Tiago Machado

Avaliação de Desempenho de uma Aplicação do tipo Bag-of-Tasks utilizando o Middleware OurGrid

Orientador:
Marcelo Lobosco

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Juiz de Fora

Dezembro, 2009

Monografia submetida ao corpo docente do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como parte integrante dos requisitos necessários para obtenção do grau de bacharel em Ciência da Computação

Prof. Marcelo Lobosco, D. Sc.
Orientador

Prof. Rodrigo Weber dos Santos, D. Sc.

Prof. Eduardo Barrere, D. Sc.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Grades computacionais são cada vez mais utilizadas para prover o poder de processamento necessário para aplicações com grandes demandas computacionais. Um dos *middlewares* utilizados para a criação da infraestrutura de grades computacionais é o OurGrid. O objetivo deste trabalho é avaliar a viabilidade de implantação do *middleware* em questão bem como avaliar seu desempenho. Para isto foi utilizada a aplicação Genecodis. Os resultados obtidos mostram a) a viabilidade do uso do OurGrid em grades computacionais, b) o *overhead* causado pela transferências de arquivos relacionados a execução e c) o tempo de execução de cada tarefa que forma o *job* de modo individual.

1 Introdução

A busca por maior poder computacional parece ser uma corrida sem fim: à medida que a indústria lança processadores mais poderosos, idealizam-se novas aplicações que demandam processadores ainda mais poderosos. Neste cenário surge a computação paralela como alternativa para viabilizar a execução de aplicações cujas demandas computacionais superem o poder de processamento oferecido por um único processador. Quando empregadas técnicas de computação paralela, não apenas um, mas vários processadores ou núcleos (*cores*) de processamento são utilizados simultaneamente para realizar o processamento de um único programa. A execução de aplicações com grande demanda computacional é portanto viabilizada pela soma do poder de processamento de vários processadores.

Tais processadores ou núcleos de processamento podem estar presentes fisicamente em uma mesma máquina, em uma arquitetura chamada multiprocessador (ou *multicore*), ou mesmo distribuídos em máquinas distintas, em uma arquitetura chamada multicomputador. O menor custo e maior escalabilidade da arquitetura multicomputador, quando comparada a arquitetura multiprocessador, foram fatores que contribuiram, ao longo das décadas de 1990 e 2000, para a popularização dos chamados agregados (*clusters*) de computadores e, mais recentemente, das grades (*grids*) computacionais.

Agregados de computadores são ambientes computacionais formados por grupos interconectados de computadores. Os recursos dos computadores são combinados, dando aos usuários a ilusão de tratar-se de um único sistema computacional. O sistema operacional ou mesmo um *middleware* é utilizado para esta finalidade. Agregados podem ser formados por computadores dedicados, muitas vezes interligados por redes de conexão de baixa latência, ou por computadores de uso compartilhado, onde apenas os ciclos de computação ociosos são utilizados. Neste último caso os agregados são também chamados de NOW (*Networks of Workstations*)(ANDERSON; CULLER; PATTERSON, 1995). Instituições com grandes quantidades de computadores podem assim aproveitar os ciclos ociosos de suas máquinas para formar um agregado de computadores. As grades computacionais são um passo além dos agregados computacionais por constituirem-se de recursos computacionais

amplamente distribuídos, não limitados portanto a uma única instituição.

O objetivo deste trabalho é estudar um *middleware* para computação em grade, o OurGrid(CIRNE et al., 2006). A principal contribuição deste trabalho é a avaliação de desempenho de uma aplicação paralela quando executada em OurGrid.

O trabalho está assim estruturado: O segundo capítulo apresenta as grades computacionais, enquanto o terceiro descreve o *middleware* utilizado neste trabalho. A avaliação de desempenho está no quarto capítulo e, por fim, o quinto capítulo apresenta as conclusões e trabalhos futuros.

2 Grades Computacionais

As grades computacionais (*grid computing*) surgiram em meados da década de 1990 com o objetivo de viabilizar a execução de aplicações com grande demanda computacional. As mesmas consistem em uma arquitetura composta da união dos mais diversos tipos de plataformas computacionais pertencentes a entidades espalhadas ao redor do globo terrestre, portanto localizados em locais geograficamente dispersos. Tal arquitetura abre as portas para a execução de aplicações em uma escala antes simplesmente impossível de se obter em um único agregado de computadores ou mesmo em um único supercomputador. As grades constituem-se portanto em uma importante ferramenta de suporte a ser utilizada para resolver os problemas que se constituem hoje como os Grandes Desafios da Computação, em áreas como biomedicina, modelos climáticos e física (HOARE; MILNER, 2009).

