de um inteiro de dois *bytes*. Os identificadores de nós e trabalhos foram definidos como sendo campos textuais, e são representados na comunicação como vetores de caracteres ASCII terminados em nulo. A carga de trabalho e a capacidade são valores escalares, e são codificados como inteiros de quatro *bytes*. Os dados dos trabalhos e das respostas são definidos pelo usuário do Tuxur, na implementação do trabalho, e para não restringir a escolha do implementador do trabalho tais dados são expostos como vetores de *bytes*. Na tabela 3.2 esses tipos estão sumarizados, juntamente com os tipos equivalentes expostos pelas interfaces em Java e em C. Para serem encapsulados em vetores de *bytes* para serem enviados pelo canal de comunicação, todos os inteiros são codificados em ordem *big-endian*.

Em Java, isso foi feito através de objetos que operam como *proxies* para os nós remotos. Foi criada uma classe para cada componente do sistema, e os componentes devem

Tabela 3.2: Tipos de dados utilizados

Descrição	Java	C
Inteiro de dois *bytes*	`short int`	`int16_t`
Inteiro de quatro *bytes*	`int`	`int32_t`
Vetor de caracteres terminado em nulo	`String`	`char *`
Vetor de *bytes*	`byte[]`	`size_t` + `char *`

criar um objeto para cada nó remoto com o qual desejem se comunicar.

Cada componente deve ter um objeto implementando uma interface que descreve um método para cada tipo de mensagem que pode ser recebido por esse componente. Esse objeto deve ser passado para o *proxy* na criação, para que o *proxy* chame os métodos ao receber uma mensagem.

Esses *proxies* oferecem métodos correspondentes às mensagens que podem ser enviadas para o nó que ele representa.

No código 3.1 pode ser visto um exemplo de uma possível implementação de um trabalhador em Java, sob o aspecto da interface de comunicação.

Em C, foi definida uma função para cada tipo de mensagem, e os componentes devem implementar as funções correspondentes às mensagens que espera receber.

Para o envio de mensagens, foi definida uma função para cada mensagem que pode ser enviada, e os componentes devem chamar as funções passando como argumento a estrutura que representa o canal de comunicação com o nó que deve receber a mensagem.

No código 3.2 há um exemplo das partes de uma possível implementação de um trabalhador em C referentes à comunicação.

Para o funcionamento desses sistemas, são criadas *threads* que ficam permanentemente recebendo as mensagens do canal de comunicação, desempacotando elas e chamando os métodos ou funções correspondentes, enquanto as funções e métodos de envio empacotam os argumentos recebidos no formato apropriado para a mensagem e enviam a mensagem através do canal de comunicação.

```
class ExampleWorker implements Worker {

  String nodeId;
  RemoteLauncher launcher;
  RemoteManager boss;
  Queue<Job> jobs;
  ⋮
  ExampleManager(String id, String launcherDescription) {
    nodeId = id;

    /* Cria um proxy para o lançador e um para o gerente.
     * Passa "this" como o objeto que implementa as chamadas recebidas
     */
    launcher = new LauncherProxy(launcherDescription, this);
    boss = new ManagerProxy(this);

    //Envia a mensagem NODE_STARTED para o lançador
    launcher.nodeStarted(nodeId, boss.channelDescription());
    ⋮
  }

  //Implementação das mensagens que um trabalhador recebe

  //Implementação do recebimento da mensagem BOSS_CONTACT
  public void bossContact(String id) {
    System.out.printf("Boss␣found␣me.", nodeId, id);
  }

  //Implementação do recebimento da mensagem NEW_JOB
  public void newJob(String jobId, byte[] jobDescription) {
    Job job = unpackJob(jobId, jobDescription);
    jobs.add(job);
  }
  ⋮
}
```

