JProber: Ferramenta para Análise de Desempenho de Aplicações Paralelas

Lucas G. Silva [1], Arthur P. Souza [2], Carlos Augusto P. S. Martins [3], Luís Fabrício W. Góes [4]
Laboratório de Sistemas Digitais e Computacionais (LSDC)
Pontifícia Universidade Católica de Minas Gerais [1, 3, 4], Faculdade COTEMIG [2, 4]
Belo Horizonte, Minas Gerais
{lgoulartsilva [1], aps_si [2], lfwgoes [4]} @yahoo.com.br, capsm@pucminas.br [3]

Resumo

Este trabalho tem o objetivo de propor e apresentar uma ferramenta desenvolvida em Java, para auxiliar a análise de desempenho de aplicações seqüenciais ou paralelas. A ferramenta, que recebe o nome de JProber, inicia a execução dos programas e executa a contagem de métricas para realização da análise. Os resultados capturados são armazenados em arquivos XML para posterior avaliação. A ferramenta realiza ainda cálculos estatísticos de desempenho e plotagem de gráficos.

1. Introdução

Existe hoje, no meio acadêmico, uma grande busca por soluções para problemas complexos que demandam grande poder computacional, como por exemplo, o tratamento de conjuntos de imagens, mineração de dados, simulação de modelos científicos e problemas matemáticos [1].

Dentre as técnicas utilizadas como solução para estes problemas, os agregados de computadores, também chamados de *clusters* [1], e as grades computacionais [2], são mais freqüentes.

Os *clusters* de computadores são arquiteturas paralelas compostos de vários computadores de propósito geral. A grande virtude desse tipo de arquitetura é conseguir alcançar o alto poder computacional desejado, a um baixo custo financeiro [1]. Cada nó que compõe um *cluster* possui um ou mais processadores, e cada processador, uma área de memória. Essas máquinas se comunicam através de passagem de mensagens entre os nós. Em alguns casos, pode ainda haver uma área de memória compartilhada entre os nós. Para realizar a troca de mensagem no *cluster*, a técnica mais habitual é o uso de bibliotecas para passagem de mensagem tal como a *Parallel Virtual Machine (PVM)* [3] e a *Message Passing Interface (MPI)* [4]. O uso dessas bibliotecas se dá através da inserção de funções no código que se deseja paralelisar, realizando a troca de informações e a divisão do trabalho.

Como resultado de décadas de pesquisa na área de processamento paralelo, aliado às pesquisas na área de sistemas distribuídos, foi desenvolvido o conceito de grades computacionais [2].

A área de sistemas distribuídos preocupa-se com aspectos ligados à comunicação e heterogeneidade. A área de processamento paralelo se preocupa em extrair o máximo do poder computacional através da criação de máquinas compostas por agregados de baixo custo financeiro. As grades computacionais são inspiradas nos sistemas de energia elétrica, onde é possível que sejam acoplados recursos geograficamente distribuídos, servindo de fonte de energia sob demanda para diferentes finalidades [2].

A **motivação** para o desenvolvimento deste trabalho se deu durante o atual projeto de pesquisa desenvolvido no Laboratório de Sistemas Digitais e Computacionais (LSDC). Durante o desenvolvimento de aplicativos paralelos os programadores se deparavam com situações nas quais o desenvolvimento e análise de desempenho de aplicativos paralelos apresentavam muita dificuldade [5] [6].

Dada a necessidade que os *clusters* têm de apresentar alto poder computacional, realizando pesados processos em um curto espaço de tempo, a análise de desempenho deste tipo de arquitetura é fundamental. Também nas grades computacionais, a análise de desempenho dos aplicativos é de grande importância, tanto para os programadores quanto para os administradores do sistema [7].

O principal **objetivo** deste trabalho é propor e apresentar uma ferramenta que auxilie a análise de desempenho de aplicações paralelas em *clusters* ou grades computacionais. Este trabalho deve ainda contribuir no meio acadêmico através dos resultados de suas pesquisas.

2. Trabalhos Relacionados

A criação de ferramentas utilizadas em ambientes paralelos aumenta cada vez mais desde a primeira versão dos padrões PVM e MPI. Analisadores de

desempenho, monitores e *benchmarks* vêm sendo desenvolvidos utilizando conceitos semelhantes [7].

Dentre essas ferramentas podemos destacar o Prober [5] [6], Paradyn [8], SvPablo [9] e Carnival [10].

