Trabalho de Formatura Supervisionado

Previsão de Utilização de Recursos por Aplicações no InteGrade

Fábio Augusto Firmo

Orientador: Marcelo Finger

Instituto de Matemática e Estatística

Universidade de São Paulo

Novembro de 2009

Sumário

Parte I

Parte técnica

Capítulo 1

Introdução

O InteGrade [2] é um *middleware* para grades oportunistas desenvolvido por diversas instituições pelo Brasil. Seu objetivo é aproveitar os recursos de uma grade computacional para executar aplicações computacionalmente complexas. Uma grade computacional consiste em uma rede de computadores que compartilham seus recursos sob demanda para realizar uma determinada tarefa. Uma grade oportunista utiliza tipicamente máquinas compartilhadas, como laboratórios de organizações e empresas, e apenas utiliza seus recursos ociosos.

Em ambientes como esse, escalonar um trabalho para um nó que que será ocupado por seu usuário antes do término da computação compromete a Qualidade de Serviço oferecida ao usuário, pois apesar de executar suas aplicações em prioridade mínima, o InteGrade ainda não é capaz de limitar a utilização de recursos tomados. Além disso a vazão da grade diminui, pois o trabalho executa mais lentamente ou é cancelado. Esse cenário estimula o desenvolvimento de mecanismos que diminuam a ocorrência de tais eventos.

Uma possibilidade é prever os períodos de disponibilidade das máquinas. Para isso foi criado um módulo, chamado *Local Usage Pattern Analyzer* (LUPA) [6, 7, 10], capaz de prever por quanto tempo um nó pode oferecer uma determinada quantidade de recursos. Entretanto é necessário que o dono da aplicação forneça uma estimativa de tempo, processamento e memória necessários para que sua aplicação termine. Essa não é uma boa abordagem pois essas informações são geralmente desconhecidas, além de tornar a experiência de submissão mais complexa e demorada. Devido a esses fatores a previsão fornecida ao LUPA é geralmente muito imprecisa ou nem mesmo fornecida, levando ao subaproveitamento da funcionalidade.

Este trabalho de formatura descreve o planejamento e implementação de um um novo módulo do InteGrade capaz de realizar tais previsões de forma automática, retirando do usuário a responsabilidade de fornecer esses dados.

1.1 Objetivos

Nosso objetivo é construir um novo módulo do InteGrade para que o LUPA funcione sem qualquer entrada do usuário. Isso significa produzir uma expressão que contenha a taxa de utilização de CPU, memória e tempo mínimos para que uma aplicação termine.

Embora em um primeiro momento nos preocupamos em estudar e simular separadamente vários métodos para resolver o problema, é importante que exista uma implementação pronta e bem integrada ao projeto, e não somente resultados teóricos. É necessário também que essa implementação seja simples, de modo a minimizar o impacto que o cálculo das previsões causará nas máquinas da grade. Optamos por começar com métodos mais simples quanto possível e continuar com um processo de refinamento até encontrar um método que forneça uma previsão com uma precisão e sobrecarga aceitáveis em termos práticos.

Não está no escopo deste trabalho qualquer atividade relacionada diretamente ao escalonamento das aplicações. Essa é uma área que demandaria muito mais experimentos para obter alguma conclusão. A idéia é fornecer boas estimativas para o LUPA, que assim poderá fornecer boas estimativas ao escalonador.

A monografia é organizada da seguinte maneira: o segundo capítulo contém uma breve descrição dos trabalhos já publicados sobre o assunto, a descrição e as justificativas de todos os passos da abordagem escolhida e por fim resultados de simulações do nosso modelo feitas utilizando cargas de trabalho reais. O terceiro capítulo apresenta a arquitetura e os módulos do InteGrade e do novo módulo de previsão. As conclusões e possíveis trabalhos futuros estão apresentados no quarto capítulo.

Capítulo 2

Planejamento

2.1 Trabalhos relacionados

Estimar a duração de aplicações científicas é um desejo antigo. Foi possível, portanto, encontrar uma gama variada de trabalhos publicados sobre o assunto que ajudaram a conduzir a pesquisa.

Uma das abordagens estudadas tenta modelar o comportamento da aplicação ao longo de sua execução [9, 15]. Várias métricas, como utilização de CPU, rede, requisições de disco e mensagens MPI, são monitoradas a fim de encontrar padrões de comportamento. A partir dessa classificação é possível prever, utilizando técnicas variadas como Cadeias de Markov e teoria do caos, quando e para qual padrão a aplicação se comportará em seguida. Isso permite que o gerenciador da grade tome medidas ao longo da execução, como por exemplo migrar o trabalho para um nó mais indicado para o futuro estado.

Outros pesquisadores desenvolveram sistemas capazes de personalizar suas previsões para aplicações específicas, como por exemplo o Cactus [14], utilizado em pesquisa astrofísica.

Grande parte da literatura estudada, entretanto, concentra-se em realizar suas estimativas analisando observações passadas. A essência das abordagens é a mesma: é preciso definir um subconjunto de execuções passadas que se assemelhe à recém-submetida. Em seguida as características desse subconjunto são processadas para obter a previsão desejada. As diferenças geralmente estão nas definições de similaridades, nos mecanismos de busca de objetos similares e nas técnicas previsão aplicadas às execuções passadas.

Smith, Foster e Taylor [17] agruparam trabalhos com certas características idênticas em *templates*. Para definir quais são as características relevantes em cada submissão foram testados um algoritmo guloso e um genético. Finalmente, foi levado em conta a média ou a regressão linear do subconjunto gerado pelo *template*. Essa abordagem foi generalizada para um modelos baseados em *Case-based Reasoning* (CBR). A principal diferença é que a comparação de características deixa de ser binária, dando lugar a funções de distância, permitindo assim selecionar os pontos mais próximos para a análise.

Nassif, Nogueira et al. [16] utilizaram CBR para construir um sistema distribuído para grades. Um método mais eficiente para a busca dos vizinhos mais próximos, chamado *refined nearest neighbour algorithm* foi utilizado, apresentando bons resultados. Já em [13], o autor utilizou uma estrutura de dados mais complexa (árvore-M) para resolver o problema de desempenho da busca. Além disso são propostos algoritmos para

ajuste automático de parâmetros de busca e um sistema adaptativo que pode incorporar informações da política de escalonamento quando conveniente.

2.2 Decisões iniciais

Com base na análise da literatura pudemos definir algumas características desejáveis do nosso modelo. Um conceito fundamental desse trabalho é a simplicidade. Embora o problema seja complexo optamos por testar primeiramente as opções mais simples, testar sua eficácia, e refiná-las se necessário. Uma grande razão para essa política está na implementação, pois um modelo mais simples torna a implementação mais rápida, o código mais estável, a manutenção mais fácil e reduz o impacto no desempenho das máquinas da grade.

Outra decisão foi em relação à granularidade da análise da uma execução. Nesse contexto, chamamos de **análise local** a análise e previsão de utilização de recursos durante a execução da aplicação. Nesse caso a utilização de recursos da aplicação é resumida em uma série temporal. Em oposição, a **análise global** desconsidera em que ponto da execução os eventos ocorreram, interessando-se apenas por números representativos dessa utilização, como a média, mínimo ou máximo. Escolhemos a abordagem global por dois motivos: por sua simplicidade, e pelo pouco valor em saber em que período da execução os recursos serão utilizados, já que idealmente o nó estará ocioso a maior parte do tempo e pela dificuldade na migração de trabalhos dentro da grade.

2.3 Definindo similaridade

Nesse trabalho usamos observações passadas para encontrar uma boa previsão para o próximo trabalho. Analisar todas as execuções já realizadas não é uma boa idéia. É preciso definir um subconjunto de trabalhos com características similares às da nova aplicação. Esse é um típico problema de Aprendizado Computacional, e nessa seção apresentaremos algumas soluções propostas por outros autores, assim como os mecanismos que utilizamos em nosso modelo.