A arquitetura das grades viabiliza o compartilhamento de poder de processamento e capacidade de armazenamento pela internet, a fim de obter baixo custo e otimização das tarefas realizadas. A idéia desta arquitetura é de ir bem além de ser apenas um meio de comunicação entre computadores, e sim de tornar a internet em um único e vasto recurso computacional.

O termo grade computacional deriva de uma analogia dos sistemas de energia elétrica (*eletric power grid*). Nestes sistemas o acesso à eletricidade é amplamente difundido e de simples utilização: o usuário utiliza a eletricidade sem ao menos saber qual sua origem ou como ela é transmitida. A idéia é que a grade possua um nível de abstração semelhante às redes elétricas, de forma que o uso de recursos computacionais distribuídos seja tão simples quanto ligar qualquer equipamento à energia elétrica. Neste contexto, tal como acontece com a energia elétrica, é importante que o usuário não veja a grade computacional como um conjunto distribuído de recursos, mas como um recurso unificado que oferece serviços de computação. Bastaria ao usuário submeter sua aplicação e receber o resultado final, de forma centralizada e simples (FOSTER et al., 2002).

Podem ser destacados os principais tipos de grades (FOSTER et al., 2002):

- Grades computacionais: É uma grade voltada para o processamento de dados. Neste caso, é importante que se tenham máquinas de alto desempenho;
- Grades de dados: Tem como principal objetivo disponibilizar às mais diversas organizações armazenamento e acesso a dados compartilhados. Nestas grades é transparente ao usuário a localização dos dados.

A arquitetura das grades é comumente dividida em camadas. Nesta arquitetura, as camadas mais baixas são focadas no *hardware* e as camadas mais altas no usuário.

A camada responsável por fazer a comunicação entre os recursos é a camada de rede. Superior a ela está a camada de recursos que podem ser: computadores, serviços de armazenamentos, etc. A terceira e principal camada da grade é o *middleware*, este é responsável por tornar possível os mais distintos equipamentos participarem da grade. Na camada mais alta da grade está a aplicação propriamente dita, esta pode ser de diferentes áreas de conhecimento. Nesta última camada se encontram os *softwares* responsáveis por auxiliar no gerenciamento da grade.

Nos últimos anos as grades computacionais tem sido incrementadas em seu poder computacional pelo baixo custo dos equipamentos e pelo aumento no número de instituições que doam seu poder computacional a outras instituições. Este crescente número faz com que pesquisadores da área de processamento paralelo e distribuído busquem formas de tornar as grades computacionais rápidas, confiáveis, seguras e transparentes, propondo soluções para os novos desafios decorrentes dos ambientes de grades computacionais.

2.1 Middleware

Segundo (COULOURIS; DOLLIMORE; KINDBERG, 2007) um *middleware* em sistemas distribuídos é uma camada de *software* que fornece uma abstração de programação, assim como o mascaramento da heterogeneidade das redes, do *hardware*, de sistemas operacionais e linguagems de programação adjacentes. Ou seja, cabe ao *middleware* tratar das diferenças no nível dos sistemas operacionais e do *hardware*. Em geral, os sistemas interconectados que utilizam *middlewares* são muito heterogêneos, o que significa que os mesmos são compostos por dispositivos muito diferentes entre si.

Além de resolver os problemas de heterogeneidade, o *middleware* fornece um modelo computacional uniforme para ser usado pelos programadores de serviços e de aplicativos distribuídos. Estes modelos incluem a invocação remota de objetos, a notificação remota de eventos, o acesso remoto a banco de dados e o processamento de transação distribuído. Como exemplos da heterogeneidade temos as redes de computadores, os sistemas de telecomunicação e principalmente a Internet. Em suma, as principais funções do *middleware* são (OBJECT, 2009):

- Conectar aplicações;
- Garantir interoperabilidade entre arquiteturas distribuídas;
- Garantir comunicação em um meio heterogêneo, com diferentes sistemas operacionais, linguagens de programação e arquiteturas de computador;
- Localizar serviços ou aplicações transparentemente na rede;
- Manter o serviço confiável e disponível.

Exemplos de *middlewares* existentes:

- CORBA (CORBA, 2009);
- gLite (SUN, 2009a);
- Remote Method Invocation (RMI) (SUN, 2009b);
- OurGrid (CIRNE et al., 2006).

Dos *middlewares* listados acima, apenas gLite e OurGrid foram desenvolvidos para facilitar o desenvolvimento de aplicações paralelas em grades computacionais. RMI e CORBA foram criados para dar suporte ao desenvolvimento de aplicações distribuídas. Neste trabalho, focaremos especificamente o *middleware* OurGrid.

3 OurGrid

3.1 Introdução

O projeto OurGrid é desenvolvido desde dezembro de 2004 pela Universidade Federal de Campina Grande (PB) em parceria com a Hewlett Packard (HP), Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES). O projeto chamado de *free-to-join* é aberto (sob licença GPL), cooperativo e de código livre.