O Prober é uma ferramenta desenvolvida em C++, que analisa o desempenho de aplicações paralelas ou seqüenciais em Windows. Assim como o JProber ele utiliza o tempo de execução como métrica e realização de cálculos estatísticos. O Prober permite a configuração de um ambiente de execução e realização automática dos processos [5] [6].

Paradyn é uma ferramenta para análise de desempenho capaz de monitorar programas pesados por um longo tempo. Essa ferramenta possui uma minuciosa biblioteca de análise de desempenho que utiliza contadores de *hardware*. A sobrecarga na execução do programa e a influência na análise de desempenho do mesmo são mínimas. O grande diferencial do Paradyn é que ele atua dinamicamente nos arquivos executáveis, inserindo e customizando medidas no código enquanto o programa está rodando. Na maioria dos casos a ferramenta é capaz de localizar as partes do programa que são responsáveis pela queda no desempenho [8].

O SvPablo é um software capaz de realizar análise de desempenho e paralelização de programas. Sua biblioteca para instrumentação de códigos trabalha inserindo chamadas da biblioteca MPI nas linguagens C e Fortran, incluindo instrumentação de laços e chamadas de sub-rotinas e funções. A partir da captura dos dados o SvPablo exibe medidas estatísticas dinamicamente. Possui ainda informações dos contadores de hardware e sumário de resultados de desempenho relacionados aos processadores. Uma das grandes vantagens da ferramenta é suportar e interagir com programas instrumentados pela biblioteca PAPI[11]. Outra facilidade é a entrada via arquivo na qual o usuário seleciona os eventos que deseja medir. Caso o usuário não indique um arquivo o software realiza uma medição padrão. A ferramenta roda em plataformas Sun Solaris, SGI IRIX, IBM AIX e Linux/x86 [9].

O Carnival foi desenvolvido com o intuito de melhorar a compreensão sobre o desempenho de processos e auxiliar o desenvolvimento de programas paralelos. O tempo de execução gasto por cada parte do processo é utilizado como métrica para análise das causas/efeitos encontrados nos resultados de análises [10].

A idéia de uma ferramenta totalmente orientada por objetos, que facilite a configuração do ambiente paralelo, são as principais vantagens observadas nestes trabalhos que serão utilizadas pelo JProber. O JProber acrescenta ainda simplificação no suporte à linguagem Java, que não foi observado nas ferramentas apresentadas, e a geração de *scripts* e resultados no formato XML, permitindo a portabilidade entre diferentes aplicações.

3. A Ferramenta JProber

JProber é uma ferramenta em Java, inspirada no Prober, para monitoramento e análise de aplicações distribuídas e paralelas. Ele permite a configuração de um ambiente de execução de aplicações, executa as aplicações sem que seja preciso intervenção do usuário e gera estatísticas sobre o desempenho destas.

A ferramenta foi implementada utilizando o ambiente NetBeans 5.0 [12]. A linguagem Java foi escolhida por agregar vários benefícios para desenvolvimento de aplicações paralelas e distribuídas, como por exemplo: independência de plataforma; mecanismos para a carga e execução seguras de programas remotos; invocação de métodos remotos; uso de *threads* em operações de E/S não - bloqueantes, gerência de temporizadores e execução de tarefas independentes.

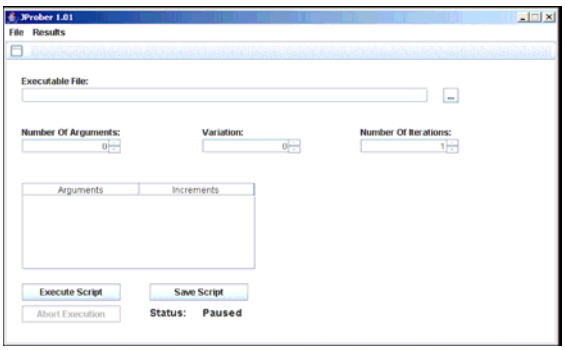

Figura 1. Criação e Execução de *Scripts*

JProber possui uma interface simples que permite ao usuário gerar *scripts* para os testes que deseja executar em seu ambiente paralelo/distribuído, conforme mostrado na Figura 1. Neste ambiente de configuração o usuário irá informar qual é a aplicação a ser testada, quais serão os parâmetros enviados para a aplicação, quantas variações estes parâmetros irão sofrer e quantas iterações serão realizadas para cada variação da execução. O JProber permite ainda que o *script* configurado seja salvo no formato XML e utilizado posteriormente.

Após a configuração do *script* de execução, o usuário submete o comando ao JProber para que os testes sejam realizados. Podem ser realizados testes de desempenho com aplicações executáveis, arquivos em lote ou implementadas em Java (repassados automaticamente para a Java Virtual Machine - JVM).