Uma das primeiras preocupações é definir quais informações são relevantes para a comparação. A lista de características (ou atributos) de cada execução é extensa. Algumas das mais intuitivas são: nome da aplicação, nome do usuário, argumentos do executável, tamanho dos arquivos de entrada e número de nós. Porém há também outras menos evidentes, como horário da submissão, "idade" e tamanho do executável; e ainda aquelas particulares da grade ou *cluster* em questão, como tipo de prioridade (ou tipo de fila) e grupos de usuários.

Além disso é necessário definir, para cada característica, uma função que permita comparar duas execuções. Chamamos de **função de similaridade** uma função sim que mapeia duas características ao intervalo $[0, 1]$. $sim(a, b) = 1$ se e somente se a e b são consideradas iguais.

Essa escolha não é trivial. Tome o exemplo do argumento do executável, uma possibilidade é utilizar uma função binária, ou seja, caso as duas cadeias de caracteres sejam exatamente iguais, a similaridade é um, e zero caso contrário. Isso implica que, por

exemplo, os argumentos `--width=10 --height=20` e `--height=20 --width=10` serão considerados diferentes, embora provavelmente deveriam ser tratados como iguais.

Com isso podemos calcular a similaridade local entre dois pontos, ou seja, mensurar a semelhança de suas características. O próximo passo é encontrar a similaridade global, que é a semelhança entre os duas execuções. Para isso basta atribuir pesos aos atributos presentes no modelo. Isso pode ser representado por:

$$sim(A, B) = \sum_{i=1}^{c} \omega_i sim_i(a_i, b_i)$$

onde A e B são execuções, c é o número de características e sim_i é a similaridade local para a característica i.

A última etapa consiste em delimitar o subconjunto similar ao objeto de previsão. Uma alternativa comum é escolher um limiar adequado e encontrar todas as execuções passadas com similaridade maior ou igual. Outra possibilidade é encontrar um número adequado de execuções mais próximas ao ponto de consulta, conhecida como busca de vizinhos mais próximos.

A utilização do algoritmo dos vizinhos mais próximos acaba levando a um possível problema de desempenho, pois o tempo de busca pode se mostrar muito grande à medida que a base de dados cresce. Esse fato, além de atrasar a submissão da nova aplicação, pode representar um pico de processamento indesejado e, conseqüentemente, queda na Qualidade de Serviço, pois muitas vezes a busca será executada em um nó não dedicado. Além disso foi constatado empiricamente [17] que, assim como é possível deduzir intuitivamente, o nome da aplicação é o atributo mais relevante, com uma grande margem em relação aos demais. Esses dois motivos nos levaram a adotar uma definição de similaridade extremamente simples, considerando como similares todas as execuções realizadas da mesma aplicação.

2.4 Métodos de previsão

Assim que encontramos um conjunto de aplicações similares ao novo caso é calcular a estimativa. Para isso utilizamos algumas estatísticas simples, descritas a seguir:

- **Média**
  A média aritmética, apesar de bastante simles, foi utilizada várias vezes com bons resultados em diversos trabalhos anteriores. Além disso possui a grande vantagem de ser calculada de maneira bastante rápida.

- **Máximo**
  Utilizar o valor máximo das execuções passadas, ou o máximo mais o desvio padrão, é uma maneira garantida de conseguir estimativas conservadoras, que podem ser interessantes dependendo da política de escalonamento. Essa seria uma opção razoável quando a aplicação não apresentasse nenhuma execução anormal que eleve muito as próximas previsões, levando a grandes erros continuamente. Todas as aplicações encontradas em cargas de trabalho reais, contudo, apresentaram picos, o que inviabilizou a adoção desse método.

- **Máximo com decaimento**
  O maior problema em utilizar o máximo é fixar todas as próximas previsões para valores maiores ou iguais a ele. Isso implica que o erro causado por um ponto extremo é propagado indefinidamente. Há algumas maneiras de evitar isso, como por exemplo considerar apenas um número fixado das execuções mais recentes. Optamos por multiplicar a previsão por uma função decrescente, no caso e^{-x}, onde x é o número de previsões feitas desde o último erro. Isso significa que a cada vez que uma previsão se mostra insuficiente, ou muito próxima ao valor observado, a próxima previsão voltará ao valor máximo.

- **Intervalo de confiança**
  Pode ser considerado como um refinamento da média, pois também leva em consideração o espalhamento dos dados. Como se trata de um intervalo consideramos o valor do limitante superior. Em todas as simulações utilizamos intervalos com 95% de confiança.

- **Mediana**
  Outra método simples. É interessante por ser menos suscetível a pontos extremos que a média e o intervalo de confiança.

- **Regressão Linear**
  Não foi utilizado nas simulações por se mostrar menos preciso que a média em [17].

Um detalhe importante é que consideramos todas as execuções passadas com o mesmo peso, porém esse não é a única opção. Poderia ser possível adotar um peso à cada execução baseado em sua "idade" ou em sua distância ao ponto a ser estimado.

2.5 Tipos de recursos

O modelo apresentado não faz nenhuma suposição sobre que tipo de recurso será analisado, porém é preciso tomar alguns cuidados nesse quesito.

Uma limitação é causada pelo LUPA. Atualmente esse módulo só é capaz de analisar padrões de utilização de CPU e memória, portanto não faz sentido realizar previsões sobre outros recursos, como utilização de disco, por exemplo, embora seja possível estender os dois sistemas para isso.

Outro detalhe importante é que nem todos os recursos apresentarão necessariamente o mesmo comportamento. Nada garante que um bom modelo de previsão para o tempo de execução apresente mesmo desempenho para memória. O mesmo vale para aplicações diferentes. Pensando nisso tomamos cuidado para não fixa nenhum método de previsão, tornando possível que cada aplicação e cada recurso tenha seu próprio método, com parâmetros possivelmente personalizados.

Por fim, até agora consideramos o tempo de execução apenas como seu tempo absoluto em segundos porque, entre outros motivos, a maioria dos trabalhos realizou suas simulações desse modo, tornando a comparação mais fácil. A implementação, no entanto, levou em conta a heterogeneidade e a carga da grade. Mais informações podem ser obtidas na seção 3.2.

2.6 Pontos extremos

Pontos extremos, ou *outliers*, são as "exceções' de uma amostra. Mais formalmente, são observações numericamente distantes do resto da amostra, onde essa distância geralmente é definida em termos do desvio padrão[1]. Ao realizar previsões com base em séries históricas é interessante prestar atenção a essas ocorrências, pois um desses pontos pode acabar influenciando a previsão de maneira não desejada, já que pode não representar o comportamento normal da aplicação.

A detecção e remoção de *outliers* é uma possibilidade. Existem várias técnicas para isso; uma das mais simples é remover todas as ocorrências com distância da média maior que 3σ, onde σ é o desvio padrão. Porém no contexto desse trabalho existem alguns casos problemáticos como, por exemplo, quando as execuções mudam de comportamento abruptamente. Embora esse caso não tenha sido observado nas simulações, é razoável pensar que é possível que uma aplicação apresente níveis de utilização de recursos bem distintos em dois períodos de tempo, como ilustrados na figura 2.1. É possível que, por exemplo, a aplicação rode inicialmente com entradas pequenas para testes e ajustes, e só depois passe a ser utilizada para resolver entradas maiores.