O OurGrid é um *middleware* para criação de grades computacionais, formado por computadores, *clusters* de computadores ou ambos. Inicialmente visa atender aos laboratórios de pequena e média escala, sem possibilidade de custeio de uma tecnologia de grades. Até então estes necessitavam de pessoal altamente capacitado para instalar e operar.

Os laboratórios que utilizam o OurGrid doam seu poder computacional ocioso, e podem consumir recursos de outros laboratórios da mesma comunidade. A idéia básica é que as pessoas não utilizam os recursos computacionais de seus computadores o tempo todo, assim sendo, estes recursos podem ser utilizados quando ociosos.

O projeto utiliza a arquitetura de sistemas distribuídos par-a-par (do inglês *p2p - peer-to-peer*). Conforme (COULOURIS; DOLLIMORE; KINDBERG, 2007) as redes *p2p* são caracterizadas pela descentralização das funções, ou seja, cada nó da rede funciona tanto como servidor quanto cliente. Os nós da rede p2p podem diferir em capacidade de processamento, capacidade de armazenamento, largura de banda, entre outras. Neste contexto vários *middlewares* para sistema *peer-to-peer* tem surgido oferecendo a capacidade para compartilhar recursos computacionais, armazenamento e dados presentes em computadores "nos limites da internet", em uma escala global. Nas grades *peer-to-peer* disponíveis pelo OurGrid, há grande escalabilidade pois o cooperado pode utilizar os recursos computacionais de outras grades, seguindo a regra de que se pode usar a grade de maneira

proporcional ao que se doa, ou seja, quanto mais doado mais poderá ser usado. De acordo com (CIRNE et al., 2006), o projeto utiliza uma rede de favores, que exclui um dos grandes problemas das redes *peer-to-peer*: o de usuários que apenas utilizam recursos da rede, sem jamais doar.

Neste trabalho o *middleware* foi instalado e configurado em sistema operacional Linux, especificamente a distribuição Kubuntu, mas o mesmo pode pode ser utilizado em Windows e, em alguns casos, ambos. Com exceção do Openfire, os módulos do OurGrid são desenvolvidos em Java, e, por isto, possuem grande portabilidade e podem funcionar em qualquer sistema operacional que possua uma implementação da Máquina Virtual Java.

Atualmente a plataforma pode ser usada para executar aplicações do tipo *Bag-of-Tasks* (BoT). As aplicações do tipo BoT são aplicações paralelas cujas tarefas são completamente independentes, não havendo portanto comunicação entre as mesmas durante a fase de processamento. As mesmas são utilizadas em inúmeras áreas como: mineração de dados, cálculos de fractais, computação gráfica, etc (CIRNE et al., 2006). Este tipo de aplicação é importante pois torna o tratamento a erros e os mecanismos de segurança mais eficazes.

O projeto se encontra atualmente na versão 4.1.5, lançada em 22 de maio de 2009, e está em constante atualização. O *middleware* é basicamente composto por quatro módulos: *Openfire*, *Peer*, *Broker* e *Worker*, conforme a Figura 1.

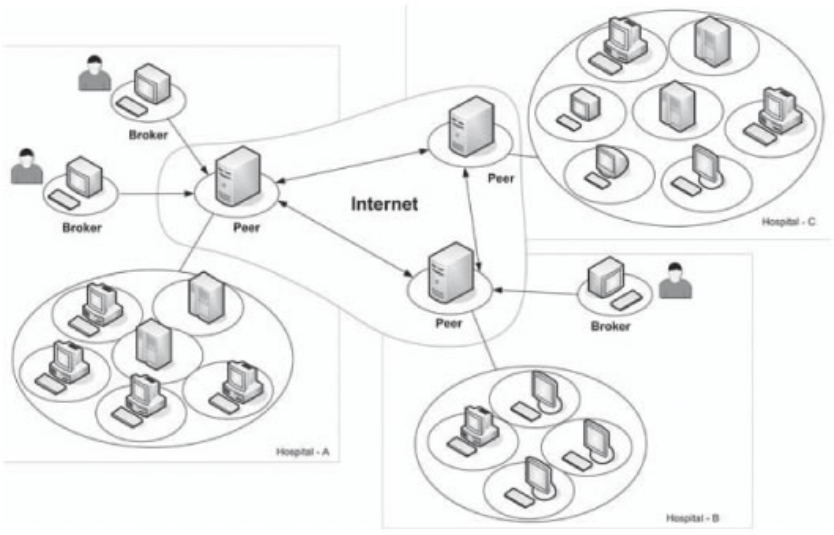

Figura 1: Arquitetura do *middleware* OurGrid.