Código 3.1: Exemplo de uso da comunicação em Java

```
typedef struct {
  char *workerId;
  channel_t bossComm;
  queue_t jobs;
} worker_t;

int main(int argc, char *argv[])
{
  worker_t w;
  channel_t laucherComm;
  char *launcherChannelDescription = argv[2];
  w.workerId = argv[1];
  queue_create(w.jobs);

  /* Cria um proxy para o lançador e um para o gerente.
   * O ponteiro passado como último argumento será passado para as funções
   * de recebimento de mensagens
   */
  connectToChannel(&laucherComm, launcherChannelDescription, &w);
  createChannel(&w.bossComm, &w);

  //Envia a mensagem NODE_STARTED para o lançador
  sendNodeStarted(&laucherComm, w.workerId,
                  getChannelDescription(&w.bossComm));
  ⋮
}

//Implementação das mensagens que um trabalhador recebe

//Implementação do recebimento da mensagem BOSS_CONTACT
void managerContact(void *context, char *workerId)
{
  printf("Boss␣found␣me.\n");
}

//Implementação do recebimento da mensagem NEW_JOB
void newJob(void *context, char *jobId, char *jobDescription, int jobDescriptionSize)
{
  worker_t *w = (worker_t *)context;
  job_t *myJob = unpackJob(jobId, jobDescription, jobDescriptionSize);
  queue_add(w->jobs, myJob);
}
⋮
```

Código 3.2: Exemplo de uso da comunicação em C

4 AVALIAÇÃO

Nesse capítulo será analisado o projeto concluído de acordo com os requisitos, assim como serão detalhados os resultados de testes de desempenho realizados.

4.1 Funcionamento da Ferramenta

Conforme os requisitos levantados foi desenvolvida uma camada de comunicação para um sistema de gerência de grades computacionais.

No desenvolvimento da ferramenta os detalhes de comunicação foram mantidos isolados em uma camada simples que oferece uma interface para as camadas superiores, de forma que o sistema pode ser facilmente portado para outros mecanismos de comunicação simplesmente reimplementando-se essa camada mais inferior.

Além disso, como não se usou nenhum mecanismo exclusivo a uma linguagem no projeto do sistema, ele pode ser portado para novas linguagens através da definição de uma interface a ser usada nessa linguagem e da implementação dessa interface, conforme foi realizado em Java e em C.

O projeto também atingiu o objetivo de permitir que os componentes do Tuxur possam se comunicar sem saber detalhes da comunicação, oferecendo uma interface simples de chamadas remotas de métodos e procedimentos.

4.2 Desempenho

Para analisar o desempenho da ferramenta desenvolvida foram criados alguns cenários de uso da mesma para testar a eficiência da camada de comunicação.

4.2.1 Testes

Foram realizados testes para analisar o desempenho da camada inferior de comunicação.

Os cenários utilizados para testar o desempenho foram: comunicação entre dois pontos e comunicação entre um ponto e vários outros pontos. Esses cenários foram escolhidos por ocorrerem em uma rede organizada em forma de árvore, como é o caso do Tuxur. Os testes foram realizados utilizando-se a implementação em Java da comunicação.

Para a realização dos testes foi implementado um programa que utiliza a camada base de comunicação criada nesse trabalho para enviar um número de mensagens entre os pontos, de forma que o tempo levado para a troca dos dados possa ser comparado com o desempenho máximo esperado da rede.

Para o teste do cenário de comunicação entre um ponto e vários pontos foram utilizados cinco pontos, de forma que um deles se comunicou com os outro quatro simultaneamente.

Os testes foram realizados utilizando-se o programa implementado para enviar um número de mensagens totalizando um tráfego total de 10 *megabytes* em cada ponto, de forma que o total transferido é de 10 *megabytes* no teste de comunicação entre dois pontos e de 40 *megabytes* no teste de comunicação entre um ponto e vários pontos. Foram usados três tamanhos de mensagens: pequenas (100 *bytes*), médias (1 *kilobyte*) e grandes (1 *megabyte*). Os resultados foram comparados com o tempo levado para transferir uma quantidade igual de dados diretamente pela rede, consistindo na transmissão do total de dados através de *sockets* TCP em Java, assim como com o desempenho máximo esperado da rede.

4.2.2 Resultados

Nas tabelas 4.1 e 4.2 estão sumarizados os tempos obtidos na realização dos testes, assim como os tempos obtidos em uma transferência bruta de dados através da rede, e as taxas de transferências correspondentes. Os tempos exibidos na tabela são a média de três execuções, para evitar que uma execução afetada por condições adversas atrapalhe o julgamento dos resultados, apesar dos tempos de execução terem apresentado uma variação baixa ao longo das amostras.