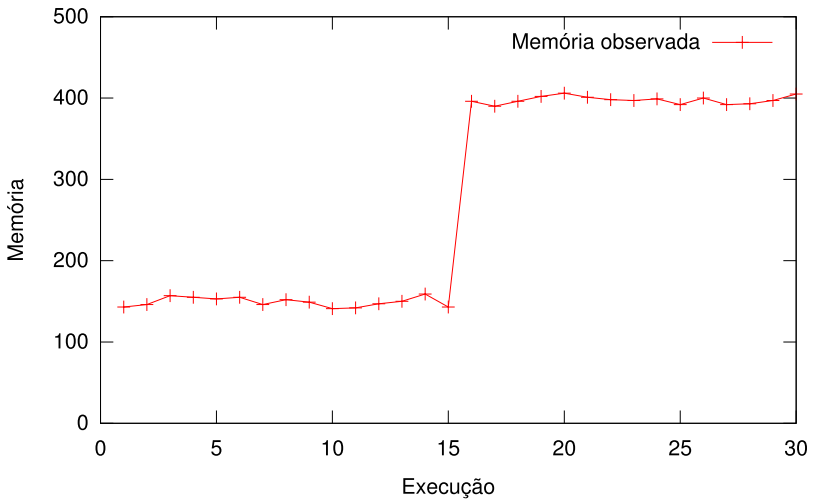

Figura 2.1: Exemplo artificial de aplicação com dois comportamentos distintos.

Essa e outras peculiaridades do tema impediram a adoção de algum tipo de remoção de pontos extremos, já que o problema demanda mais estudos antes que seja possível escolher uma boa política. Porém essa continua sendo uma questão em aberto, motivação de um possível refinamento futuro.

2.7 Simulações

Durante todo o processo de planejamento realizamos várias simulações a fim de obter informações sobre a viabilidade do modelo proposto antes de iniciar a implementação no InteGrade.

[1] Definição baseada na encontrada na Wikipedia.

A maioria das simulações feitas, e todas apresentadas nessa monografia, foram realizadas com base em cargas de trabalho reais de *clusters* obtidos através do Parallel Workload Archive [3]. Essas cargas de trabalho apresentam, para cada execução, o tempo de execução e, para alguns clusters, o máximo de memória requisitada. As cargas utilizadas foram:

Nome	Período	Execuções analisadas	Execuções totais
LANL	Outubro/1994 até Setembro/1996	4.431	201.387
DAS2-4	Janeiro/2003 até Dezembro/2003	17.447	33.795

Tabela 2.1: Cargas de trabalho utilizadas

Várias execuções foram descartadas em limpezas sugeridas pelo próprio administrador dos *logs*, outras não puderam ser utilizadas porque pertenciam a aplicações que não foram executadas em quantidade suficiente para formar uma base razoável para realizar as estimativas, tipicamente menos que 30 vezes.

Embora seja possível identificar execuções da mesma aplicação, não é possível obter mais informações sobre a aplicação, como linguagem em que foi escrita, argumentos da execução ou qual o problema que ela resolve. Não sabemos assim, por exemplo, se determinada aplicação realiza muitas operações de ponto flutuante ou se muitas consultas em um banco de dados. Como a abordagem é genérica e não faz nenhuma suposição sobre as aplicações, essa falta de informação não compromete os resultados das simulações, porém elas poderiam ser interessantes para uma analise menos especulativa dos resultados e uma melhor segurança para propostas de refinamento.

As simulações foram realizadas da seguinte forma: para cada execução de uma determinada aplicação, calcula-se uma previsão do recurso baseada estritamente nas informações passadas, que então é comparada com o valor real. Execuções que não terminaram com sucesso ou que terminaram em menos de 15 segundos foram ignoradas.

2.7.1 Métricas de precisão

Utilizamos diferentes métricas para mensurar a precisão das previsões, descritas a seguir:

- **Erro:** Valor absoluto da diferença entre o recurso estimado e o recurso observado.
- **Erro relativo:** É a razão entre a média do erro e a média do recurso observado. Um erro relativo de 0,5, por exemplo, diz que as previsões ficaram, na média, 50% maiores ou menores que o valor observado. Pode-se dizer que é a métrica mais "geral" para comparação de dois métodos, e é utilizada pela maioria dos trabalhos relacionados.
- **Desperdício:** Proporção das previsões que foram maiores que o recurso observado. Nesses casos é provável que algumas máquinas que poderiam ceder recursos não foram cogitadas pelo escalonador devido à estimativa demasiada. No sentando, como não o InteGrade não faz nenhum tipo de reserva de recursos, ao término da aplicação o nó estará apto a receber outra aplicação normalmente.

- **Estimativa insuficiente:** Proporção das previsões que foram menores que o recurso observado. Nesses casos é possível que a aplicação exceda o limite "garantido" pelo LUPA, o que pode resultar, por exemplo que a aplicações ainda não tenha terminado quando a máquina é retomada pelo usuário local.

Devido à política do InteGrade em priorizar os donos dos nós compartilhados, consideramos estimativas insuficientes mais graves que desperdícios.

2.7.2 Resultados

Ao analisar visualmente gráficos gerados pelas simulações, foi possível encontrar algumas tendências nos comportamentos das aplicações analisadas. O tempo de execução na maioria das simulações manteve-se estável em boa parte do tempo, com picos ocasionais. Esse comportamento é ilustrado pela figura 2.2, que mostra o tempo de execução da aplicação de número 3 da carga de trabalho DAS2-4.

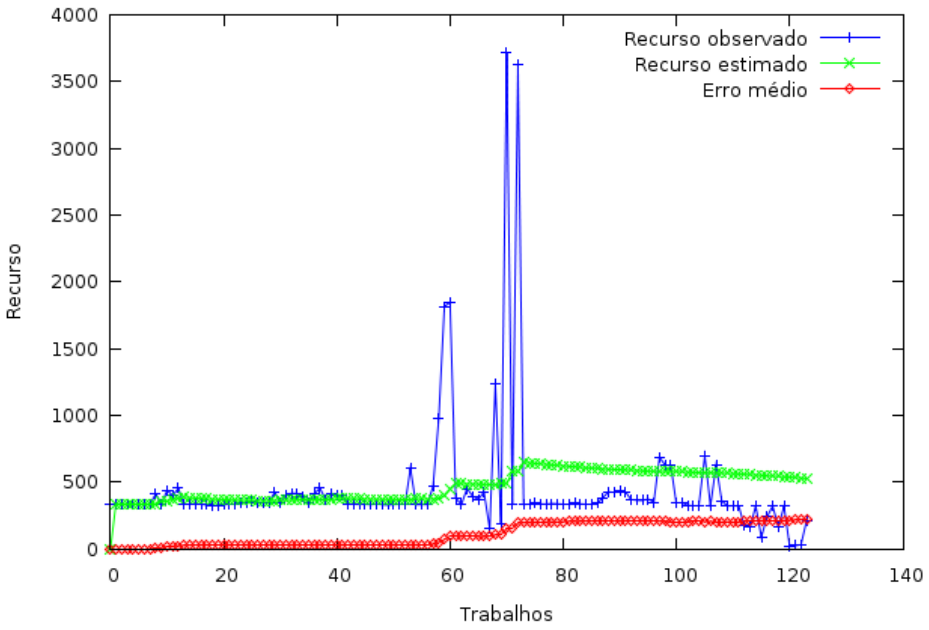

Figura 2.2: Previsão do tempo de execução de uma aplicação utilizando intervalo de confiança

Já a memória mostrou-se um pouco mais instável do que o tempo de execução, porém com menor ocorrência de grandes picos. Essa instabilidade, porém, não prejudicou a previsão. Inclusive as previsões de memória erraram menos que as do tempo de execução.

	Tempo de execução		Memória	
Cluster	Intervalo de Confiança	Mediana	Intervalo de Confiança	Mediana
LANL	1,18	0,86	0,7	0,64
DAS2-4	1,08	0,75	0,67	0,58

Tabela 2.2: Média dos erros relativos do tempo de execução e memória

Outra característica importante mostrada nos experimentos é proporção de estimativas insuficientes em relação ao total de previsões. O gráfico 2.3 mostra os dois métodos aplicados a uma mesma aplicação, onde é possível perceber que essas estimativas indesejadas são mais freqüentes utilizando a mediana, apesar do erro relativo menor.