3.2 OpenFire

Openfire é o servidor XMPP certificado para o OurGrid. Segundo (JIVE, 2009) o Openfire é um software de troca de mensagens instantâneas baseado no protocolo Jabber, que é de código aberto e tem bases em XML (JABBER, 2009). Este tipo de protocolo para troca de mensagens instantâneas é muito utilizado em programas na internet como o GTalk.

O XMPP é uma padronização do Internet Engeneering Task Force (IETF) para comunicação em tempo real com base no XML. É um conjunto de protocolos de fácil entendimento que fornecem os serviços necessários para comunicação instantâneas através de redes locais ou internet, além de serviços de integração com redes estrangeiras.

Através de uma interface gráfica organizada e intuitiva, em poucos minutos é possível configurá-lo conforme sugerido pelo próprio site do projeto OurGrid. Todos os usuários do OurGrid deverão estar autenticados no servidor Openfire, para que as mensagens possam ser trocadas entre eles. Assim sendo, serão necessários usuário e senha para cada *worker*, para o *broker* e para o *peer*.

3.3 Peer

O módulo *peer* do OurGrid é o responsável por gerenciar todas as máquinas destinadas ao processamento na grade. É ele quem recebe o programa a ser executado na grade e envia para cada máquina trabalhadora (*worker*) a respectiva tarefa a ser executada.

Este módulo possui uma chave pública que deve ser repassada aos trabalhadores (*workers*), para que os mesmos sejam aceitos, do contrário o módulo não envia tarefas ao trabalhador. Desta forma fica difícil uma máquina não cadastrada no *peer* executar uma tarefa, evitando que um usuário mal intencionado acesse informações indevidas, ou as altere.

No módulo *peer* o usuário pode optar por entrar na comunidade do OurGrid. Nesse caso usuários externos podem utilizar o poder computacional de sua grade. É possível ainda consultar todos os laboratórios disponíveis para processamento que utilizam o *middleware* OurGrid e o respectivo número de máquinas ociosas dos mesmos.

3.4 *Worker*

Como o próprio nome diz, são os trabalhadores da grade. Estes são gerenciados pelo *peer*. Todas as máquinas que o usuário deseja disponibilizar para processamento deverão estar executando uma instância do *worker*. Conforme dito, o *worker* deve possuir a chave pública do *Peer* para ser aceito pelo mesmo. O *worker* possui ainda um par de chaves pública e privada que são geradas automaticamente na primeira inicialização deste e que servem para distinguir os *workers* entre si.

O *worker* recebe do *peer* o comando e os arquivos necessários para a execução, após executado, retorna o resultado ao *peer*.

Os *workers* possuem ainda uma importante propriedade chamada *Idleness Detector* que detecta quando o usuário está ou não utilizando os dispositivos de entrada do computador. Este mecanismo garante que a máquina só seja disponibilizada para o uso do *middleware* quando o usuário pára de utilizar os dispositivos de entrada.

3.5 *Broker*

O *Broker* é o módulo necessário apenas para a máquina que irá solicitar as tarefas à grade. Este módulo pode estar na rede local ou externo a ela. Os trabalhos (*Jobs*) são enviados ao *peer* pelo *broker*. No entanto o usuário do *broker* deve estar cadastrado no *peer*. Assim apenas usuários autorizados podem utilizar um Grid específico.

No *broker* é possível configurar tanto o número máximo de transferência de arquivos simultâneos como o de réplicas simultâneas para cada tarefa. É possível ainda definir o máximo de execuções antes de uma tarefa falhar e o máximo de falhas aceitáveis para que uma máquina seja excluída da execução de determinada tarefa. Através destas configurações é possível criar um ambiente mais adequado à execução para cada tipo de tarefa.

3.6 Segurança

Conforme explicado o *middleware* utiliza alguns mecanismos de segurança como a troca de chaves pública e privada. Apesar disto, sabe-se que a grade pode ser executada em uma rede local ou pela internet, aumentando a vulnerabilidade do sistema. No caso do administrador da grade não conhecer e nem confiar em todos os usuários desta, é

interessante que seja utilizado o sistema de virtualização. A virtualização no OurGrid faz com que a execução se torne ainda mais segura pois, neste caso, as tarefas são executadas dentro de uma máquina virtual sem acesso à rede. Neste contexto, um usuário mal intensionado ao tentar danificar ou roubar dados da execução conseguirá apenas acessar a máquina virtual. Todo o conteúdo em execução fica protegido e, além disto, o estado da máquina virtual poderá ser reinicializado por um outro *worker* tornando possível ao *job* retomar a execução do mesmo ponto na qual foi interrompida. São dois os aplicativos que podem ser utilizados para virtualização no OurGrid: VServer no caso de Linux e VMWare Server para Windows.

