Trabalho de Formatura Supervisionado

Gerenciamento de Recursos para Grades Computacionais *Node Control Center*

Carlos Eduardo Moreira dos Santos
Orientador: Fabio Kon

Universidade de São Paulo
Instituto de Matemática e Estatística
Departamento de Ciência da Computação

Dezembro de 2009

Sumário

Parte I
Parte Técnica

1 Introdução

O trabalho foi desenvolvido dentro do InteGrade [1], um projeto multiuniversidades para desenvolvimento de um *middleware* para grades computacionais oportunistas.

Uma grade computacional agrega e gerencia recursos de diversos computadores e disponibiliza-os a aplicações. As máquinas podem estar em locais geográficos diferentes e até mesmo possuírem arquiteturas heterogêneas, mas a grade fornece ao usuário acesso a seus recursos de forma transparente. Pode-se utilizar um meio de comunicação também já existente, como redes locais ou Internet. Por oportunista, entende-se que será utilizado o tempo ocioso das máquinas, sem prejudicar a qualidade de serviço para o usuário local, dono dos recursos e que os compartilha [Gom09]. Nada impede que computadores dedicados também façam parte dessa infraestrutura.

A utilização do InteGrade é uma alternativa a supercomputadores que possui baixo custo, mas é poderosa, pois une máquinas subutilizadas através de conexões já existentes, somando recursos e possibilitando ao programador utilizá-los através de padrões bem conhecidos como *Message Passing Interface*[2] (MPI) e *Bulk Synchronous Parallel*[3] (BSP).

1.1 Motivação

Foi realizado um trabalho [Cor08] para prever os períodos de ociosidade dos nós (computadores que doam recursos), mas nem sempre é possível executar aplicações da grade em computadores que não tenham usuários enquanto são executadas.

Antes desse trabalho, o InteGrade executava aplicações nos nós com prioridade mínima, para que um eventual usuário da máquina não percebesse queda de desempenho. Apesar de atingir esse objetivo, o ajuste na prioridade do processo não evita que ele consuma toda a CPU, mantendo seu uso total a 100%. Isso acontecia na grande maioria dos casos, pois a grade, em geral, executa aplicações que requerem grande poder de processamento.

São muitas as consequências do aumento no uso do processador. Quanto mais ele é utilizado, maior os gastos com energia, não somente para fazê-lo funcionar, mas também para refrigerá-lo. Para refrigeração, é comum fazer-se uso de ventiladores e exaustores, sendo que em alguns casos, principalmente em *notebooks*, a velocidade deles é automaticamente ajustada conforme a temperatura. Dessa forma, o aumento no uso de CPU também acarreta em um nível maior de ruído.

Com a primeira ocorrência documentada no ano de 1992 (USENET), o termo *Green Computing*, traduzido para o português como Computação Verde ou TI Verde, tem ganhado cada vez mais repercussão. Uma das maneiras para tornar o uso da computação ambientalmente sustentável é pelo chamado uso verde, prática através da qual se reduz

[1] `http://www.integrade.org.br`
[2] `http://www.mcs.anl.gov/research/projects/mpi/`
[3] `http://www.bsp-worldwide.org/`

o consumo de energia dos computadores [4]. Ao diminuir o consumo de CPU, o trabalho tornará possível essa prática.

1.2 Objetivos

Como objetivo, deve-se fornecer ferramentas que permitam aos usuários das máquinas controlar a disponibilidade de seus próprios recursos na grade, minimizando o impacto que as aplicações do InteGrade possam causar, além de diminuir o gasto de energia, geração de calor e ruído causado pela refrigeração (como citado na Seção 1.1) ao limitar o uso da CPU.

Limitações de novos recursos poderão ser adicionadas no futuro e deve-se realizar o trabalho de maneira a permitir atualizações sem grandes alterações no código.

Pensando no usuário local (quem doa os recursos à grade), a interface deve ser descomplicada, pois ele não necessariamente possui conhecimentos avançados na área, assim como disposição para adquiri-los. Quanto mais o usuário tiver que aprender para configurar os recursos da maneira que desejar, menor será a probabilidade dele se dispor a utilizar o InteGrade para doar seus recursos à grade.

1.3 Problemas a Serem Resolvidos

Ainda não é possível limitar recurso algum no InteGrade. Para realizar essa tarefa, qualquer programa a ser adicionado no nó não deve sobrecarregá-lo nem dificultar a instalação ou uso do InteGrade por parte do usuário local. Preferencialmente, programas eventualmente adicionados devem ser multiplataforma, para não dificultar a portabilidade do InteGrade a outros sistemas operacionais.

Considerando a limitação de recursos, funções específicas a serem chamadas pelos aplicativos executados pela grade com o objetivo de limitar os recursos não devem ser desenvolvidas, pois exigiriam atualização de código existente e limitariam suas execuções em grades computacionais do tipo InteGrade. Além disso, códigos que não utilizassem as funções de limitação ganhariam mais recursos sem respeitar as decisões de quem os fornece.

A interface do usuário deve apresentar uma maneira simples, para leigos, de gerenciar os recursos, independentemente da complexidade desta tarefa. Configurações avançadas podem ser possíveis, desde que haja uma maneira alternativa mais simples e intuitiva.

2 Conceitos e Tecnologias Estudados

Foram estudados os programas: InteGrade, CPUReserve e interface para o *Node Control Center*. Os três foram integrados para tornar possível a reserva de CPU pelos usuários locais dos nós. Seguem informações sobre cada um deles, incluindo as tecnologias que utilizam.

[4] `http://en.wikipedia.org/wiki/Green_computing` (05/11/2009)

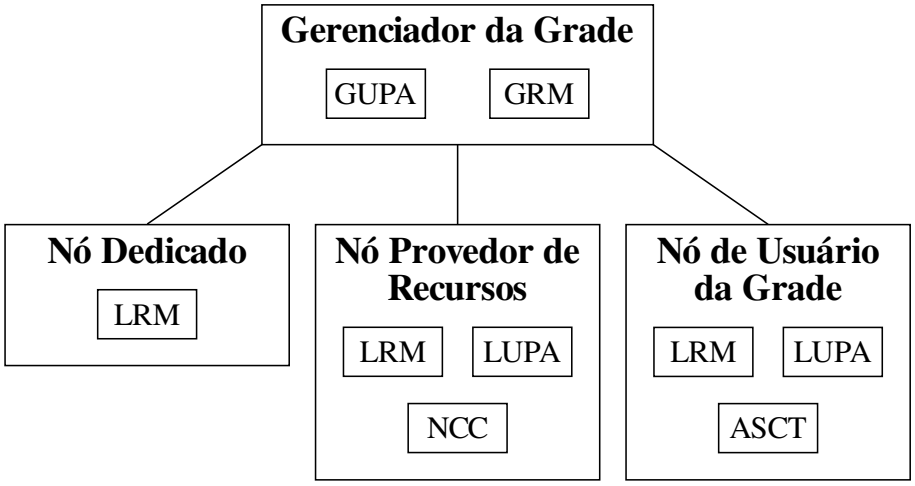

Figura 1: Arquitetura *Intra-Cluster* do InteGrade.

2.1 InteGrade

Estudando a arquitetura do InteGrade [GKG+04], notamos na Figura 1 que todo computador provedor de recursos executa o *Local Resource Manager* (LRM). É sua função coletar informações sobre o estado dos nós, como a utilização de CPU, disco e rede e enviá-las ao *Global Resource Manager* (GRM), que as utilizará para critérios de gerenciamento. Mas o mais relevante para o trabalho é o fato de ser o LRM quem inicia as aplicações da grade nos nós.

Para realizar alterações no código do LRM, é necessário conhecer a linguagem C++ e conceitos de programação orientada a objetos.

2.2 CPUReserve

Para controlar o uso de CPU, podemos utilizar tecnologia de virtualização [SN05]. Contudo, há uma grande sobrecarga em manter uma máquina virtual funcionando. Novas tecnologias como OpenVZ[5] e Xen[6] surgiram para minimizar este problema, mas requerem modificações no núcleo do sistema operacional. Essa tarefa só pode ser realizada por usuários com privilégios de administrador e dificulta ou até mesmo inviabiliza a instalação do InteGrade nos nós provedores de recursos.

Integrantes do Projeto InteGrade da PUC-Rio desenvolveram o CPUReserve [RC08], uma ferramenta para controle do uso de CPU em nível de usuário, ou seja, sem a necessidade de alterar o núcleo do sistema operacional. Para compreender e adaptar seu comportamento, é necessário conhecimentos sobre a linguagem C, estrutura de dados e chamadas ao sistema (*system calls*).

2.3 Interface para o *Node Control Center*

O *Node Control Center* (NCC) permite aos donos dos recursos controlar como a grade utiliza suas respectivas máquinas [GKG+04].