Esse mesmo gráfico também aponta um outro resultado: o erro médio é consideravelmente menor quando a estimativa é mais alta que o necessário. Essa informação se torna mais nítida ao relembrar os picos mencionados anteriormente. Quando a aplicação se mantém estável, o erro da previsão tende a ser pequeno, porém quando ela demonstra algum pico de utilização de recurso o erro é muito maior. Esse fato aponta que o erro das previsões na verdade é menor que sua média na maioria dos casos, porém é elevado muito pelo erro causados por tais picos, de natureza mais instável e imprevisível.

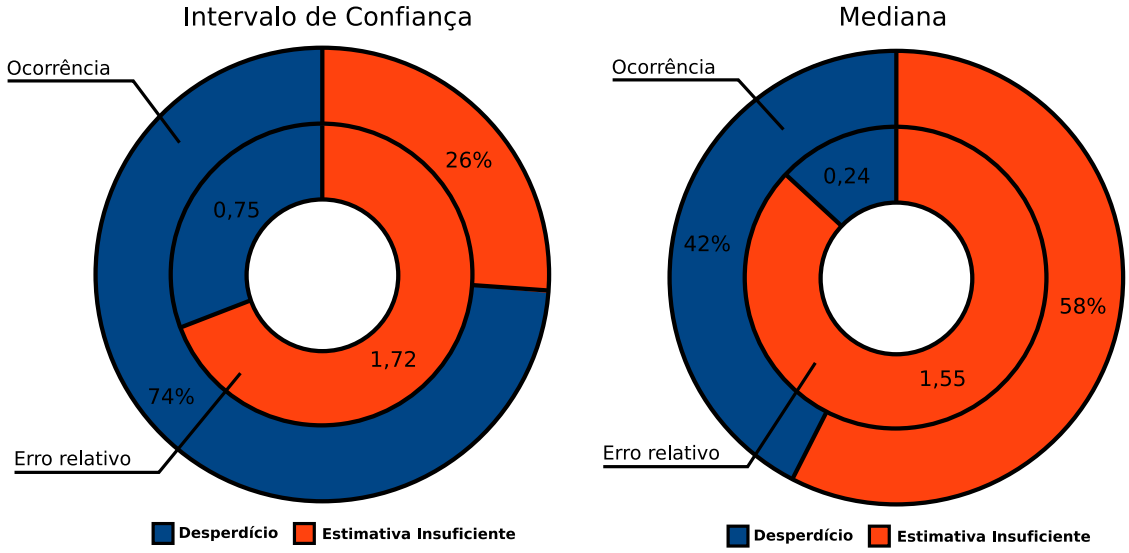

Figura 2.3: Proporção de desperdícios e estimativas insuficientes

2.7.3 Comparação com outros trabalhos

Outro autores também utilizaram simulações para mensurar a precisão de maneira muito similar ao que fizemos, possibilitando assim uma comparação. Utilizamos como métrica apenas o erro relativo, pois era a única em comum. Para todos os trabalhos, calculamos o erro relativo médio, o erro relativo no melhor caso e no pior caso. Esses dados são mostrados na tabela 2.3 e na figura 2.4.

2.7.4 Conclusões

Considerando apenas o erro relativo, as previsões feitas com a mediana apresentaram melhores resultados. Isso se deve ao fato da mediana ser menos sensível à valores extremos, que foram constatados em todas as aplicações observadas até agora. Outras métricas, entretanto, fornecem informações adicionais sobre os métodos. Em particular, foi possível notar que a quantidade de estimativas insuficientes é menor ao se utilizar intervalo de confiança.

	Erro relativo médio	Erro relativo mínimo	Erro relativo máximo
Intervalo de Confiança	1,12	0,50	1,47
Mediana	0,80	0,33	0,93
Smith, Foster, Taylor	0,49	0,39	0,58
Gibbons	0,71	0,68	0,77
Downey (Mediana)	0,88	0,58	0,99
Downey (Média)	1,46	0,61	2,04
Li	0,52	0,49	0,58

Tabela 2.3: Comparação do erro relativo com outros trabalhos da literatura

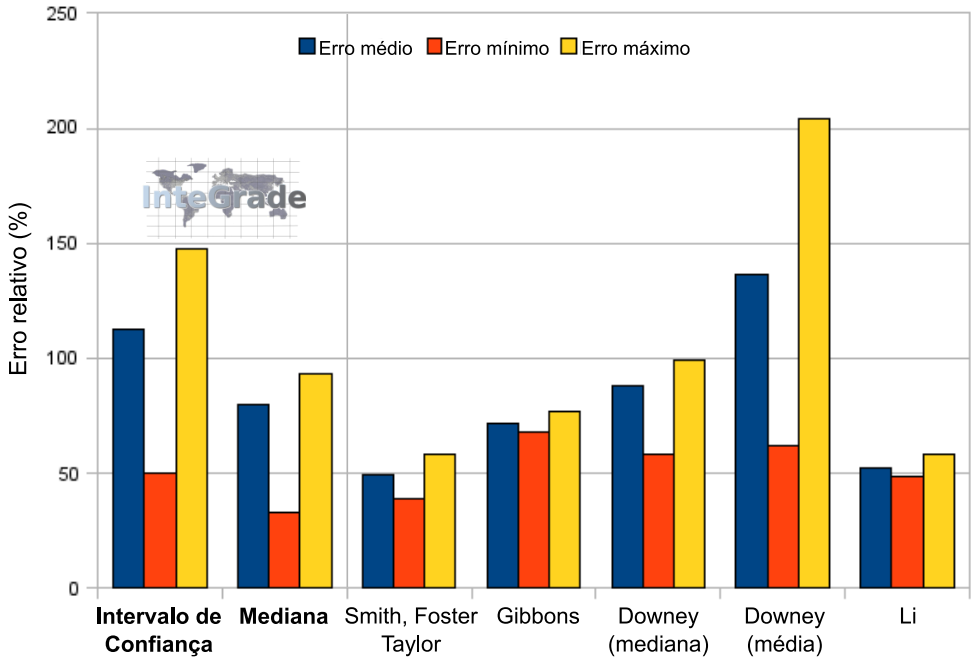

Figura 2.4: Gráfico comparativo com outros trabalhos da literatura

As simulações também mostraram que a previsão do tempo de execução e da memória apresentaram resultados similares. Isso indica que o modelo pode ser utilizado para prever diferentes recursos, embora não descartamos a possibilidade de pesquisar maneiras de explorar possíveis particularidades de recursos específicos.

Por fim, constatamos que erro médio apresentado pelo nosso modelo não está muito longe do erro de trabalhos similares.

Capítulo 3

Implementação

Era imprescindível que houvesse uma implementação em produção dessa nova funcionalidade. Esse capítulo descreve a arquitetura do InteGrade e das modificações necessárias, bem como as intenções de possíveis melhorias. Todo o código pode ser encontrado no *branch* resourcePredictor no repositório público do projeto [4]. Descrições mais detalhadas sobre o InteGrade podem ser encontradas em [11] e [12]

3.1 Arquitetura do InteGrade

O InteGrade é estruturado como uma hierarquia de aglomerados, ou *clusters*. Um aglomerado é um conjunto de máquinas, geralmente em um mesmo espaço físico ou organização, com um nó gerenciador. Todos os nós gerenciadores são organizados em uma árvore que forma toda a grade.