4 Avaliação do Ambiente

4.1 Ambiente

Todos os testes realizados neste trabalho foram feitos em um dos laboratórios do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora (UFJF), o Infocentro. Esse laboratório consiste de 22 máquinas Intel Celeron D, com 3.06GHz e 1GB de memória principal, executando o sistema operacional Linux versão 2.6.24-16.30, distribuição Kubuntu 8.04. Durante os experimentos, este conjunto de máquinas foi dividido em vinte *workers*, um *peer* e um *broker*. A conexão entre as máquinas é feita através de dois *switches* 3Com modelo 3C16470 com 16 portas cada. As máquinas do laboratório foram alocadas exclusivamente para os experimentos, ou seja, nenhum usuário utilizava as máquinas durante a realização dos experimentos.

4.2 *Benchmark*

Como o objetivo deste trabalho não é apenas de analisar a viabilidade de implantação do *middleware* na UFJF, mas também avaliar o desempenho do mesmo, foi utilizada uma aplicação para esta análise. A escolha de apenas uma aplicação tem como objetivo analisar o desempenho de um modo mais detalhado. Para este propósito, foi escolhido o Genecodis (NOGALES-CADENAS et al., 2009) como *benchmark*. Esta é uma ferramenta baseada em grade computacional que procura características que frequentemente aparecem em um conjunto de genes e são categorizadas com base em sua significância. A ferramenta pode ser utilizada, por exemplo, para determinar alguma característica biológica ou combinações de características significativamente associadas a uma lista de genes (CARMONA-SAEZ et al., 2007). O aplicativo utilizado é dividido em 36 tarefas. Para a execução de cada tarefa é enviado para cada *worker* um arquivo compactado no formato .tar.gz de aproximadamente 2,5Mb, que contém o executável da aplicação, bem como os dados a serem computados.

4.3 Resultados Experimentais

Para avaliar a aceleração (*speedup*), ou seja, a influência do aumento do número de máquinas no tempo total de execução da aplicação, executou-se cada tarefa do Genecodis 16 vezes (para se obter um desvio padrão abaixo de 9%), variando o número de *workers*. Nesta seção foi calculada a aceleração à partir do valor médio dos tempos de execução encontrados para completar todas as tarefas que compõem uma execução da aplicação. O desvio padrão para cada execução em um grupo de máquinas. Os dados coletados estão resumidos na Tabela 1. Com base na tabela, foi possível traçar o gráfico da aceleração obtida, representada na Figura 2. A Figura 2 representa, no eixo x, o número de máquinas utilizadas na execução e, o eixo y, representa o ganho computacional (*speedup*) baseado no tempo de execução com uma máquina.

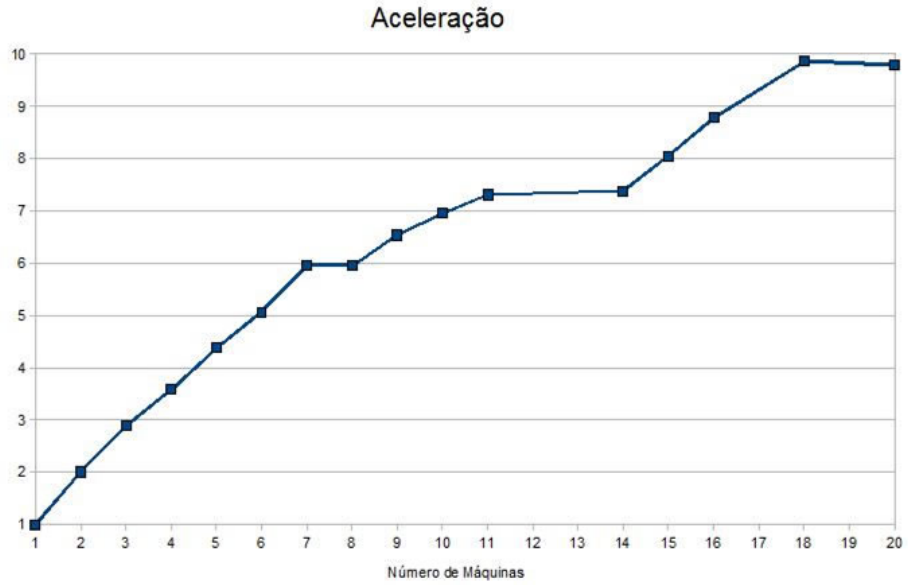

Figura 2: *Speedup* relativo do aplicativo Genecodis variando a quantidade de *workers*

Dois aspectos chamam a atenção em relação a aceleração obtida na Figura 1. O primeiro aspecto diz respeito a aparência de escada observada no gráfico. A aceleração é mantida em algumas configurações, apesar do aumento no número de máquinas, observando-se em seguida um novo aumento na aceleração, à medida que o número de máquinas é novamente aumentado. O segundo aspecto diz respeito a baixa aceleração observada. Como a aplicação é do tipo *Bag-of-tasks*, esperava-se uma aceleração linear.