A utilização de *multithread* na gerência de processos, permite ao JProber interagir com o usuário enquanto testes estão sendo realizados. Sua arquitetura, apresentada na Figura 2, conta ainda com *threads* leitoras do processo analisado. Isso permite armazenar todos os erros e saídas do programa

analisado, gerando informações importantes para a análise de desempenho das aplicações.

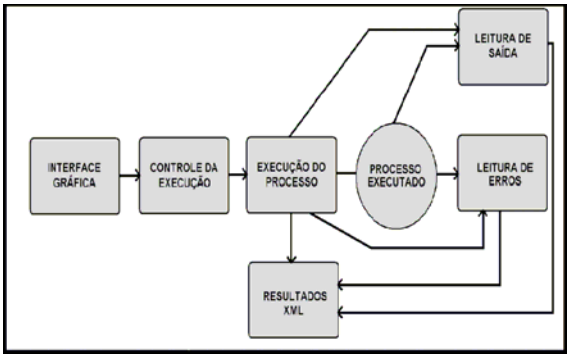

Figura 2. Arquitetura JProber

Os dados obtidos nos testes são salvos no formato XML. A partir destes dados o JProber cria o gráfico de *speedup,* proporcionando ao usuário a visão real de todas as execuções realizadas. A partir dos mesmos dados o JProber gera informações estatísticas dos testes para cada iteração realizada.

4. Experimentos

4.1. Métodos

Os testes realizados utilizando o JProber foram divididos de acordo com seus objetivos principais: o primeiro é verificar o grau de intrusão da ferramenta sobre o aplicativo analisado; o segundo é analisar programas paralelos em um *cluster*. Também podem ser destacados alguns objetivos secundários como encontrar possíveis falhas na ferramenta e localizar limitações que possam ser corrigidas em trabalhos futuros.

Na **primeira parte dos testes** (verificação do grau de intrusão) a ferramenta foi utilizada para analisar diferentes programas simples, escritos em C++ e Java e testados com variadas cargas de trabalho. Os programas utilizados nesta parte dos testes possuíam seus códigos instrumentados com contadores de tempo. A análise realizada compara o tempo coletado pelo próprio programa com o tempo coletado pelo JProber.

Na **segunda parte dos testes** utilizou-se duas aplicações desenvolvidas em nosso laboratório utilizando a biblioteca de passagem de mensagem JPVM [13]. A biblioteca JPVM foi escolhida para que pudessem ser observadas as facilidades que o JProber proporciona à execução de aplicativos Java .

A primeira das duas aplicações realiza o cálculo por aproximação do valor de pi (aplicação **PI**), a segunda realiza o somatório de grandes valores inteiros (aplicação **SOMA**). A escolha dessas aplicações se deve ao alto grau de paralelismo que apresentam e por serem simples de se implementar.

Essas aplicações foram testadas variando-se os valores do número de intervalos para PI e a quantidade de números somados para SOMA, bem como o número de nós do *cluster* utilizados em cada um dos casos.

Utilizamos um *cluster* de 8 máquinas Pentium III 1.0 Ghz (128MB RAM), trabalhando na plataforma Windows 98 e interligadas por uma rede Fast-Ethernet.

4.2. Resultados

A Tabela 1 apresenta os resultados obtidos na primeira parte dos testes. Esses resultados mostram que a diferença entre o JProber e a análise por instrumentação é maior quando o tempo de processamento do aplicativo é menor. Isso ocorre devido ao fato da análise por instrumentação não levar em conta o tempo para carga e descarga do aplicativo da memória, enquanto o JProber considera esse processo na análise do desempenho.

Tabela 1. Análise de Intrusão

Análise de Intrusão						
Aplicação	JProber (ms)			Instrumentação do Código (ms)		
	Média	Pior	Melhor	Média	Pior	Melhor
Java 01	515	610	484	404	407	391
Java 02	4133	4187	4110	4034	4047	4031
C++ 01	4143	4157	4125	4129	4156	4109
C++ 02	5837	5953	5797	5810	5829	5781
Aplicação	Diferença Absoluta (ms)			Diferença Relativa (%)		
	Média	Pior	Melhor	Média	Pior	Melhor
Java 01	111	203	93	21,55%	33,28%	9,21%
Java 02	99	140	79	2,40%	3,34%	1,92%
C++ 01	14	1	16	0,34%	0,02%	0,39%
C++ 02	27	124	16	0,46%	2,08%	0,28%

Quando o tempo de processamento aumenta, o tempo de carga do aplicativo passa a ser menos significativo na análise, aproximando o resultado das duas análises. Isso comprova que a intrusão do JProber sobre o aplicativo analisado é baixa.