[5] `http://wiki.openvz.org/Main_Page`
[6] `http://xen.org`

Uma interface para o NCC já está disponível [LS03]. Para incluir o suporte ao CPUReserve e tornar possível a limitação do uso de CPU pelo usuário local (primeiro recurso a ser controlado no projeto), é necessário conhecer C++ (linguagem na qual o programa foi escrito), orientação a objetos, soquetes e também tecnologias para apresentar conteúdo através de um navegador Web (HTML, CSS, manipulação de imagens).

3 Atividades Realizadas

Definidos os aplicativos a serem utilizados (CPUReserve e NCC), bastaria, resumidamente, iniciar os dois novos programas nos nós provedores de recursos e fazer o LRM utilizar o CPUReserve para iniciar aplicações da grade. Porém, ao testar as ferramentas, notou-se que seriam necessárias alterações no CPUReserve e NCC que não têm como objetivo direto a integração entre os projetos.

Nesta seção, serão descritos os programas originais, assim como as modificações realizadas. Na Seção 4, serão mostrados os resultados finais.

3.1 CPUReserve

O CPUReserve é uma boa solução para os problemas de sobrecarga do nó e modificações do *kernel* citados na Seção 1.3. Como o programa baseia-se em chamadas ao sistema, os aplicativos da grade não precisam ser alterados para respeitar o limite de processamento. Além disso, o programa não sobrecarrega o nó, pois foi escrito (em linguagem C) para ser leve e eficiente, utilizando um servidor para centralizar o gerenciamento de vários processos, minimizando a sobrecarga ao acrescentar aplicativos para monitoramento. Outra vantagem do CPUReserve é o fato de também haver uma versão para Windows (não utilizada neste trabalho), possibilitando uma futura integração com a versão do InteGrade para esse mesmo sistema operacional.

O CPUReserve mantém um servidor controlando o tempo de execução dos programas que são iniciados por um cliente. O cliente se comunica com o servidor por soquetes enviando o nome do programa e seus argumentos. Depois, o servidor chama a função `fork` e, em seguida, `execvp` com as informações que acabou de receber, passando então a monitorar e controlar o tempo do novo processo criado.

Os processos são controlados através de alarmes temporizadores que são ajustados conforme as informações sobre o tempo de execução obtidas no pseudo-sistema de arquivos de processos, *procfs*. Quando um alarme é disparado, uma árvore de decisões é percorrida (Figura 2), podendo ocorrer, por exemplo, o congelamento do processo pelo sinal `SIGSTOP` ou a mudança de seu escalonador e sua prioridade.

3.1.1 Versão Original

Inicialização do Servidor. Um exemplo de inicialização do servidor é: `server 8000 3 1`. O primeiro argumento (`8000`) refere-se à porta de escuta do soquete; o segundo, à representação decimal da máscara de bits que tem como 1 as CPUs que poderão executar os programas da grade (no exemplo, `3` habilita o primeiro núcleo (0) e o segundo (1)); o último argumento (`1`) indica o limite de processamento do sistema (no máximo 1, que significa 100%). Esse limite existe para que o CPUReserve possa funcionar corretamente

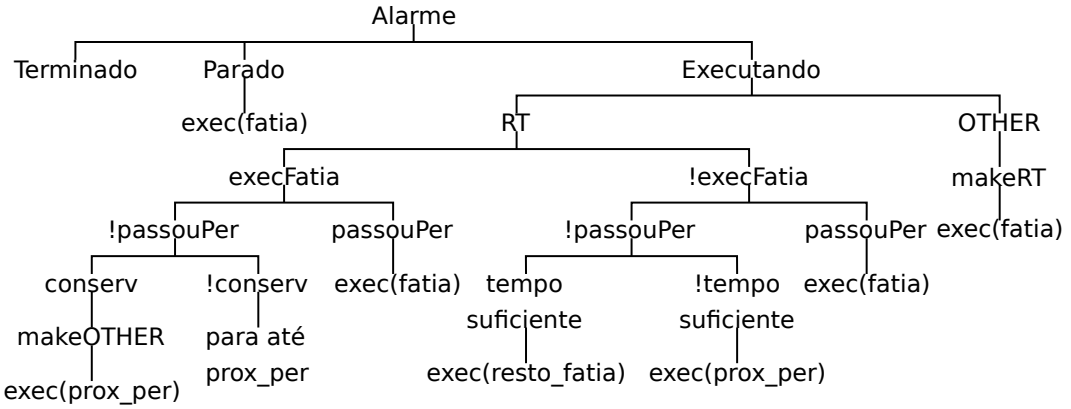

Figura 2: Árvore de decisão de um alarme no CPUReserve [RC08].

dentro de máquinas virtuais com restrição de uso de CPU. Como o *procfs* da máquina virtual não expressa o tempo de execução de processos e *threads* em relação a ela mesma, mas sim em relação ao sistema real, é necessário passar ao servidor do CPUReserve o limite de CPU da máquina virtual como argumento. Salvo esse caso, utilizamos `1` como limite de processamento do sistema.

É interessante notar que não só podemos especificar um limite para o uso dos núcleos como também exatamente quais deles se deseja utilizar.

Uso do Cliente para Iniciar Programas. Quando quisermos iniciar um programa e controlar o quanto de CPU ele consumirá no máximo, usamos o cliente como neste exemplo: `client 127.0.0.1:8000 1000 800 1 exec arg1 arg2...`. O primeiro argumento contém o endereço IP e porta do servidor do CPUReserve; o segundo, o tempo do período (em milissegundos); o terceiro, a fatia de tempo a ser consumida pelo programa durante o período; o quarto argumento refere-se à conservação, ou seja, caso o processo consuma toda a fatia especificada e o período ainda não tenha acabado, ele será congelado ou terá sua prioridade diminuída ao extremo, dependendo do valor deste argumento (0 ou 1, respectivamente). Por último, especificamos o caminho do programa e seus argumentos.

Através do exemplo dado, `exec arg1 arg2...` seria executado e, a cada 1000 ms, `exec` poderia usar 800 ms, ou seja, 80% do período. Na verdade essa porcentagem seria ainda corrigida pelo limite de processamento passado como parâmetro na inicialização do servidor. Supondo que o servidor fora iniciado com `0.6` sendo seu último argumento, 60% da fatia de tempo pedida pelo cliente (80%) seria reservada, ou seja, o programa consumiria, no máximo, 48% (60% × 80%) do período. Isso seria útil caso o CPUReserve estivesse sendo executado em uma máquina virtual limitada a utilizar 60% da CPU do sistema real.

Alterando o Limite de CPU. Para modificar os limites de tempo de execução dos aplicativos, utiliza-se o comando `adapt` com quase os mesmos parâmetros que `client`. A diferença é que, como a aplicação já foi iniciada, informamos seu *pid* ao invés do nome do programa e seus argumentos. Pelo comando `adapt 1000 400 0 1234`, o processo de *pid* 1234 passará a consumir 40% do período antes de ser congelado até o início do próximo.

Comportamento com *Tasks* Filhas. Quando um aplicativo lança uma *task* (*thread* ou novo processo), a sua fatia de tempo é compartilhada com todos seus filhos. Isso acontece para evitar que programas burlem o limite de CPU através da criação de múltiplas *tasks*.

3.1.2 Problemas Encontrados

Ao executar a versão original em um computador com duas CPUs, pôde-se notar alguns problemas indesejáveis para o InteGrade.

Travamentos Periódicos. Pausas periódicas na interface gráfica (atualizações de janelas, movimento do mouse) e na reprodução de áudio atrapalharam consideravelmente a interação entre computador e usuário (inaceitável em uma grade computacional oportunista).

O problema dos travamentos periódicos pôde ser minimizado através da diminuição dos valores absolutos de período e fatia de tempo dos aplicativos gereciados pelo CPUReserve e/ou da proporção entre a fatia e o período, mas ainda assim não foi possível resolvê-lo sem alteração de código.

Aplicações Rejeitadas ao Atingir Limite. Antes de executar uma aplicação informada pelo cliente, o servidor confere se a proporção entre a soma de fatias de tempo e de períodos pedidos (incluindo a nova requisição) ultrapassa o limite de reserva, uma constante pré-definida no código com valor máximo 1 (100%). Se a nova proporção ultrapassar essa constante, `RESERVATION_LIMIT`, o servidor não executará a aplicação que o cliente acabou de pedir, informando esse motivo através de um código de erro.

A primeira aplicação deve ser iniciada com a possibilidade de utilizar toda a CPU disponível, para não desperdiçar processamento. Porém, caso a grade decida executar mais uma aplicação nesse mesmo nó, isso não seria possível, pois não haveria mais fatias disponíveis para ela. Uma maneira de contornar esse problema seria esperar o término de uma aplicação antes de iniciar outra no nó, ou então diminuir o uso de CPU de todas as aplicações (pelo `adapt`) que já estejam em execução para que a soma de mais uma não ultrapasse o limite de reserva. A solução é apresentada na Seção 3.1.3.