Em relação ao aspecto de escada observado no gráfico de aceleração, verificamos que a causa decorre diretamente da relação entre o número de tarefas que formam o *job* e o número de máquinas empregadas para a execução, o que divide as máquinas em grupos

Nro de Máquinas	Tempo Médio (ms)	Aceleração	Desvio Padrão
1	606385	1	0,03
2	301912	2,01	0,05
3	210048	2,89	0,05
4	169185	3,58	0,07
5	138290	4,38	0,06
6	119725	5,06	0,07
7	101871	5,95	0,04
8	101848	5,95	0,07
9	92735	6,54	0,09
10	87161	6,96	0,08
11	82960	7,31	0,09
14	82121	7,38	0,08
15	75298	8,05	0,08
16	69058	8,78	0,08
18	61460	9,87	0,06
20	61948	9,79	0,09

Tabela 1: Tabela do Tempo Médio (ms), Aceleração e Desvio Padrão do respectivo número de máquinas.

de execução, conforme ilustra a equação 4.1.

$$GE = \left\lceil \frac{N}{M} \right\rceil \tag{4.1}$$

onde GE é número de grupos de execução, N o número de tarefas e M o número de máquinas. Como exemplo temos para 36 tarefas e 14 máquinas um grupo de execução igual a 3. Isso quer dizer que 14 das 36 tarefas a serem executadas seriam enviadas para as 14 máquinas disponíveis, em seguida as próximas 14 tarefas seriam enviadas para as 14 máquinas, e por fim sobrariam 8 tarefas a serem enviadas para 8 das 14 máquinas.

A partir da equação 4.1, foi possível formar a Figura 3, que consiste na divisão das 36 tarefas em grupos de execução para 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 15, 16, 18 e 20 máquinas. Onde cada linha representa a configuração para uma máquina, e cada coluna o número de tarefas que compõem um grupo de execução.

A Figura 3 ilustra que o número de grupos de execuções formados a partir da equação 1 é o mesmo em alguns casos. Com isto, é possível esperar 10 distintos tipos de comportamento na execução do algoritmo, conforme a divisão das 36 tarefas pela quantidade de máquinas disponíveis. Em especial a aceleração com 6 e 7; 9, 10 e 11; 14, 15, e 16; e 18 e 20 máquinas, respectivamente, tenderia a ser igual ou muito próxima pelo fato das mesmas pertencerem aos mesmos grupos de excecução.

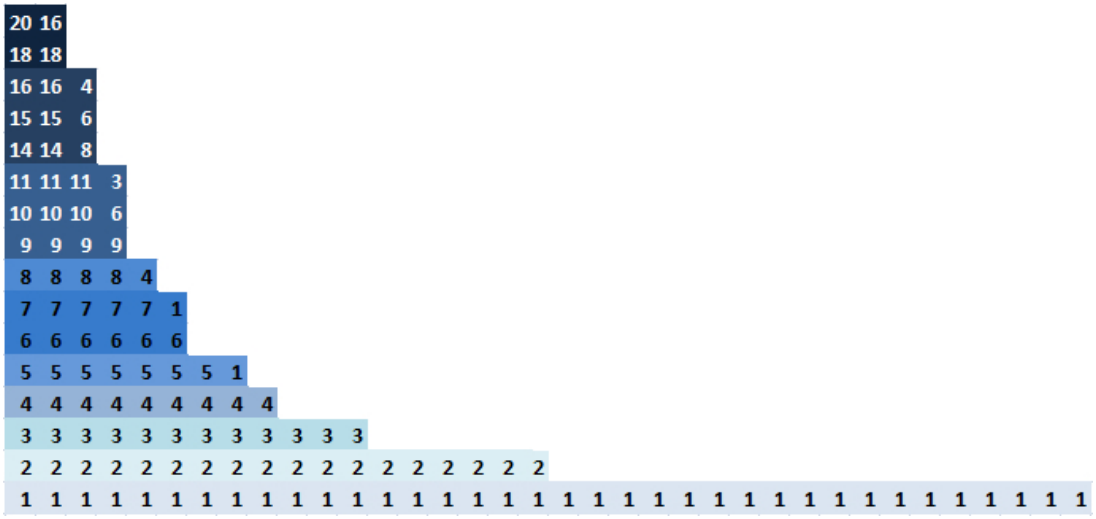

Figura 3: Número de grupos de execuções formados a partir da equação 4.1

Entretanto este comportamento de 10 grupos distintos só se aplicará se o tempo de execução de cada uma das 36 tarefas que formam o *job* for aproximadamente igual. Assim as máquinas alocadas para a execução de cada tarefa terminariam a execução em tempos próximos e receberiam novas tarefas, também em tempos próximos. Neste caso o comportamento de grupos teóricos preservaria o esperado.