Pode ser observado que o teste entre vários pontos com mensagens pequenas obteve

Tabela 4.1: Resultados dos testes realizados entre dois pontos

	Pequenas	Médias	Grandes	*Sockets*	Teórico
Tempo (s)	$1,394$	$1,482$	$1,423$	$1,112$	$0,8$
Taxa de transferência (MB/s)	$7,17$	$6,75$	$7,03$	$8,99$	$12,5$

Tabela 4.2: Resultados dos testes realizados entre vários pontos

	Pequenas	Médias	Grandes	*Sockets*	Teórico
Tempo (s)	$4,663$	$4,589$	$4,693$	$3,792$	$3,2$
Taxa de transferência (MB/s)	$8,58$	$8,72$	$8,52$	$10,55$	$12,5$

um desempenho de cerca de $81,33\%$ do obtido na transmissão bruta de dados, o que, considerando-se que a comunicação se deu através de várias mensagens de 100 *bytes*, é um bom desempenho, pois além de impor uma sobrecarga de dados de 4% por causa dos 4 *bytes* que indicam o tamanho de cada mensagem, também há um certo tempo gasto entre o envio de cada mensagem que não ocorre na transmissão bruta.

Também pode ser notado que não há uma grande variação de desempenho de acordo com o tamanho das mensagens, o que é desejável, pois as mensagens enviadas no sistema podem ter tamanhos variados.

5 CONCLUSÃO

Neste trabalho foi desenvolvida uma camada de comunicação para o sistema dinâmico de gerência de grades computacionais Tuxur. Essa camada de comunicação tem por objetivo possibilitar a implementação dos componentes do Tuxur sem a necessidade de se preocupar com detalhes da comunicação, e tornar a portagem de tais componentes para ambientes diferentes simples.

Para o desenvolvimento desse sistema foi inicialmente realizado um levantamento de requisitos de comunicação do Tuxur, seguido pelo projeto do sistema de comunicação baseado em uma camada inferior para isolar detalhes da comunicação e uma camada superior expondo a comunicação como chamadas remotas para os componentes.

Após o desenvolvimento foi realizada uma análise do cumprimento dos requisitos originais, assim como testes de desempenho, que mostraram que o projeto obteve sucesso em atingir os objetivos a que se propôs.

Este trabalho oferece uma implementação básica de uma camada de comunicação, mas possibilita expansões e adições de recursos que possam vir a ser úteis. Ficam como sugestões para trabalhos futuros:

- Adição de novos recursos necessários para a expansão do Tuxur;
- Implementação de funcionalidades de segurança;
- Implementação da camada inferior para outros mecanismos de comunicação além de *sockets*;
- Implementação do sistema em outras linguagens de programação.

REFERÊNCIAS

ANDERSON, D. P. **BOINC**: a system for public-resource computing and storage. Pittsburgh: 5th IEEE/ACM International Workshop on Grid Computing, 2004.

ANDERSON, D. P.; KORPELA, E.; LEBOFSKI, M.; WERTHIMER, D. **SETI@home**: an experiment in public-resource computing. [S.l.]: Communications of the ACM, 2002.

BOOTH, D.; HAAS, H.; MCCABE, F.; NEWCOMER, E.; CHAMPION, M.; FERRIS, C.; ORCHARD, D. **Web Services Architecture**. [S.l.]: W3C, 2004.

FOSTER, I. **Globus Toolkit Version 4**: software for service-oriented systems. [S.l.]: IFIP International Conference on Network and Parallel Computing, 2006.

FOSTER, I.; KESSELMAN, C. **Globus**: a metacomputing infrastructure toolkit. [S.l.]: The International Journal of Supercomputer Applications, 1997.

FOSTER, I.; KESSELMAN, C. **The Grid**: blueprint for a new computing infrastructure. [S.l.]: Morgan Kaufmann Publishers, 1999.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. **The Anatomy of the Grid**: enabling scalable virtual organizations. [S.l.]: The International Journal of Supercomputer Applications, 2001.

GUDGIN, M.; HADLEY, M.; MENDELSOHN, N.; MOREAU, J.-J.; NIELSEN, H. F.; KARMARKAR, A.; LAFON, Y. **SOAP Version 1.2 Part 1**: messaging framework (second edition). [S.l.]: W3C, 2007.

SNIR, M.; OTTO, S.; HUSS-LEDERMAN, S.; WALKER, D.; DONGARRA, J. **MPI**: the complete reference. Cambridge: MIT Press, 1995.