Impossibilidade de Atualizar Parâmetros do Servidor. Uma vez iniciado, não podemos alterar os parâmetros do servidor. Para atualizar o limite de processamento, temos que terminar sua execução e reiniciá-lo com o novo limite. O servidor, ao ser terminado, finalizaria todos os processos controlados por ele, tornando também necessária a reinicialização de cada um deles. Além dessa alternativa ser trabalhosa, as aplicações da grade não obrigatoriamente salvam e recuperam seus estados, o que torna a atualização dos parâmetros do servidor inviável.

Subutilização de Múltiplas CPUs. Um teste foi realizado numa máquina com dois núcleos através de uma aplicação `primos` que lança um processo e ambos (pai e filho) consomem toda a CPU disponível. Para limitar o processamento a 90% de cada uma das CPUs (0 e 1), executou-se o Código 1. Contudo, ao monitorar o uso do processador, notou-se que apenas 45% de cada CPU estava sendo utilizado.

```
server 8000 3 1
client 127.0.0.1:8000 1000 900 0 primos
```

Código 1: Sequência de comandos que mostram a subutilização de múltiplos núcleos.

Para entender o que aconteceu, devemos lembrar do comportamento do CPUReserve em relação a *threads* ou processos filhos. Os 900 ms referem-se à soma dos tempos de execução do pai e do filho para evitar o favorecimento de aplicações com mais *tasks*. Em contrapartida, há 100% do processamento utilizado não aproveitado.

Ao tentar fixar o dobro de fatias (1800) para usar todo o processamento desejado, o CPUReserve rejeitou o pedido pelo fato de o período ser menor que o número de fatias. Conclui-se então que é impossível fornecer parâmetros que resultem no consumo de mais de 50% de cada CPU neste caso, ou mais de $\frac{100\%}{n}$ no caso geral, sendo n o número de núcleos a serem utilizados.

Servidor e Processos Executados como *root*. O servidor precisa ser executado com privilégios de administrador, e os processos a serem monitorados são iniciados com o mesmo usuário do servidor.

Cada vez que o InteGrade fosse executado em um nó, seriam necessários privilégios de administrador, limitando o número de pessoas capazes de executá-lo ou até mesmo inviabilizando sua execução. Rodar aplicativos nos nós como usuário super-privilegiado é uma falha grave de segurança.

Mais detalhes sobre o motivo desse comportamento serão dados ao resolver este problema na Seção 3.1.3.

Ausência de Informações do Estado de Execução. A interface para o *Node Control Center* precisa, de alguma forma, obter informações de *status* para exibir ao usuário. No mínimo, o CPUReserve deve mostrar qual o limite atual de processamento para o usuário local decidir se deseja alterá-lo. Guardar essa informação em qualquer outro aplicativo poderia gerar inconsistências, mostrando uma configuração enquanto outra está em uso.

Saídas Padrão e de Erro de Processos no Servidor. Após receber o programa a ser executado e controlado (assim como seus argumentos) pelo cliente, o cliente é finalizado e o servidor usa *fork* e *execvp*. As saídas padrão e de erro de todos os programas saem juntas com as do servidor, não necessariamente separadas.

O usuário de uma grade InteGrade pode pedir a saída padrão e/ou de erro das execuções de seus aplicativos, mas o atual código do CPUReserve não provê essa funcionalidade.

3.1.3 Alterações

Diante dos problemas encontrados na Seção 3.1.2, seguem as alterações realizadas e quais problemas cada uma resolve.

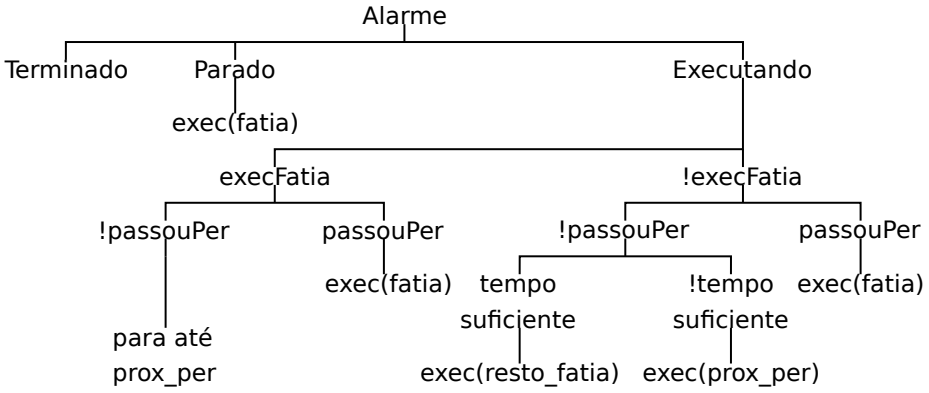

Figura 3: Nova árvore de decisões, após o desuso de *SCHED_RR* na Figura 2.

Escalonamento. Duas classes de escalonamento são utilizadas pelo CPUReserve em sistemas operacionais com Linux: *SCHED_OTHER* e *SCHED_RR*. Os processos convencionais são do tipo *SCHED_OTHER*, com tempo compartilhado. Já processos de tempo-real pertencem à classe *SCHED_RR* e sempre têm prioridade sobre os do tipo *SCHED_OTHER*.

O CPUReserve utiliza *SCHED_RR* de forma a garantir que os processos executarão toda a fatia fornecida se assim desejarem. A prioridade do servidor é máxima e a dos processos controlados, uma imediatamente abaixo do servidor até que executem todas as suas fatias. Após consumi-las, caso o parâmetro *conservação* esteja ativado (1) e ainda haja CPU disponível, o processo será alterado para *SCHED_OTHER* com prioridade mínima, caso contrário será congelado. Essas alterações podem ser realizadas quando um alarme é disparado conforme a árvore de decisões da Figura 2.

Como somente *root* ou outro usuário com privilégios de administrador pode utilizar *SCHED_RR* com a prioridade mais alta, o servidor necessita ser executado como superusuário.

Ao rodar com prioridade maior do que qualquer processo de usuário comum, as aplicações da grade atrapalham outras de grande importância para a interação entre o computador e o dono dos recursos, como aquelas responsáveis pela movimentação do mouse, atualização de janelas e por componentes gráficos em geral. Isto explica por que ocorre o problema dos travamentos periódicos citado em 3.1.2.

Se *SCHED_RR* não fosse mais utilizado, o servidor e os clientes poderiam ser executados como usuário comum e o problema dos travamentos também seria resolvido ao diminuir a prioridade das aplicações controladas. Por outro lado, a execução de toda a fatia especificada não seria mais garantida, uma vez que programas do usuário teriam maior prioridade. Mas é exatamente isso que desejamos em uma grade computacional oportunista e o uso de *SCHED_RR* foi removido, obtendo a árvore de decisão da Figura 3.

Todos os processos controlados agora são sempre do tipo *SCHED_OTHER* e possuem prioridade mínima para não atrapalhar o usuário local. Já o servidor deve ser executado com a maior prioridade possível, pois quanto maior ela for, menor a probabilidade dos processos que ele controla ultrapassarem o limite de fatias. O servidor consome pouca CPU e, por isso, não atrapalha o usuário mesmo quando executado com altas prioridades. Testes em diferentes distribuições mostraram que pode ser possível um usuário comum utilizar *SCHED_RR* (porém com prioridade máxima reduzida) e, nestes casos,

seria melhor aproveitar essa possibilidade. Ao iniciar, então, o servidor tenta alterar sua própria prioridade, da maior para a menor, parando no primeiro sucesso.

O parâmetro de conservação passado ao cliente deixa de fazer sentido, uma vez que não há mais trocas entre classes de escalonamento. Como podemos ver na Figura 3, sempre que uma aplicação consome suas fatias e o período ainda não terminou, ela é congelada até o início do próximo período.

Portanto, ao remover a utilização de *SCHED_RR*, resolvemos dois problemas encontrados:

1. Travamentos periódicos;
2. Servidor e processos executados como root.

Podemos também citar como vantagem a simplificação do código e do uso da ferramenta (não há mais a opção de conservação ao usar o cliente).

Cálculo de Fatias. Devido ao fato de não existir um limite de processamento global, à impossibilidade de atualizar os parâmetros do servidor e dos processos da grade, e ao problema da subutilização de múltiplos núcleos (citados em 3.1.2), decidiu-se escrever um novo algoritmo para distribuir fatias entre os processos controlados.

A limitação de CPU será especificada em outro programa (interface para o NCC), mas, se o CPUReserve receber as preferências do usuário no mesmo formato em que ele as fornece, os dados não precisarão ser tratados e alterados numa etapa intermediária entre o usuário e o CPUReserve, diminuindo a complexidade da implementação.