Coletou-se então todos os tempos referentes a execução de cada uma das 36 tarefas que compõem um *job*. O tempo total de execução de cada tarefa consite na união de 3 diferentes tempos: a) o tempo do envio do arquivo com o executável e os dados necessários para a execução da aplicação entre o *peer* e o *worker*, chamado de fase inicial; b) o tempo de execução propriamente dito da aplicação, chamado de fase remota; e c) o tempo de transferência do arquivo com o resultado, chamado de fase final. A Figura 4 apresenta, para cada tarefa, o tempo médio total de execução.

Através da Figura 4 é possível observar que as tarefas de número 31 e 35 possuem tempo de execução muito superior às demais (34 e 27 segundos, respectivamente, contra 5 e 7 segundos das outras).

Com isto o comportamento teórico esperado para os grupos de 6 e 7; 9, 10 e 11; 14, 15, e 16; e 18 e 20 foi divergente do real. O comportamento em escada da Figura 2 pode ser explicado pela divisão dos grupos de execução em conjunto com os tempos de execução das tarefas 31 e 35. Por exemplo, na execução com 6 e 7 máquinas tem-se 6 grupos de execução, porém, no caso específico da execução com 7 máquinas, o último grupo é de apenas uma máquina. Como a tarefa 31 demora mais tempo para finalizar do que todas as outras, a tarefa de número 36, que compõe o último grupo de execução, irá finalizar antes

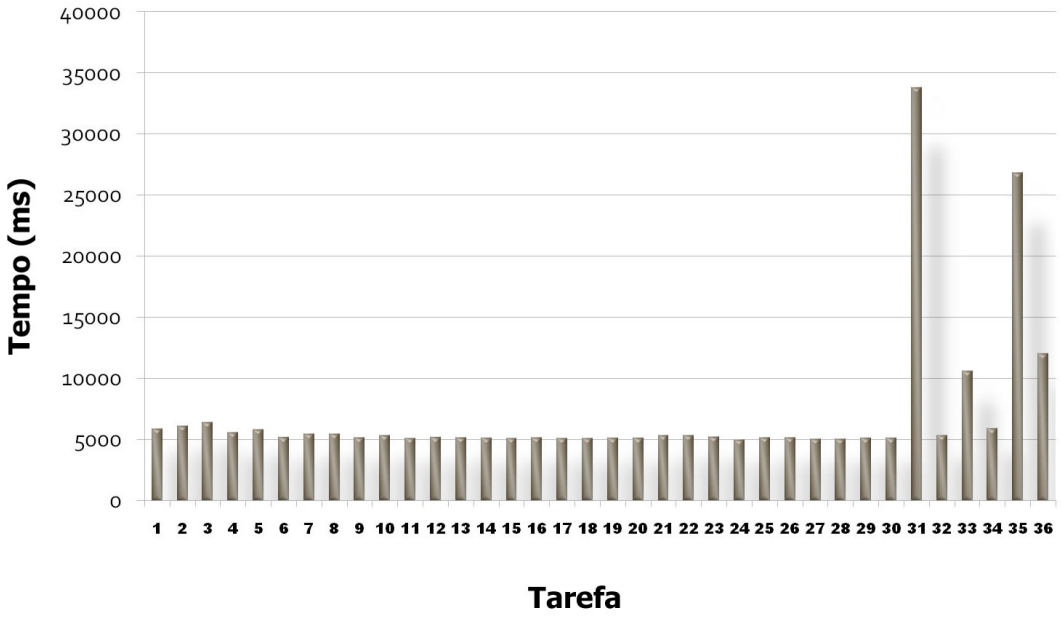

Figura 4: Tempo médio total de execução de cada tarefa do aplicativo Genecodis em milissegundos

da 31, que compõe o penúltimo grupo. Logo o número efetivo de grupos de execução é 5 e não 6. Isto justifica a execução com 7 máquinas ter tempo médio semelhante à com 8 máquinas, pois a mesma também possui 5 grupos de execução. Entretanto, seu comportamento contraria o teórico, no qual a execução com 7 máquinas deveria ter tempo semelhante à com 6 (ambas com 6 grupos de execução). O comportamento contrário ao teórico também ocorre para os grupos 11 e 14 e 18 e 20. E é justamente nestes 3 grupos, 7 e 8, 11 e 14, e 18 e 20 que o comportamento em escada é observado.