Cada nó de um aglomerado é classificada em relação à sua função. O **gerenciador do aglomerado** é responsável por gerenciar o seu funcionamento, escalonar as aplicações recebidas e se comunicar com outros aglomerados. O **nó de usuário** é responsável por oferecer ao usuário da grade a interface de submissão de aplicações e exibir seus resultados. Finalmente o **provedor de recursos** é o nó onde as aplicações são executadas. Ele pode ser **dedicado**, caso ceda todo seu potencial computacional à grade, mas tipicamente é **compartilhado**, quando só é aproveitado nos períodos em que o usuário local não o utiliza. É importante notar que essas categorias não são mutuamente exclusivas, um provedor de recursos pode também ser um nó de usuário, por exemplo.

3.1.1 Módulos do InteGrade

Cada uma das categorias introduzidas anteriormente apresenta um ou mais módulos característicos, descritos a seguir:

- **Global Resource Manager (GRM):** Módulo que gerencia outros provedores de recursos. É responsável por armazenar informações sobre as máquinas do aglomerado, desde características da máquina como freqüência do processador e quantidade de memória ao estado atual, como por exemplo se o nó recebeu um trabalho recentemente ou se está indisponível. Também é responsável pelo escalonamento das aplicações e pela comunicação com outros GRMs. Escrito em Java.

- **Application Repository (AR):** Módulo responsável por armazenar cópias dos binários das aplicações, bem como suas características, como a arquitetura de processador que foi compilado. Escrito em Java.

- **Local Resource Manager (LRM):** Responsável por controlar a execução da aplicação no provedor de recursos. Suas funções envolvem requisitar o binário ao AR, disparar o novo processo que executará a aplicação e notificar o término da execução. Além disso é encarregado de se registrar ao GRM e atualizar as informações necessárias periodicamente. Escrito em C++.

- **Local Usage Pattern Analyzer (LUPA):** Módulo que analisa o padrão de uso da máquina. Responde às consultas feitas pelo GRM sobre a provável disponibilidade futura de CPU e memória. Escrito em C++.

- **Application Submission and Control Tool (ASCT):** É a interface com o usuário da grade. Oferece uma interface gráfica para a submissão de aplicações e a visualização dos resultados. Também é possível cancelar trabalhos já enviados. Escrito em Java.

A comunicação dos módulos é feita em CORBA (*Common Object Request Broker Architecture*), uma tecnologia que permite que diversos componentes se comuniquem de maneira padronizada, independentemente de localização e implementação. Isso possibilita, por exemplo, que um módulo escrito em Java troque mensagens com outro escrito em C++ e que está rodando em outra máquina.

3.1.2 Funcionamento do LUPA

Como explicado na seção anterior, o LUPA é responsável por analisar os padrões de utilização dos nós da grade e responder a consultas sobre sua futura disponibilidade. Essa seção dedica-se a explicar seu funcionamento, visto que esse módulo é central para entendimento de várias decisões de implementação.

O LUPA utiliza técnicas de *clustering*, ou agrupamento, para os períodos em que a utilização dos recursos são semelhantes. Os recursos são coletados a cada cinco minutos e armazenados em um arquivo de *log*. Uma vez por dia o algoritmo de *clustering* atualiza os grupos. Assim é possível identificar, por exemplo, que uma máquina é geralmente utilizada em dias de semana pela manhã e tarde, porém permanece ociosa à noite e nos finais de semana.

Essas informações são armazenadas nas próprias máquinas, pois é razoável supor que muitos usuários não desejem que todo o histórico de utilização de seus computadores seja transmitido pela rede e armazenado em outro nó. Toda a comunicação do LUPA é feita através de consultas feitas pelo gerenciador.

As consultas, chamadas no InteGrade de *constraints*, são feitas em linguagem TCL [5]. Elas podem conter um limite de utilização de CPU, memória e tempo como, por exemplo `cpuUsage < 60 and freeRam > 10000 and hours == 2`. O valor de `cpuUsage` deve estar entre 0 e 100, e representa a taxa média de CPU livre. `freeRam` deve ser fornecido como o valor absoluto em *kilobytes*. Antigamente o valor de `hours` deveria ser um valor inteiro, porém isso prejudicaria demais as previsões. Devido a essa limitação

modificamos adaptamos o LUPA para trabalhar também com valores não inteiros no campo horas.

Em seu comportamento padrão, o LUPA devolve apenas uma resposta booleana, `true` se, de acordo com sua previsão, o nó em questão terá CPU e memória livres durante o período pedido, `false` caso contrário. Refinamentos no LUPA e escalonamento feitos recentemente [8] introduziram novos métodos à interface para, por exemplo, obter o número de horas que o nó pode garantir aquele nível de CPU e memória livres.

3.1.3 Execução de aplicações

Com a descrição dos módulos do InteGrade já é possível entender como eles se trabalham durante a submissão de uma aplicação. Esse conhecimento é importante pois a implementação da previsão apenas adiciona algumas etapas ao fluxo, que pode ser visualizado na figura 3.1[1]

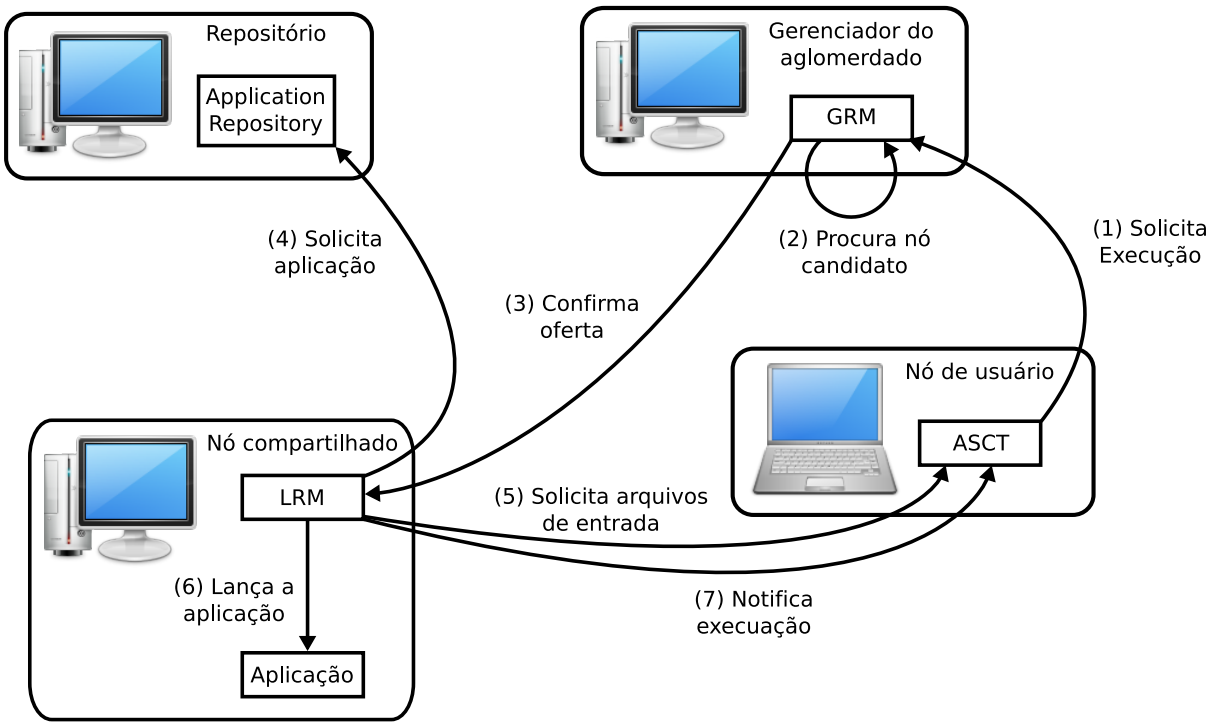

Figura 3.1: Fluxo de execução de uma aplicação no InteGrade

O processo inicia-se com a submissão da aplicação pelo usuário através do ASCT (1). Ao receber a mensagem de uma nova submissão, o GRM seleciona (2) os LRMs que satisfaçam certas condições, como por exemplo arquitetura do processador, e então verifica se o nó está disponível (3), ou seja, se o LUPA responde positivamente à consulta com as exigências de recursos, que pode ser vazia. Ao aceitar o trabalho, o LRM busca o binário da aplicação no AR (4) e os possíveis arquivos de entrada no ASCT (5), para então iniciar a execução em um novo processo (7), notificando o ASCT que a execução se iniciou (7). Ao fim da execução, as saídas da aplicação são mandadas para o ASCT.