O ideal seria que o usuário local pudesse especificar uma porcentagem máxima de CPU a ser utilizada, de 0 a 100%, e que ela fosse respeitada independente de quantidade, períodos e fatias de aplicações. Usuários menos leigos também poderiam especificar quantos e quais núcleos a grade poderia usar. Algumas pessoas podem estar acostumadas a ver mais de 100% de uso de CPU se seus computadores não possuem apenas um núcleo, mas isto é menos intuitivo para outras, pois depende de características da máquina.

A porcentagem de 0 a 100 fornecida pelo usuário deve ser utilizada, em baixo nível, para distribuir da melhor forma possível as fatias dos núcleos ativos entre todas as *tasks* controladas. Todos os processos somados não devem ultrapassar o limite imposto e, assim como na versão original, processos com muitos filhos não devem ganhar mais fatias de maneira a prejudicar outros.

É mais rápido dissipar o calor de n núcleos com $\frac{100\%}{n}$ de uso do que o de apenas um com 100%, pois a superfície de contato com o dissipador de calor é n vezes maior. Pensando neste fato e em suas consequências (consumo de energia e ruído para refrigeração), adotou-se que o limite fornecido pelo usuário será interpretado como sendo por núcleo. Caso contrário, se a grade executasse um único processo, ele poderia consumir 100% de apenas uma CPU se o usuário especificasse, por exemplo, um limite de 25% em uma máquina com quatro núcleos, o que não seria intuitivo e causaria uma falsa impressão de erro.

Para manter a interface e configurações simples, adotou-se experimentalmente o período de 1000 ms, mas este valor pode ser facilmente alterado no código e o algoritmo desenvolvido o recebe como parâmetro para não perder a generalidade.

Tendo como 1000 o valor do período, definimos os seguintes tempos:

ideal: o máximo que a aplicação poderia ganhar sem ultrapassar o limite imposto pelo usuário (por núcleo). É calculado por:

$$\textbf{ideal} = (1000 \times \text{limite}) \times \min(\textit{tasks}, \text{núcleos})$$

disponível: subtraímos do total o tempo já fornecido para outras aplicações:

$$\textbf{disponível} = (1000 \times \text{limite}) \times \text{núcleos} - \textbf{fornecidos}$$

porAplicação: tempo disponível por aplicação:

$$\textbf{porAplicação} = \frac{\textbf{disponível}}{\text{aplicações restantes}}$$

fornecido: fatias que finalmente foram liberadas para a aplicação:

$$\textbf{fornecido} = \min(\textbf{ideal}, \textbf{porAplicação})$$

Primeiramente, ordena-se as aplicações em ordem crescente de número de tarefas (*tasks*). Em seguida, calcula-se o tempo ideal e o disponível por aplicação, fornecendo à aplicação atual o valor mínimo entre os dois. Para a próxima aplicação, atualiza-se o tempo disponível, descontando o que acabara de ser fornecido e repete-se o processo. Pelo fato de as aplicações estarem ordenadas, ao encontrar uma cujo tempo ideal é superior ao disponível por aplicação, esta e todas as seguintes receberão o último **porAplicação** calculado, sem necessidade de demais cálculos.

O trecho do código para calcular as fatias com maior complexidade é a ordenação das aplicações por tarefas, feita por seleção. O algoritmo consome tempo $O(n^2)$, porém é simples e suficientemente rápido para o número de aplicações esperado. Caso esse número aumente consideravelmente, pode-se obter maior eficiência usando, por exemplo, *merge-sort* que consome tempo $\theta(n \cdot log(n))$.

Como exemplo, deve-se distribuir fatias entre as aplicações da Tabela 1, dado que o limite imposto pelo usuário é de 60% e que podemos utilizar quatro CPUs. Aplicando o algoritmo, chegamos à Tabela 2. Notamos que a soma dos tempos fornecidos não ultrapassa o tempo limite especificado pelo usuário (neste exemplo, são iguais). É importante perceber que a divisão do tempo **fornecido** pelo número de tarefas não ultrapassa o limite de tempo por núcleo (600). Dessa forma, caso uma tarefa seja executada sempre em uma mesma CPU, ela respeitará o limite por núcleo. Finalmente, mas não menos importante, aplicações com mais tarefas não consomem mais fatias que outras, salvo ocasiões em que as outras não conseguem utilizar todo o tempo disponível por aplicação (quando **ideal** < **porAplicação**).

O algoritmo é executado sempre que uma aplicação é alterada, removida, ou tem seu número de tarefas modificado. Caso o usuário deseje alterar o limite de CPU, sua implementação permite redistribuir as fatias sem finalizar nenhum processo. Para melhor garantir seu correto funcionamento, foram escritos testes para este algoritmo.

Com este novo cálculo de fatias, três problemas descritos na Seção 3.1.2 foram resolvidos:

1. Aplicações Rejeitadas ao Atingir Limite;

Aplicação	*Tasks*
a	2
b	1
c	3

Tabela 1: Exemplo para distribuição de tempo de CPU entre aplicações.

Aplicação	*Tasks*	**ideal**	**disponível**	**porAplicação**	**fornecido**
b	1	600	2400	800	600
a	2	1200	1800	**900**	900
c	5	2400			**900**

Tabela 2: Resultado do algoritmo de distribuição de tempo aplicado na Tabela 1.

2. Impossibilidade de atualizar parâmetros;
3. Subutilização de múltiplas CPUs.

Além disso, o código do CPUReserve foi melhorado pela introdução de testes e também de códigos para executar testes em geral. A utilização da ferramenta também foi simplificada como será descrito em 4.1.

Informações sobre o Estado de Execução. A interface do NCC precisa das seguintes informações:

- Limite de processamento atual;
- Número total de núcleos;
- Quais são os núcleos ativos (que a grade pode utilizar);
- Processos controlados com hierarquia (pais e filhos).

```
CPU: 1 2 40
Processes:
-10925
--10926
--10927
--10928
--10929
-8786
-8680
--8681
```

Código 2: *Status* de CPUReserve.

Foi acrescentada uma funcionalidade chamada pelo programa `adapt`. Ao executar, por exemplo, `adapt 127.0.0.1:8000 status`, uma resposta válida seria a do código

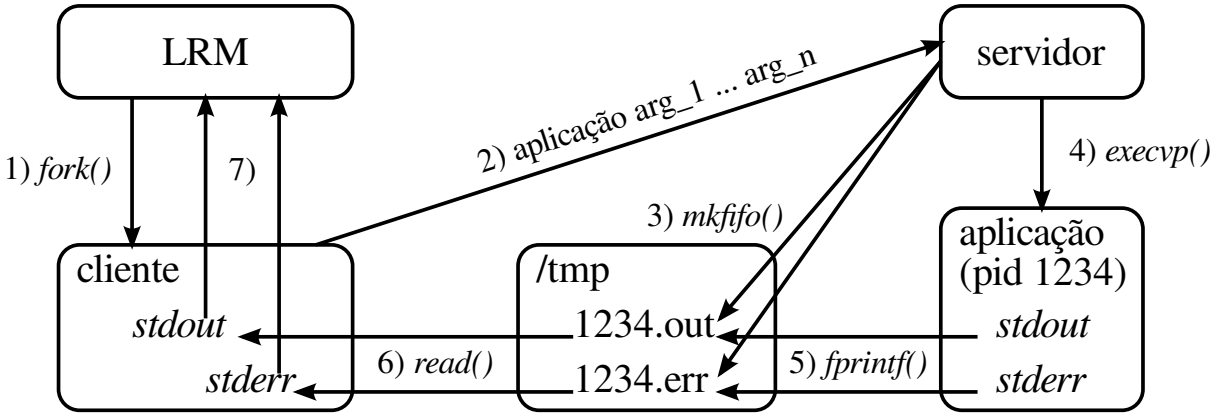

Figura 4: LRM obtendo as saídas padrão e de erro de aplicações da grade

2. Na primeira linha, notamos por `1` que apenas o núcleo 0 pode ser utilizado pela grade (representação decimal de uma máscara de bits); `2` é o número total de CPUs no computador; `40` é o limite de processamento (podendo chegar até 100). Os processos controlados são lidos da seguinte forma: os números abaixo de `Processes:` são *pids*, sendo que os pais são iniciados com apenas um hífen e os filhos, com dois. Dado um filho, seu pai é o primeiro a ser encontrado ao percorrer a lista para cima. No exemplo, o processo de ID 10925 possui quatro filhos e o 8786, nenhum. Uma observação importante é que, no CPUReserve, há apenas pais e filhos diretos. Filhos de filhos são considerados filhos da raiz.

Saídas Padrão e de Erro. Ao contrário do que acontece na versão original (problema descrito em 3.1.2), as saídas padrão (*stdout*) e de erro (*stderr*) de cada programa devem ser separadas umas das outras, pois o InteGrade (LRM) deve continuar sendo capaz de devolver os dados escritos em *stdout* e/ou *stderr* de cada aplicação para o usuário da grade.