Outro fator que influiu na divergência entre o resultado experimental e o teórico foi o tempo médio de cada uma das 3 fases das tarefas. O que se pode observar é que a fase final é responsável por uma parcela significativa do tempo de execução total, representando de 51% a até 61% deste tempo (Tabela 2). Apesar da tarefa utilizada no experimento ser do tipo *bag-of-tasks*, este grande *overhead* também explica o porque da aceleração linear não ter sido obtida.

FASE	TEMPO (ms)
Fase Inicial	233,22
Fase Remota	2879,46
Fase Final	3529,39

Tabela 2: Tempo médio total da execução de cada uma das três fases do aplicativo Genecodis em milissegundos

Os testes mostram a importância do balanceamento de carga no processamento dis-

tribuído. De acordo com a Lei de Amdahl (PATTERSON; HENNESSY, 2005), o ganho de desempenho quando da melhora de uma determinada parte do sistema depende da fração de tempo que esta parte é utilizada no processamento. No caso do algoritmo utilizado um melhor ganho no *speedup* seria obtido através de otimizações das tarefas que tem maior tempo de processamento, explicitamente as tarefas 31 e 35. Do contrário, por maior que seja o número de máquinas, o tempo total da excecução do algoritmo sempre será o da tarefa 31, tornando a escalabilidade da grade inútil.

5 Conclusão

Neste trabalho foi analisada a viabilidade de implantação do *middleware* OurGrid bem como o desempenho de uma aplicação do tipo *bag-of-tasks* quando executada neste ambiente. O *middleware* mostrou-se uma interessante ferramenta para a criação de grades computacionais, principalmente por ter o código livre e ser portável entre diferentes sistemas operacionais.

Este trabalho tem duas principais contribuições. Primeiramente foi possível observar o *overhead* no tempo de execução da aplicação Genecodis decorrente do tempo gasto na transferência do arquivo com resultados, limitando o desempenho da grade. Finalmente, o ganho computacional na execução da aplicação Genecodis não está apenas ligado ao número de máquinas utilizadas, mas também ao tempo de execução de cada tarefa de modo individual. Como trabalho futuro seria interessante verificar se este mesmo comportamento ocorre com outras aplicações. Para isso seria necessário desenvolver e portar outras aplicações para que possam ser executadas neste *middleware*.

Como o experimento em um laboratório foi bem sucedido, o próximo passo é implantar o OurGrid em todas as máquinas de todos os laboratórios (Infocentros) da Universidade Federal de Juiz de Fora e disponibilizá-lo para toda a comunidade do OurGrid. Os laboratórios totalizam hoje cerca de 400 máquinas, que ficam completamente ociosas por um período de 10 horas diárias (das 22h00 até às 08h00 de segunda à sábado) e durante todo o dia aos domingos. Mesmo durante os horários em que estão disponíveis para uso, é fácil verificar a existência de máquinas ociosas.

Referências

ANDERSON, T. E.; CULLER, D. E.; PATTERSON, D. A case for now (networks of workstations). *Micro, IEEE*, v. 15, n. 1, p. 54–64, 1995.

CARMONA-SAEZ, P. et al. Genecodis: A web-based tool for finding significant concurrent annotations in gene lists. *Genome Biology*, v. 8, 2007.

CIRNE, W. et al. Labs of the world, unite!!! *Journal of Grid Computing*, v. 4, n. 3, 2006.

CORBA. Object management group. *http://www.corba.org/ Acessado em agosto*, 2009.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Sistemas Distribuídos, Conceitos e Projeto*. [S.l.]: Bookman, 2007. ISBN 0321263545.

FOSTER, I. et al. The physiology of the grid: An open grid services architecture for distributed systems integration. In: . [S.l.: s.n.], 2002.

HOARE, T.; MILNER, R. Grand challenges in computing – research. *http://www.ukcrc.org.uk/gcresearch.pdf. Acessado em julho.*, 2009.

JABBER. Site oficial do jabber. *http://www.jabber.org/. Acessado em julho*, 2009.

JIVE, S. Site oficial do openfire. *http://www.igniterealtime.org/projects/openfire/index.jsp. Acessado em Agosto*, 2009.

NOGALES-CADENAS, R. et al. Genecodis: interpreting gene lists through enrichment analysis and integration of diverse biological information. *Nucleic Acids Research*, 2009.

OBJECT, W. What is middleware. *http://middleware.objectweb.org/ Acessado em Julho*, 2009.

PATTERSON, D.; HENNESSY, J. *Computer Organization and Design: The Hardware/software Interface*. [S.l.]: Morgan Kaufmann, 2005.

SUN, M. The globus alliance. *http://www.globus.org/ Acessado em setembro*, 2009.

SUN, M. Remote method invocation. *http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp Acessado em setembro*, 2009.