[1]Figura baseada na encontrada em [12]. Os ícones fazem parte do tema Oxygen [http://www.oxygen-icons.org], licenciados sob a LGLPv3

3.2 Capacidade de processamento e tempo de execução

O foco principal do InteGrade está em grades heterogênias e oportunistas. Nesses ambientes é comum encontrar máquinas com diferentes processadores sob cargas de trabalho diferentes. Nessas condições é interessante encontrar um relação entre modelo, freqüência e carga de processadores com o tempo de execução das aplicações. Nossa intuição e alguns estudos dizem que há uma relação linear entre eles na maioria dos casos. Em outras palavras, um processador com o dobro de capacidade executa uma aplicação em aproximadamente metade do tempo. Muitos fatores, contudo, contribuem para tornar essa relação instável, como espera para requisições de disco e rede e velocidade do barramento. A própria definição de capacidade de um processador é difícil de ser resumida em uma escala única, pois depende de muitos fatores, como tamanho e velocidade do *cache* e otimizações de *pipeline*.

Diante da necessidade de definir uma medida de capacidade de processamento estudamos algumas opções. Uma forte alternativa são os *benchmarks* da *Standard Performance Evaluation Corporation* (SPEC), organização especializada em criar métricas para comparação de CPUs e outros componentes. Esses *benchmarks*, no entanto, demoram muito para executar e não são gratuitos, o que inviabiliza sua utilização. Outro *benchmark* mais simples encontrado foi o *Tom Kerrigan's Simple Chess Program* (TSCP), que, apesar de ser destinado ao ensino de técnicas de programação para xadrez, possui a funcionalidade de medir o poder de processamento. Uma última opção é o *BogoMips* (*Bogus million instructions per second*), que basicamente mede quantas vezes o processador executa a instrução NOP em um segundo.

Juntamente com outros desenvolvedores do InteGrade, comparamos os resultados do TSCP e BogoMips de 6 máquinas diferentes tendo como referência números produzidos pelos *benchmarks* da SPEC, obtidos através da internet. Constatamos que tanto o erro quanto erro ao quadrado do *BogoMips* foram menores que os do TSCP para aquela amostra. Aproveitando a maior facilidade de se obter os *BogoMips* de um processador, decidimos utilizar essa métrica de comparação de processadores.

O LRM já possuía uma classe, chamada `NodeStaticInformation`, que era encarregada de colher informações sobre a máquina como, por exemplo, a quantidade de memória. Essa classe foi então modificada para obter também os *BogoMips* e o número de cores do processador.

No restante da implementação há duas maneiras de representar o tempo de execução: o tempo de relógio, geralmente referenciado pelo nome `executionTime`, e o tempo normalizado, referenciado pelo nome `normalizedExecutionTime`, que é a razão entre o tempo de relógio e os *BogoMips* da máquina que rodou a aplicação.

3.3 Arquitetura da implementação

As mudanças ocorreram em diversos pontos do código, porém podem ser resumidas na criação de três novos pacotes ou conjunto de classes, explicados a seguir:

3.3.1 ApplicationHistoryDatabase

Tem como finalidade armazenar de maneira persistente as informações sobre todas as execuções passadas. É uma base de dados centralizada, localizada no Application Repository, o que implica que só há uma por aglomerado. Isso evita que todos os LRMs enviem pela rede as informações sobre as execuções anteriores a cada vez que uma nova previsão precisa ser feita. Também evita a volatilidade de informações, pois provedores de recursos antigos podem ser removidos momentaneamente ou permanentemente.

A comunicação dessa nova classe é feita pela extensão da interface CORBA do Application Repository com os seguintes métodos e tipos:

`struct ResourceUsageReport`
: Estrutura que contém informações sobre uma execução. Contém o nome da aplicação, o tempo de execução normalizado, o máximo de memória utilizada e o valor dos BogoMips do processador da máquina que executou o trabalho.

`void reportResourceUsage(string applicationName, ResourceUsageReport report)`
: Método invocado pelo LRM ao fim de uma execução. Incorpora à base de dados as informações de uma execução.

`int getNumberOfApplicationExecutions(string applicationName)`
: Recebe um nome de aplicação e devolve o número de execuções armazenadas com aquele nome.

`double getApplicationExecutionsStatistic(string applicationName, Resource resource, Statistic statistic)`
: Principal método para recuperar informações sobre uma aplicação. Devolve uma estatística de um recurso sem a necessidade de vários métodos separados na interface. As enumerações `Resource` e `Statistic` também são definidas na interface CORBA e descritas a seguir.

`enum Resource`
: Enumeração contendo os recursos monitorados. Cada recurso corresponde a uma coluna na tabela do banco de dados. Os recursos existentes até o momento são: `executionTime`, `memory` e `mips`.

`enum Statistic`
: Enumeração com todas as estatísticas implementadas em ApplicationHistoryDatabase. Atualmente são: `mean`, `median` e `standardDeviation`.

A persistência é feita através do banco de dados relacional H2 [1], que já era utilizado no InteGrade para armazenar as informações sobre os LRMs, portanto nenhuma nova dependência foi criada.

3.3.2 ResourcePredictor

Esse novo pacote do GRM agrupa as classes relativas à previsão dos recursos. Cada método de previsão, como mediana ou intervalo de confiança, é representado por uma

classe que implementa a interface `ResourcePredictor`. Essa interface contém apenas um método, o `predictNext`, que recebe um nome de aplicação e deve devolver um objeto contendo as previsões dos recursos para a nova submissão.

Além das classes responsáveis pela previsão, há a classe `ApplicationHistoryDatabase StubWrapper` que se encarrega de fazer a comunicação em CORBA com o banco de dados no Application Repository, oferecendo uma interface mais simples para os métodos de previsão. Ela segue o mesmo padrão encontrado nas classes de comunicação do ASCT.

3.3.3 ResourceMonitor

Localizada no LRM, essa nova classe trata da monitoração de recursos, atualmente CPU e memória, na máquina local. Por se tratar de um serviço que deve rodar continuamente, foi necessário criar uma nova *thread* que acorda a cada cinco minutos para coletar os dados dos processos lançados pelo InteGrade. As informações são obtidas através do programa *top*, padrão em sistemas Unix.

3.4 A previsão no fluxo de execução

A previsão ocorre no momento em que o GRM procura os nós candidatos a executar o trabalho, e somente se o usuário não fornecer uma estimativa própria. Nesse momento, a classe de previsão correspondente ao método escolhido devolve um objeto da classe `Prediction` contendo um tempo de execução normalizado e um valor de memória. Essa é uma previsão genérica, que ainda precisa ser modificada de acordo com o nó que receberá o trabalho, devido à heterogeneidade da grade, e de acordo com o formato de entrada do LUPA. Vale notar que é preciso haver um número mínimo de casos passados, pois não faz muito sentido basear-se em uma série muito curta para gerar uma previsão. Por esse motivo só realizamos previsões quando há, no mínimo, 15 execuções similares armazenadas no banco de dados.

As modificações são feitas ainda no gerenciador do aglomerado, que multiplica o tempo normalizado pelos BogoMips de cada nó candidato, obtendo uma previsão de tempo de execução em horas. A partir desse ponto o escalonamento é feito da mesma maneira, excluindo os nós que não garantiram os níveis de processamento e memória naquela janela de tempo.