Pelo menos duas soluções poderiam resolver o problema. A primeira consiste em o próprio cliente realizar o *fork* e as saídas serem passadas por *pipe*. A segunda alternativa seria a utilização de *named pipes*, ou seja, a comunicação seria feita através de escrita e leitura em arquivos (no caso de sistemas operacionais da família Unix).

Entre as modificações a serem feitas ao adotar a solução por *pipes*, destacam-se:

- A troca de mensagens entre servidor e cliente deveria ser alterada para apenas informar o *pid*.
- O código da *thread* que monitora os processos controlados (término e criação de filhos) não poderia apenas tratar a árvore de processos do servidor, pois os processos não mais seriam executados por ele.

Levando em consideração a maior complexidade dessas alterações, optou-se por utilizar a outra solução: *named pipes*.

Como podemos ver na Figura 4, o LRM passa ao cliente do CPUReserve o programa a ser exeutado e seus argumentos. O cliente então repassa essas informações ao servidor que cria *named pipes* antes de executar o programa. As saídas *stdout* e *stderr* da aplicação são escritas nos *pipes* nomeados e estes são lidos pelo cliente que repassa seus conteúdos

por suas respectivas saídas ao LRM. A leitura dos *named pipes* é feita de maneira não bloqueante para que o LRM obtenha os dados na medida em que eles são gerados.

Com a alteração, o cliente passa a ter o mesmo comportamento que o programa chamado sob o ponto de vista do LRM, resolvendo mais um problema.

3.2 Interface para o *Node Control Center*

Giulian Dalton Luz e Rogério Guaraci Santos desenvolveram uma interface para o NCC na disciplina Tópicos (Avançados) de Programação Orientada a Objetos (MAC0413/5715) em 2003, no IME. Vejamos seu funcionamento, os problemas encontrados e como foram resolvidos.

3.2.1 Versão Inicial

. Escrita em C++, a solução é um mini-servidor Web leve (ocupa menos de 2 MB de RAM) e multiplataforma por utilizar APIs que estão disponíveis no sistemas operacionais mais utilizados.

Suas funcionalidades são poucas, mas suficientes. O programa espera por requisições em uma porta e responde carregando arquivos, trocando ocorrências de variáveis escritas de forma padronizada (entre `<@` e `>`) por valores desejados e devolvendo o resultado. Um exemplo de arquivo está no Código 3.

```
<html>
    <body >
        <b><@MENSAGEM></b>
    </body>
</html>
```

Código 3: Exemplo de modelo HTML para a interface do NCC.

O servidor escuta somente no endereço 127.0.0.1 por segurança, mantendo-se acessível somente pela máquina em que é executado. A porta para conexões é passada como único argumento ao iniciá-lo. Através do endereço `http://127.0.0.1:8080/`, o usuário tem acesso à interface do NCC iniciado na porta 8080.

3.2.2 Problemas Encontrados

No arquivo `ncc.cpp`, encontra-se a função `getNCC(const char *sFileName, string &sData)`, responsável por todas as trocas de variáveis em todos os arquivos. Ela chama funções como `getProcessList()` e `getScheduleList()` (também em `ncc.cpp`) para obter valores a serem colocados nas páginas. Um trecho de `getNCC()` pode ser visto no código 4.

Nas linhas de número 3 a 19 do Código 4, notamos que a função `getNCC` é responsável por receber os valores enviados pelo usuário ao interagir com as páginas (*query strings*).

Pelas linhas de 22 a 25, percebemos que são realizadas as substituições de todas as variáveis existentes em qualquer página acessada.

O código completo desta função possui muito mais linhas e acrescentar mais substituições para controlar a CPU prejudicaria a legibilidade, atualizações futuras e manu-

```
1 std::string getNCC(const char * sFileName, string& sData)
2 {
3  i = sData.find("=", j);
4  //reads parameters of GET method
5  while (i != string::npos) {
6   tag = sData.substr(j, i-j);
7   j = i + 1;
8   i = sData.find("&", i);
9   if (i != string::npos ) {
10    value = sData.substr(j, i-j);
11    j = i + 1;
12   }
13   else {
14    value = sData.substr(j);
15    j = sData.size();
16   }
17   i = sData.find("=", j);
18   if (tag == "txtMemory")
19    iLimitMemory = atoi(value.c_str());
20   // ...
21
22   //gets a list of running process
23   s = replaceAll(s, "<@PROCESS>", getProcessList());
24   //gets a list of schedules
25   s = replaceAll(s, "<@SCHEDULE>", getScheduleList());
26   // ...
27  }
28 }
```

Código 4: Função encarregada de substituir variáveis em arquivos.

tenção do código. Então decidiu-se por refatorá-lo, mostrando as páginas através de um código mais orientado a objetos.

3.2.3 Alterações

Foi criada a classe `WebPage`, responsável por substituir variáveis nas páginas e facilitar o acesso a *query strings*. Classes que carregam páginas devem estendê-la. Para completar o funcionamento da página do Código 3, bastaria a classe do Código 5 e a atualização da função `getNCC` em `ncc.cpp` com as linhas do Código 6.

```
class OlaMundo : public WebPage {
    public:
        OlaMundo(string tpl): WebPage(tpl){}
    private:
        void generateVars() {
            setVar("MENSAGEM", "Olá, mundo!");
        }
};
```

Código 5: Classe que substitui variável no modelo do Código 3.

```
// ...
else if (strcmp(file,"olamundo.ncc")==0){
    OlaMundo pagina(tpl);
    tpl = pagina.getPage();
}
// ...
```

Código 6: Nova maneira de exibir uma página na função `getNCC`.

No Código 6, notamos que o modelo é passado para o construtor dos objetos que estendem `WebPage` (e, portanto, encarregados da parte visual) e depois é atualizado pela chamada à função `getPage`. Isto permite que mais de um objeto altere o modelo, sendo possível tratar diferentes partes de um modelo com objetos de classes distintas.

Para interagir com o CPUReserve, foi criada a classe `CPUReserve`. A interface com o usuário é realizada pela classe `CPUPage` que exibe informações para o usuário e também as recebe dele (`CPUPage` estende `WebPage`).

Para mostrar uma lista de aplicações e suas informações, com a possibilidade de interrompê-las, criaram-se as classes `Process` e `ProcessList`. Para visualização, foram escritas as classes `ProcessPage` e `ProcessListPage`.

Finalmente, também foram criados e modificados modelos de páginas HTML.

3.3 InteGrade (LRM)

Como visto na Seção 2.1, todos os nós que compartilham recursos executam o *Local Resource Manager* (LRM). Ele é o responsável pela execução de aplicativos da grade. Para executar um programa, o LRM utiliza as chamadas do sistema *fork* seguida de *execv*, informando o caminho do executável e seus parâmetros. Para limitar o uso de CPU, a chamada de *execv* foi modificada como na Tabela 3.

Momento	Executável	Argumentos
Antes	`gridExec`	`arg_1 arg_2 ... arg_n`
Depois	`client`	`127.0.0.1:8000 gridExec arg_1 arg_2 ... arg_n`

Tabela 3: LRM executando a aplicação *gridExec* com limite de CPU.

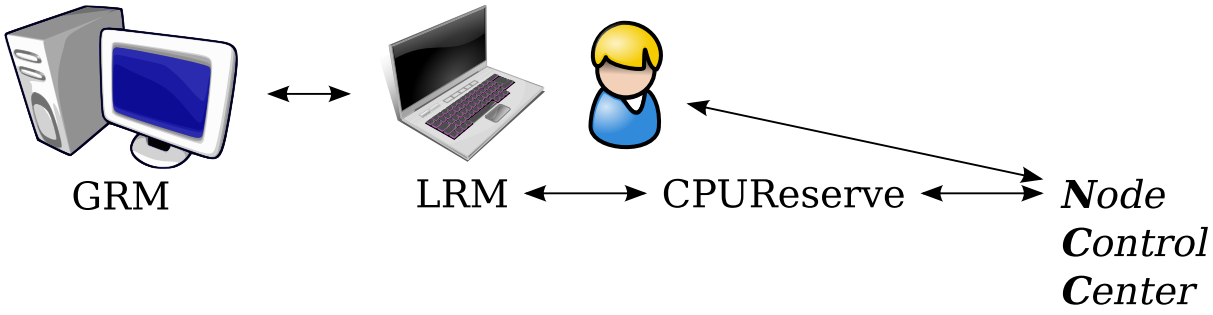

Figura 5: Relacionamento entre os programas deste trabalho.

Isto foi o suficiente para garantir o correto funcionamento do InteGrade, pois, apesar de o LRM possuir o *pid* do cliente do CPUReserve ao invés do processo "real", o cliente imprime tanto a saída padrão quanto a de erro do programa passado como argumento (`gridExec` no exemplo). Além disso, ele é terminado quando o programa chamado termina e vice-versa.

3.4 Relacionamento entre os Componentes

O relacionamento entre CPUReserve, *Node Control Center*, LRM e usuário local pode ser visto na Figura 5. O *Global Resource Manager* (GRM) continua executando suas funções normalmente, como explicado na Seção 2.1. Ao receber uma aplicação para ser executada, o LRM a executa através do CPUReserve que, por sua vez, respeita o limite de recursos imposto pelo usuário a qualquer momento através da interface do NCC.

3.5 Avaliação Experimental de Desempenho

Quem compartilha recursos em uma grade computacional oportunista não pode perceber queda de desempenho em seu computador. Foram realizados experimentos para certificar de que isso não aconteceria com a execução do LRM, CPUReserve, *Node Control Center* e aplicações da grade.

Os experimentos foram realizados em um notebook da Semp Toshiba Informática, modelo AS1560G, com 3 GB de RAM e processador AMD Turion 64 x2 de 1.6 GHz. A distribuição Linux utilizada foi a Archlinux 64 bits, com núcleo Linux versão 2.6.31.6.

3.5.1 Programas Comumente Utilizados

Visando simular as aplicações mais conhecidas e utilizadas, mediu-se o tempo de 10 inicializações de um navegador Web e um programa de escritório para cada uma das três situações abaixo.

1. InteGrade completamente desativado no nó;

Situação	Média	Desvio Padrão	Erro Padrão da Média
1	8,65	1,81	0,43
2	8,43	0,89	0,30
3	8,17	2,43	0,49

Tabela 4: Tempo para carregar o navegador Web Firefox.

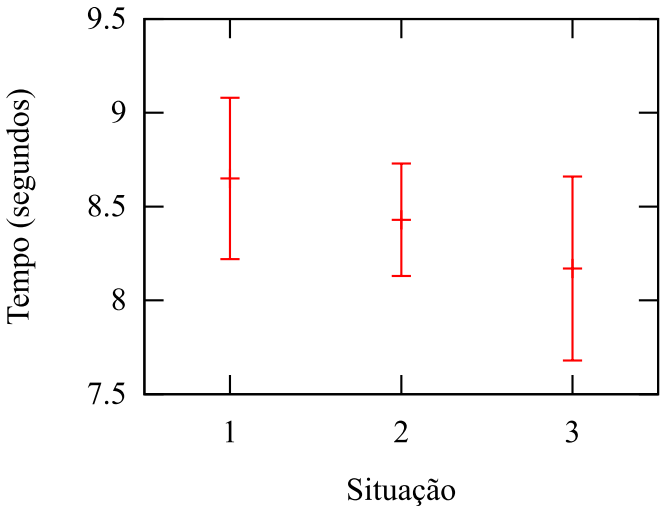

Figura 6: Tempo para carregar o Firefox.

2. InteGrade funcionando com uma aplicação com um processo e outra com dois. Limite de 20% de CPU;

3. Idem à anterior, mas com limite de 90%.

Entre cada inicialização, *buffers* e caches da memória RAM foram esvaziados para reproduzir o primeiro carregamento (mais demorado) dos programas após a inicialização do computador. A mudança de frequência dos núcleos foi desabilitada, mantendo-se a máxima frequência durante todos os experimentos.

Navegador Web. Utilizou-se o navegador Firefox 3.5.5. Para não ter influência humana nos resultados, o experimento foi realizado através do carregamento de uma página escrita em PHP, em um servidor Web local, que exibe a hora corrente com precisão de microssegundos. Antes de chamar o navegador, passando como parâmetro esta página, a mesma página é executada pelo interpretador PHP em linha de comando. A diferença entre a hora exibida no navegador e a da linha de comando corresponde ao tempo que o Firefox levou para iniciar e exibir a página. A Tabela 4 mostra o resultado dos experimentos em cada situação e a Figura 6 exibe graficamente os resultados.

Programa de Escritório. Foi medido também o tempo de inicialização do OpenOffice.org Writer, versão 3.1.1. Primeiro imprimimos a hora atual com precisão de microssegundos através do script em PHP rodado em um terminal (como no experimento do navegador Web em 3.5.1). Em seguida, carregamos o programa passando como argumento um arquivo que executa uma macro ao carregar. A macro, escrita em *python*,

Situação	Média	Desvio Padrão	Erro Padrão da Média
1	18,28	2,24	0,47
2	18,72	0,47	0,22
3	18,86	0,81	0,29

Tabela 5: Tempo para carregar o OpenOffice Writer.

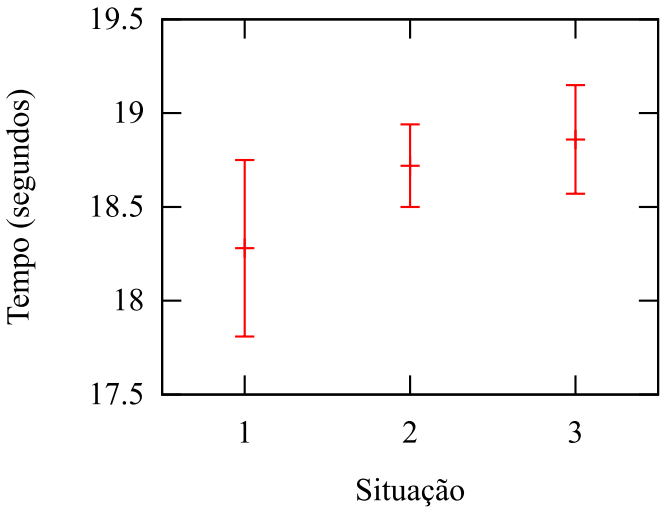

Figura 7: Tempo para carregar o OpenOffice Writer.

exibe a hora atual da mesma maneira que o script PHP. As diferenças entre os valores compõem a Tabela 5 e geram o gráfico da Figura 7.

Conclusão. Supondo uma distribuição normal dos resultados, podemos realizar um teste de hipótese sobre a média de uma população com variância conhecida [BM06] para verificar se a execução do InteGrade interfere no desempenho do nó.

Indiquemos por X o tempo de carregamento dos programas, μ a média e σ^2 a variância. $X \sim N(\mu, \sigma^2)$. As hipóteses são:

$$H_0 : \mu_2 = \mu_3 = \mu_1$$
$$H_1 : \mu_2 \neq \mu_1 \text{ ou } \mu_3 \neq \mu_1$$

Após calcular a região crítica de ambos os programas com nível de significância de 1%, H_0 não é rejeitada, ou seja, o funcionamento do InteGrade não prejudicou o desempenho do nó.

O resultado está de acordo com o esperado, pois dificilmente utilizamos todo o poder de processamento disponível. As atividades mais lentas, como as testadas, têm como fator limitante a velocidade de acesso ao disco rígido.

3.5.2 Programa com Alto Consumo de CPU

Como o recurso limitado é a CPU, também foram feitas avaliações experimentais de desempenho de um programa que utiliza ao máximo todas as CPUs disponíveis. Tanto

Situação	Média	Variância	Erro Padrão da Média
1	27,88	0,74	0,27
2	28,32	0,55	0,23
3	29,19	0,15	0,12

Tabela 6: Tempo real de uma aplicação que consome toda a CPU.

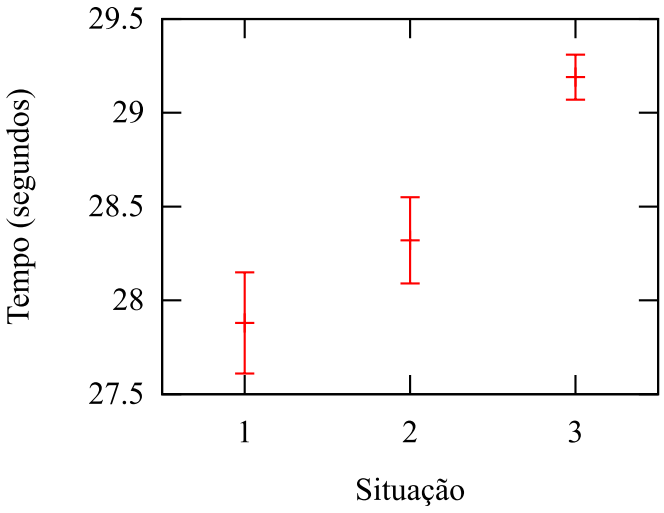

Figura 8: Médias da Tabela 6 e seus erros padrão.

o programa executado pelo usuário local quanto pela grade foram os mesmos nestas simulações, com a diferença que o da grade não tinha limite para terminar. As situações criadas foram:

- Nenhum programa do InteGrade sendo executado;
- Programa da grade executado sem limite de CPU;
- Aplicação da grade com limite de 30%.

A aplicação da grade sempre foi executada com prioridade mínima. Já o servidor do CPUReserve possuía a prioridade padrão de um usuário comum (*nice* 0). O tempo real da aplicação do usuário foi medido 10 vezes em cada situação e os dados estão disponíveis na Tabela 6, assim como na Figura 8.