Ao término do trabalho o LRM agrupa as informações da execução e se prepara enviá-las à base de dados. Isso só acontece, contudo, se a execução terminou com sucesso e em menos de 5 minutos. Esse limite foi escolhido pois também é a granularidade em que o LUPA trabalha com suas previsões (mesmo que o formato de entrada seja apenas para horas inteiras), e execuções muito curtas muitas vezes são resultados de testes e erros na aplicação, mesmo quando não devolvem nenhum código de erro.

3.5 Testes automatizados

O InteGrade é um projeto complexo, e qualquer alteração no código pode levar a regressões, muitas vezes difíceis de detectar, especialmente devido à grande variedade de combinações de configuração de módulos e da grade. Um dos métodos mais eficazes

para evitar esse problema são testes automatizados, que não foram feitos no início do projeto. Recentemente houve um esforço para cobrir pelo menos parte do código com testes de unidade, além da criação de um arcabouço de testes de aceitação, ainda em desenvolvimento.

Ciente da importância dos testes para a qualidade do código, e aproveitando a adição de uma nova funcionalidade, nos preocupamos em escrever alguns testes de unidade para cobrir algumas das modificações feitas. O resultado são testes para o banco de dados de aplicações e para as classes de previsão, presentes no diretório `test/` de seus respectivos módulos.

Capítulo 4

Conclusão

Apresentamos nesse trabalho um modelo simples dedicado a prever os recursos utilizados por aplicações em uma grade oportunista e sua implementação no InteGrade. Sua importância reside em retirar dos donos de aplicações a responsabilidade de estimar os recursos que serão utilizados por suas aplicações, fazendo com que o potencial da previsão de padrões de uso da máquina, tema de vários estudos anteriores e em andamento, seja plenamente explorado.

Dois métodos foram implementados, o mais conservador, que utiliza intervalo de confiança, pode ser particularmente interessante por apresentar menor probabilidade de ocorrência de cenários em que o usuário local sente degradação no desempenho do seu computador. Por outro lado, o outro método, que utiliza mediana, apresentou menor erro médio.

Simulações utilizando dados de *clusters* em atividade apontaram erros comparáveis ao de outros trabalhos da literatura. Comparando com o modelo mais preciso, nosso erro é no máximo 62%, com a mediana, e 128%, com intervalo de confiança, pior. Em outros casos, porém, apresentamos melhores resultados, chegando a um erro médio 71% menor. Esse é um bom resultado, especialmente porque a pretensão inicial não era obter um modelo mais preciso, mas sim criar um sistema simples, porém funcional e integrado ao resto do projeto.

Além dos resultados teóricos, a implementação representa uma grande parte desse trabalho. Um dos resultados desse trabalho é um novo *branch*, baseado na atual versão do InteGrade, com todo o sistema de previsão implementado e funcional, pronto para ser integrado *trunk* e lançado na próxima versão.

Infelizmente não foi possível realizar experimentos mais amplos utilizando essa nova versão. Isso se deve à falta de tempo e à complexidade da tarefa, pois é preciso, entre outros pequenos desafios, preparar uma grade oportunista, realizar o treinamento dos padrões de uso utilizando o LUPA, definir os algoritmos de escalonamento e encontrar um conjunto representativo de aplicações a serem executadas, possivelmente com comportamentos bem distintos de uso de processamento e memória. Além disso, a análise dos resultados deve ser feita com cuidado, pois a eficácia dos experimentos não depende apenas da previsão, mas também do LUPA, do escalonador e do trabalho em conjunto desses três sistemas.

4.1 Trabalhos futuros

O passo mais natural é realizar experimentos em conjunto com a previsão de padrão de uso das máquinas e o escalonador, pois além de ser importante para assegurar a eficácia da abordagem pode ajudar a guiar os possíveis refinamentos futuros. Outra melhoria que consideramos importante é modificar o LUPA para que seja possível fornecer valores para o campo horas que não sejam inteiros, como ocorre atualmente.

Além disso, existem várias sugestões para melhorar o trabalho apresentado, entre elas:

- Criar um sistema adaptativo, para monitorar o erro das previsões, escolher o melhor método e ajustar os parâmetros. Nos casos extremos, quando o comportamento da aplicação for muito aleatório, ainda existe a possibilidade de desligar a previsão, caso ela não esteja trazendo benefícios.

- Refinamentos na definição de similaridade de execuções. Com base nos estudos as características mais interessantes são o argumento da execução e o tamanho dos arquivos de entrada.

- Introdução de novos recursos, como uso de disco e rede. Para que isso ocorra, porém, é preciso monitorá-los e adicioná-los ao LUPA.

- Aplicar algoritmos de *clustering* ao histórico de aplicações, para detectar pontos extremos ou vários comportamentos distintos de uma aplicação.

- Criar uma biblioteca para extrair informações da máquina local. Atualmente há várias classes diferentes espalhadas no LRM e LUPA com funcionalidades parecidas, e há planos para utilizar um programa externo, chamado Ganglia para obter essas informações. Uma biblioteca evitará duplicação de código, além de proporcionar diferentes métodos de extração dessas informações de acordo com o sistema operacional, de maneira que evite alterações no código principal do InteGrade.

Parte II

Parte subjetiva

Capítulo 5

O curso

Como muitos, passei no vestibular sem aquela certeza de que era a escolha certa. Felizmente não demorou muito para perceber que de fato tinha feito a escolha certa. Apesar da grande carga de matemática e física no início do curso, as matérias MAC110 e MAC122 foram incentivos suficientes para seguir em frente pois, além de mostrar o quanto programar pode ser divertido mostraram, mesmo que de alguma maneira meio tímida, que toda aquela matemática e estatística realmente eram importantes (embora eu ainda esteja esperando para dizer o mesmo da física).

Mais adiante no curso, o principal problema passou a ser a falta de tempo, especialmente quando há outras atividades para se conciliar com o curso, como iniciação científica. Essa dificuldade na administração do tempo gerou vários períodos de pouca produtividade, especialmente nos finais de semestre. A frustração é que às vezes não pude me dedicar a certas disciplinas como gostaria, como por exemplo melhorar um EP que já funcionava ou estudar para uma prova em que não precisava de nota.

Porém mesmo com essa falta de tempo, esperada quando se faz um curso superior, pude aproveitar bastante o curso e as disciplinas. Também considero uma grande vantagem em fazer um BCC bastante teórico como esse é a facilidade em enxergar os conceitos mais facilmente, e quando isso acontece é muito mais fácil trabalhar, pois você se sente mais seguro ao manipular conceitos familiares.

5.1 Algumas disciplinas relevantes

MAC0122 – Princípios de Desenvolvimento de Algoritmos
Professor: Yoshiharu

Há vários motivos que tornam essa disciplina importante. Um deles é mostrar que não basta criar um código que funcione, mas sim que é preciso escolher os algorítmos e estruturas de dados apropriadas para a situação. Além disso foi uma matéria que sintetizou bem vários aspectos do curso e posiibilitou perceber que realmente estava no curso certo.

MAC0438 – Programação Concorrente
Professor: Alfredo

Embora disciplinas como Introdução à Computação Paralela e Distribuída e Sistemas Operacionais tenham apresentado alguns conceitos de concorrência, foi nessa disciplina

que pude compreender de maneira mais geral essa área de extrema importância para a formação de qualquer computólogo. A familiaridade adquirida foi fundamental para entender mais facilmente o código do InteGrade e para implementar a previsão e, principalmente, o monitoramento de recursos.

MAC0342 – Laboratório de Programação Extrema
Professor: Alfredo

Essa disciplina me ajudou muito em vário aspectos. O projeto ao qual fiz parte foi o próprio InteGrade, o que aprofundou meu conhecimento sobre o código, embora nenhum dos cartões tenha sido sobre o meu trabalho de formatura. Mas muito além disso, essa foi a disciplina que mais me preparou para as pressões de fazer parte de um projeto maior. Hoje consigo lidar muito melhor com pessoas, problemas, prazos e cobranças graças à disciplina. Além disso, a proximidade do grupo (e de outros grupos, durante os horários de almoço) tornaram o ambiente extremamente agradável, mesmo com todos os desafios, que não foram poucos.