Conclusão. De acordo com um teste de hipótese com um nível de significância de 1%, como aplicado em 3.5.1, não rejeitamos que $\mu_1 = \mu_2$, ou seja, o desempenho não diminuiu da situação um para a dois. Porém, hipóteses baseadas na igualdade entre a média da situação três e qualquer uma das outras médias são rejeitadas, mostrando que há uma queda de desempenho ao utilizar a CPU quando limitamos seu uso ao invés de simplesmente executar a aplicação da grade sem limite algum. Comparar a Figura 8 com as de números 6 e 7 ajuda-nos a notar que, entre os três experimentos, esse é o que possui a maior diferença positiva entre a situação três e as demais.

3.5.3 Conclusão

Quando o usuário local utiliza toda a CPU, o tempo de sua aplicação pode ser prejudicado. Contudo, os experimentos da Seção 3.5.2 são preliminares e não conclusivos. Experimentos mais bem planejados e detalhados precisariam ser realizados para avaliar melhor o impacto da solução adotada em relação a outras alternativas.

Como as aplicações que mais utilizamos dificilmente requerem muita CPU, o uso do InteGrade com limitação de CPU não prejudica o desempenho percebido pelo usuário local, como mostrado na Seção 3.5.1.

4 Resultados e Produtos Obtidos

Após as atividades, segue a descrição do funcionamento das novas versões dos programas.

4.1 CPUReserve

Inicialização do Servidor. `server 8000 60`. Não é mais necessário informar quais núcleos executarão aplicações pedidas pelo cliente. Inicialmente utilizam-se todas as CPUs e depois o usuário pode inutilizar alguma(s) se desejar pela interface ao *Node Control Center*. O último argumento agora varia de 0 a 100 ao invés de 0 a 1, por ser mais intuitivo, correspondendo ao que o usuário especifica na interface para o NCC.

Uso do Cliente para Iniciar Programas `client 127.0.0.1:8000 exec arg1 arg2 ...`. Não há mais os argumentos especificando período, fatia, nem conservação. Como já citado, o período passou a ser um segundo e a conservação deixou de fazer sentido após se alterar o escalonamento dos processos (detalhado em 3.1.3). A fatia não precisa ser mais especificada, pois o servidor determinará a máxima possível de forma a respeitar os limites impostos pelo usuário.

Alterando o Limite de CPU. `adapt 127.0.0.1:8000 1 50`. Aqui há uma grande mudança. Os dois últimos argumentos possuem o mesmo significado que os dois últimos do comando `server`. Após sua execução, será como se o servidor tivesse sido iniciado com esses dois últimos argumentos. Todas as fatias serão recalculadas de forma a respeitar os novos limites de acordo com o algoritmo descrito em 3.1.3.

Após as alterações, não podemos mais usar diferentes períodos e fatias entre as aplicações. Mas esse ajuste fino não seria de grande utilidade para o usuário compartilhador de recursos que terá acesso a configurações simples e intuitivas, porém poderosas e úteis.

4.2 NCC

O código da exibição de páginas ficou mais organizado e legível. Assim foi mais fácil implementar as novas funcionalidades.

A maneira mais simples de limitar o uso de CPU é através do semáforo mostrado na Figura 9. Ao clicar na luz vermelha, passa a ser compartilhada uma mínima parte dos

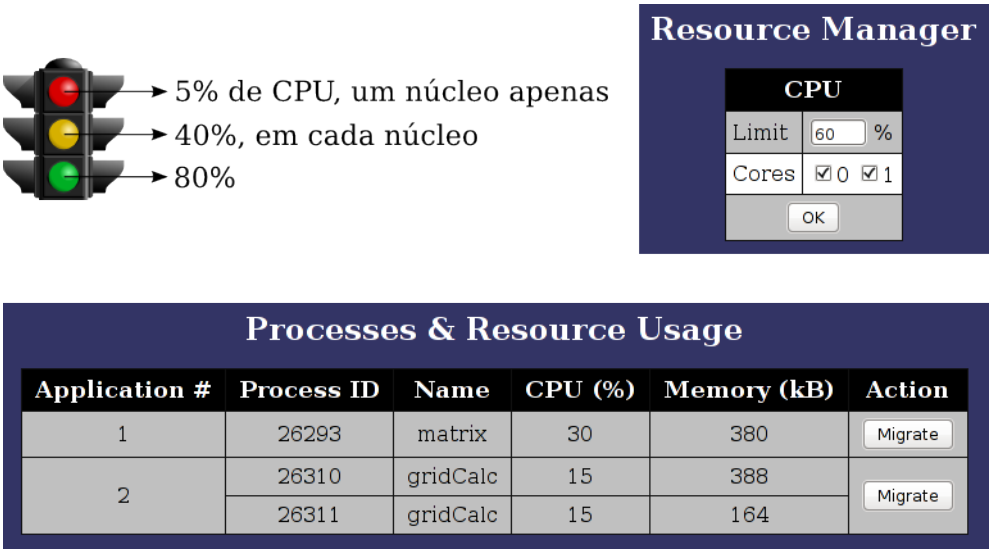

Figura 9: Acima: semáforo com configurações pré-definidas à esquerda e configuração mais detalhada de compartilhamento de CPUs. Abaixo, tabela de processos e recursos consumidos.

recursos e, ao clicar na verde, uma boa parte dos recursos é liberada para a grade. A luz amarela indica uma configuração intermediária. O quanto exatamente é compartilhado em cada um dos três casos é pré-determinado no código.

Para usuários que queiram especificar com mais detalhes o quanto e quais de sua(s) CPU(s) desejam compartilhar, há uma página cuja parte relevante é mostrada na Figura 9. Nela, pode-se especificar a porcentagem e selecionar exatamente qual(is) núcleo(s) será(ão) compartilhado(s).

Também está disponível para o usuário, em outra página, uma lista de processos da grade e o quanto dos recursos estão consumindo. Um exemplo é mostrado na figura 9. Na coluna *Action*, o botão *Migrate* termina a aplicação, podendo migrá-la para outro nó, no qual ela continuará sua execução [LdSeS05].

4.3 Produtos

Os programas CPUReserve e NCC foram entregues junto com o trabalho. Eles podem funcionar sem o InteGrade e são suficientes para rapidamente testar a limitação de CPU e a interface do *Node Control Center*. O funcionamento de ambos foi explicado no trabalho, mas instruções também estão inclusas no pacote entregue.

Os arquivos usados para testes e experimentos também foram entregues. Já o InteGrade não está incluso por ocupar muito espaço e, ao mesmo tempo, possuir poucas alterações realizadas por este trabalho.

Atualizações podem ser encontradas em:

- `https://launchpad.net/cpureserve`
- `https://launchpad.net/ncc`
- `https://launchpad.net/integrade-core` (atualmente no *branch* ncc e, em breve, no *trunk*) e `http://www.integrade.org.br/software`

As licenças do InteGrade e da interface para o NCC são LGPL[7] e a do CPUReserve é MIT[8].

5 Conclusões

Do ponto de vista do usuário, a limitação de CPU é uma nova e importante funcionalidade do InteGrade. O dono dos recursos, de maneira indireta, pode controlar o quanto de sua energia elétrica a grade pode consumir e o quanto de calor e ruído ela pode gerar. Os programas acrescentados são leves e eficientes, não prejudicando o uso do computador pelo usuário local. Por esses motivos, o número de nós voluntários pode aumentar, assim como o poder computacional da grade.

Já em relação ao InteGrade, a interface para o NCC está melhor organizada e, dessa forma, limitações de outros recursos serão mais facilmente implementadas. Iniciou-se uma separação de projetos (os códigos do NCC e CPUReserve estão em diferentes repositórios) que tornará mais prática a atualização do código do Projeto InteGrade.

Algo particular desse trabalho no Projeto InteGrade é o fato de que os programas CPUReserve e NCC podem ser utilizados sem a instalação da grade computacional. Sendo assim, um número maior de pessoas podem utilizá-los e contribuir para o aprimoramento dos aplicativos.

Referências

[BM06] Bussab, Wilton O. e Pedro A. Morettin: *Estatística Básica*. Editora Saraiva, 5ª edição, 2006.

[Cor08] Coraini, Thiago Henrique: *Escalonamento de aplicações utilizando análise de padrões de uso no InteGrade*. `http://www.ime.usp.br/~cef/mac499-08/monografias/thiago/`, 2008. Trabalho de Conclusão de Curso, IME - USP.

[GKG+04] Goldchleger, Andrei, Fabio Kon, Alfredo Goldman, Marcelo Finger e Germano Capistrano Bezerra: *InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines*. Concurrency and Computation: Practice and Experience, 16(5):449–459, March 2004.

[Gom09] Gomes, Raphael de Aquino: *Grades Computacionais Oportunistas: Alternativas para Melhorar o Desempenho das Aplicações*. Dissertação de Mestrado, Universidade Federal de Goiás, 2009.

[LdSeS05] Lopes, Rafael Fernandes e Francisco José da Silva e Silva: *Strong Migration in a Grid based on Mobile Agents*. WSEAS Transactions On Systems, Greece, 4(10):1687–1694, October 2005. ISSN 1109-2777.

[LS03] Luz, Giulian Dalton e Rogério Guaraci Santos: *Interface para o NCC do InteGrade*. Documento interno, IME/USP, 2003.

[7] `http://www.gnu.org/copyleft/lesser.html`

[8] `http://www.opensource.org/licenses/mit-license.php`

[RC08] Reis, Valéria Q. e Renato F. G. Cerqueira: *A tool for isolating performance in general-purpose operating systems.* In *Middleware Conference. Proceedings of the 6th international workshop on Middleware for grid computing*, 2008.

[SN05] Smith, J.E. e Ravi Nair: *The architecture of virtual machines.* Computer, 38(5):32–28, 2005.

Parte II
Parte Subjetiva

Relacionarei, a seguir, a experiência adquirida neste trabalho com o curso de Bacharelado em Ciência da Computação do Instituto de Matemática e Estatística da Universidade de São Paulo.

6 Desafios e Frustrações Encontrados

Trabalhar no Projeto InteGrade é um desafio. São inúmeros artigos em periódicos, conferências, *workshops*, dissertações de mestrado e doutorado, além de minicursos, capítulos em livros e relatórios técnicos [9]. Aprender sobre o projeto exige muita leitura e dedicação, que são recompensadas pelo aprendizado de tecnologias importantes, como CORBA, que permite a interoperabilidade entre aplicações de diferentes linguagens em arquiteturas distintas, sistemas operacionais e redes.

CORBA, MPI, BSD, etc. unem-se no Projeto InteGrade para agilizar ou até mesmo viabilizar aplicações que consomem muitos recursos, sem impor nenhuma forma própria de comunicação entre processos de diferentes nós, mas apoiando-se em padrões bem estabelecidos. Esse objetivo estimulou-me a progredir na curva de aprendizado.

Contudo, o trabalho parecia mais simples, ou melhor, menos difícil inicialmente do que foi. A impressão era de que bastaria integrar três programas (CPUReserve, interface para o NCC e InteGrade LRM) em cada nó, com a comunicação sendo feita basicamente através da execução dos aplicativos, em especial `server`, `client` e `adapt` do CPUReserve. No decorrer do trabalho descobriu-se problemas que fugiam dessa integração (detalhados na Seção 3.1.2). Apesar de o CPUReserve fazer parte do Projeto InteGrade, o seu objetivo é de não só limitar, mas, também, de garantir a execução das fatias pelos aplicativos em detrimento dos demais processos não gerenciados por ele, incluindo os do usuário local, o que não é admissível em uma grade computacional oportunista.

Os problemas não ligados à integração não eram esperados, mas não podem ser chamados de frustrações, pois foram resolvidos, assim como os demais problemas. O trabalho foi maior e consumiu mais tempo, porém fiquei satisfeito com o resultado, em especial com a distribuição das fatias em vários núcleos entre inúmeros processos e *threads*.

7 Disciplinas Relevantes e suas Aplicações

Muitas disciplinas cursadas durante o curso de Bacharelado em Ciência da Computação foram relevantes para o trabalho. Segue uma relação delas, em ordem alfabética de siglas, assim como a aplicação de seus conceitos no trabalho.

MAC0122 - Princípios de Desenvolvimento de Algoritmos. O CPUReserve utiliza suas próprias estruturas de dados, sendo eles unidos por listas ligadas cuja criação e manipulação foram vistas em MAC0122.

[9] `http://www.integrade.org.br/publications`

MAC0211 - Laboratório de Programação I. Tivemos contato com muitas ferramentas que continuamos usando durante todo o curso e também neste trabalho, como controlador de versão e LaTeX.

MAC0332 - Engenharia de Software. Em Engenharia de Software vimos a importância de uma boa modularização do código, com alta coesão e baixo acoplamento. Prestando atenção nesses ensinamentos, o cálculo de fatias para as aplicações pôde ser facilmente testado, sendo poucas de suas funções chamadas pelo restante do CPUReserve, facilitando a posterior manutenção do código.

MAC0338 - Análise de Algoritmos. Saber o consumo de tempo dos algoritmos é essencial para aplicativos como o CPUReserve, onde a eficiência é muito importante. O servidor precisa ser rápido para controlar o tempo de execução dos processos sem que eles ultrapassem o limite especificado pelo usuário.

A disciplina de Análise de Algortimos permitiu-me reconhecer que o trecho do código para calcular as fatias (escrito por completo neste trabalho) com maior complexidade é a ordenação das aplicações por tarefas, feita por seleção. O algoritmo consome tempo $O(n^2)$, porém é simples e suficientemente rápido para o número de aplicações esperado. Caso esse número aumente consideravelmente, pode-se obter maior eficiência usando, por exemplo, *merge-sort* que consome tempo $\theta(n \cdot log(n))$.

MAC0339 - Informação, Comunicação e a Sociedade do Conhecimento. A disciplina de MAC0339 possuiu avaliação semelhante à de Trabalho de Formatura Supervisionado (MAC0499, na qual este trabalho foi desenvolvido). Em ambas as matérias, tivemos de escrever um trabalho em português (não um código) e também fazer uma apresentação sobre ele. Este trabalho certamente aproveitou essa experiência anterior.

MAC0342 - Laboratório de Programação eXtrema. Na disciplina de MAC0342, tive a sorte de participar de um grupo muito bom cujo projeto era o InteGrade. Nosso cliente foi Fabio Kon, professor que passou a ser meu orientador neste trabalho. Tive a oportunidade de mexer no código do Projeto InteGrade, aprendendo sobre sua organização, instalação e seu funcionamento, tudo isso em um ambiente descontraído e produtivo. A construção de funções para testes em C e os testes para a distribuição de fatias (ambos escritos por completo neste trabalho) foram fortemente influenciados pela experiência que tive ao cursar esta matéria.

MAC0422 - Sistemas Operacionais. *fork*, *exec*, semáforos, *pipes*, *named pipes*, *sockets* etc. não foram estranhos ao ler os códigos de cada um dos programas. Todos eles utilizam algo visto em Sistemas Operacionais. Parte da solução para as saídas padrão e de erro no CPUReserve (Seção 3.1.3) foi discutida no fórum da disciplina.

MAC0431 - Introdução à Computação Paralela e Distribuída e MAC0438 - Programação Concorrente. Com o aumento do número de núcleos por processador, cursar MAC0431 e MAC0438 foi essencial para aproveitar o poder de processamento resultante da soma de todos eles. Foram úteis não somente os conceitos, como semáforos (muito utilizados no CPUReserve), mas também a forma de pensar após cursar estas

disciplinas. Um dos primeiros testes que fiz foi usar o CPUReserve em um computador com múltiplos núcleos e analisar seu comportamento, constatando problemas (Seção 3.1.2) e, posteriormente, resolvendo-os.

MAE0212 - Introdução à Probabilidade e à Estatística II. A estatística foi importante na avaliação experimental de desempenho. Através de testes de hipótese estudados em MAE0212 foi possível concluir a Seção 3.5.

Outras disciplinas. Há disciplinas não diretamente ligadas ao trabalho mas que, de certa forma, colaboraram para seu desenvolvimento. É o caso de Introdução à Computação (MAC0110) e Laboratório de Programação II (MAC0242). Nelas, tive mais contato com a linguagem Java, muito utilizada no Projeto InteGrade. Apesar de não utilizá-la neste trabalho, a leitura de códigos em Java foi importante para compreender o funcionamento do InteGrade como um todo antes de saber onde exatamente alterá-lo, e também para programar em outras partes do código durante a disciplina de Laboratório de Programação Extrema.

8 Passos para Aprimorar os Conhecimentos para esta Atividade

Para aprimorar meus conhecimentos sobre reserva de recursos, estudaria a limitação do uso de disco e da largura de rede, ambos prestes a serem publicados pela equipe do InteGrade da PUC-Rio. Poderiam ser necessárias alterações assim como o que ocorreu com a reserva de CPU.

Aprenderia sobre o funcionamento do LUPA [Cor08], responsável pela coleta de estatísticas de uso dos nós com a intenção de prever seus períodos de ociosidade para escalonar aplicações da grade. Faria com que as estatísticas levassem em consideração os limites impostos pelo usuário local através da interface ao NCC.

A Seção 3.5.2 mostrou que a limitação de CPU diminui o desempenho de aplicações do usuário que usam muito processamento. Caso eu continue atuando nesta área, faria experimentos mais detalhados para avaliar os benefícios reais da abordagem implementada em relação a simplesmente rodar as aplicações da grade com prioridade mínima.

Também seriam interessantes novas funcionalidades no NCC, como permitir a criação de uma agenda com horários, em que a máquina seria mais ou menos utilizada, além de uma ferramenta para configurar o NCC de várias máquinas simultaneamente.