MAC0422 – Sistemas Operacionais
Professor: Alan

Conhecer com detalhes o funcionamento dos sistemas operacionais me ajudou muito em elaborar a abordagem para a obtenção das métricas de recursos. De maneira mais sutil, essa foi a disciplina que em que mais ficou evidente que dificilmente existe um melhor algoritmo ou política, mas sim diversas opções, cada uma com suas vantagens e desvantagens. Mais do que isso, a eficácia dos modelos muitas vezes só podem ser comparadas atraveś de resultados de testes empíricos, que devem ser aplicados com extremo cuidado. Essa preocupação esteve presente na escolha dos métodos de classificação e previsão.

MAC0460 – Aprendizagem Computacional
Professora: Nina

A classificação de aplicações em grupos similares é um requisito da abordagem escolhida nesse trabalho. Embora eu tenha escolhido uma solução simples, que não exige nenhum conhecimento teórico da área, aprendi nessa disciplina várias possibilidades interessantes para a resolução do problema. Refinar a definição de similaridade entre aplicações é sem dúvida um dos caminhos possíveis para aprofundar meus estudos.

Capítulo 6

O trabalho de formatura

Após terminar o trabalho de curso posso afirmar: ele foi extremamente importante para minha formação. Apesar de todas as dificuldades encontradas, essa foi uma chance única para experimentar mais de perto como é realizar um trabalho de pesquisa. Acredito que essa experiência é até mais valiosa do que aprendi sobre o tema em si, pois é algo que vai influenciar minha carreira independente da área escolhida.

O tema escolhido contribuiu muito para tornar essa experiência ainda mais distinta do restante do curso. Foram poucas as vezes em que encontrei um problema onde houvessem tantas opções de resolução, além de diversas maneiras de realizar experimentos para tentar obter um *feedback* do trabalho feito. Os próprios experimentos foram trabalhosos, e boa parte da etapa de planejamento foi destinada a criar uma séries de scripts em Python para realizar os experimentos de forma automática a partir de uma entrada contendo a carga de trabalho, o método a ser utilizado e o recurso a ser analisado, entre outos parâmetros.

A implementação, particularmente, foi bastante desafiadora. Muitos dos problemas não estão relatados na monografia por não serem relacionados diretamente ao tema da monografia, como por exemplo a adição de novos métodos à interface CORBA. Invocar um método remoto por um código C++, como o método que reporta as informações sobre uma execução para o banco de dados, por utilizar uma biblioteca em Lua, chamada OiL, é necessário, a cada método chamado, manipular diretamente uma pilha com informações sobre o nome e localização do método e dos argumentos, para só depois desempilhar os resultados. Um outro exemplo é o `ResourceMonitor`, que por utilizar uma *thread* que coleta os dados periodicamente, precisa de atenção especial para evitar problemas de concorrência, que são difíceis de detectar e corrigir. Todas essas dificuldades, porém, já eram previstas dada a complexidade do código do InteGrade.

6.1 Estudos futuros

Caso continue com meu trabalho no InteGrade, provavelmente meu próximo passo seria estudar técnicas específicas para realizar experimentos que comprovem a eficiência do método em um ambiente real da melhor maneira possível. Esses estudos são necessários pois os resultados dos experimentos vão guiar os próximos esforços.

Tratando-se de refinamentos, um caminho interessante é analisar novas maneira de definir a similaridade de aplicações. O objetivo é encontrar uma definição que leve em

consideração outras características que podem ditar o comportamento da aplicação, mas que cotinue simples, para não aumentar o sobrecusto causado pelo InteGrade nas máquinas da grade. Outro tema interessante é a detecção e tratamento de pontos extremos. Inicialmente

Como o InteGrade é um projeto de código aberto, tais mudanças podem ser integradas ao resto do código de maneira bem simples, ou mesmo feitas por outros contribuidores. Além disso, pretendo continuar acompanhando as discussões nas listas de e-mail, especialmente para orientar em possíveis discussões sobre o trabalho feito por mim.

6.2 Agradecimentos

Gostaria de preencher um pequeno espaço para formalizar de uma maneira pessoal algumas das contribuições importantes. Agradeço à minha família, pelo apoio desde o início, ao Marcelo Finger pelas muitas sugestões valiosas, ao Alfredo Goldman, Fabio Kon e Raphael Camargo pelos questionamentos durantes as reuinões do projeto, aos demais alunos e professores do InteGrade, que proporcionaram as bases nas quais pude me apoiar nas pesquisas e à toda turma do BCC 2006, por ser manter unida por esses quatro anos.

Referências Bibliográficas

[1] *H2 Database Engine*. `http://www.h2database.com/html/main.html`.

[2] *InteGrade*. `http://www.integrade.org.br`.

[3] *Parallel Workloads Archive*. `http://www.cs.huji.ac.il/labs/parallel/workload`.

[4] *Repositório InteGrade*. `bzr://www.integrade.org.br/integrade/branches/resourcePredictor`.

[5] *TAO Trading Service Documentation*. `http://www.dre.vanderbilt.edu/~schmidt/DOC_ROOT/TAO/docs/releasenotes/trader.html#Constraints`.

[6] Bezerra, Germano C.: *Análise de Conglomerados Aplicada ao Reconhecimento de Padrões de Uso de Recursos Computacionais*. Tese de Mestrado, Departamento de Ciência da Computação - Universidade de São Paulo. Orientador: Marcelo Finger, Janeiro 2006.

[7] Conde, D.: *Análise de Padrões de Uso em Grades Computacionais*. Tese de Mestrado, Departamento de Ciência da Computação - Universidade de São Paulo. Orientador: Marcelo Finger, Janeiro 2008.

[8] Coraini, Thiago H.: *Escalonamento de aplicações utilizando análise de padrões de uso no InteGrade*. 2008. Trabalho de Formatura Supervisionado. Departamento de Ciência da Computação - Universidade de São Paulo. Orientador: Marcelo Finger.

[9] Dodonov, E., F. de Mello, *et al.*: *A Model for Automatic On-Line Process Behavior Extraction, Classification and Prediction in Heterogeneous Distributed Systems*. In *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, páginas 899–904. IEEE Computer Society, 2007.

[10] Finger, Marcelo, Germano C. Bezerra, e Danilo M. R. Conde: *Resource use pattern analysis for predicting resource availability in opportunistic grids*. Concurrency and Computation: Practice and Experience, Accepted, 2009.

[11] Goldchleger, A.: *Integrade: Um sistema de middleware para computação em grade oportunista*. Tese de Mestrado, Departamento de Ciência da Computação - Universidade de São Paulo, 2004.

[12] 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.

[13] Li, H.: *Machine learning for performance predictions on space-shared computing environments*. International Transactions on Systems Science and Applications, invited paper, 2007.

[14] Liu, Chuang, Lingyun Yang, Ian Foster, e Dave Angulo: *Design and Evaluation of a Resource Selection Framework for Grid Applications*. Volume 0, página 63, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[15] Mello, R.F. de e L.T. Yang: *Prediction of dynamical, nonlinear, and unstable process behavior*. The Journal of Supercomputing, 49(1):22–41, 2009.

[16] Nassif, L.N., J.M. Nogueira, A. Karmouch, M. Ahmed, e F.V. de Andrade: *Job completion prediction using case-based reasoning for grid computing environments*. Concurrency and Computation: Practice and Experience, 19(9), 2007.

[17] Smith, W., I. Foster, e V. Taylor: *Predicting application run times using historical information*. Lecture Notes in Computer Science, 1459(122ff):183, 1998.