Na Figura 3, é possível observar o *speedup* obtido com PI para 100 milhões de intervalos. Através da análise do gráfico gerado é possível que o programador conclua que, apesar da divisão da carga de trabalho entre oito processadores gerar *speedup* a eficiência do mesmo diminuiu à medida que se aumentava o número de processadores.

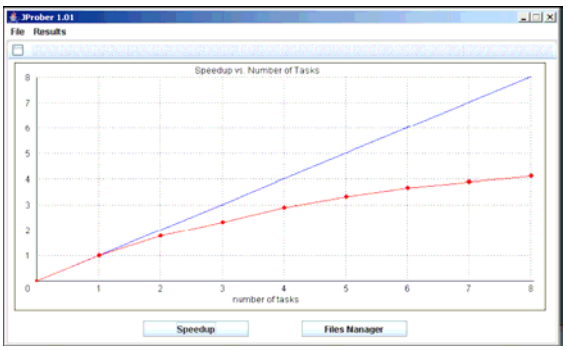

Figura 3. *Speedup* PI - 100 milhões de Intervalos

Por outro lado, a utilização de um bilhão de intervalos no cálculo de PI, por ser uma tarefa com maior carga de processamento, gerou aumento na granularidade. Essa diferença, observada na Figura 4, nos permite induzir que com maior número de máquinas no cluster o tempo de execução seria ainda menor.

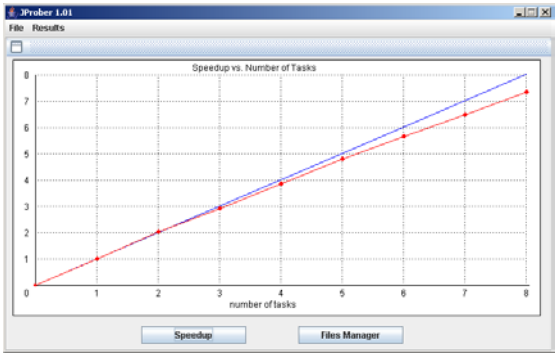

Figura 4. *Speedup* PI - 1 bilhão de Intervalos

Uma situação semelhante é encontrada na análise da aplicação SOMA. A quantidade de números utilizados foi 500 milhões e 700 milhões. Para a segunda configuração o *speedup* foi melhor do que para a primeira.

Ao compararmos o aplicativo SOMA com o aplicativo PI, notamos uma baixa carga de processamento do aplicativo SOMA, decorrente da menor complexidade do problema. A Figura 5 demonstra o menor *speedup* decorrente desta diferença.

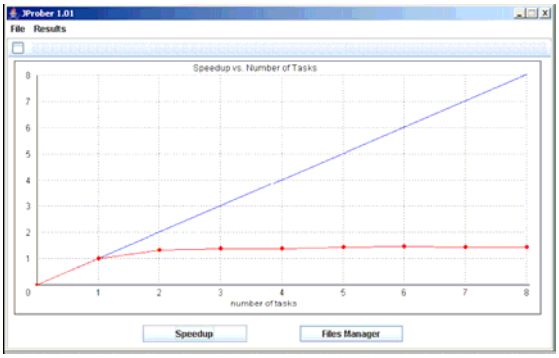

Figura 5. *Speedup* SOMA - 700 milhões

5. Conclusões

A programação paralela em *cluster*s de computadores como solução para problemas que demandam alto poder computacional se mostrou eficiente, dadas às métricas coletadas pela ferramenta proposta e analisadas.

Os objetivos do trabalho foram alcançados, pois propomos, apresentamos e verificamos uma ferramenta que facilitou o teste e a análise de desempenho de programas, sem que no entanto seu desempenho fosse prejudicado.

A geração de gráficos permitiu-nos uma análise precisa do *speedup* alcançado. A utilização de uma interface gráfica simples para a geração de *Scripts* facilitou a configuração do ambiente e, a utilização de arquivos no formato XML permitiu a visualização dos resultados alcançados através de outros aplicativos.

Como trabalhos futuros, podemos citar a implementação de outras métricas além do tempo de execução, para análise de programas paralelos. Além disso, serão realizados experimentos comparativos entre o JProber e outras ferramentas.

Referências

[1] Hwang, K.; Xu, Z. "Scalable Parallel Computing: Technology, Architecture, Programming", Editora Macgraw-Hill, 1998.

[2] Foster, I., "What is the Grid? A Three Point Checklist", GRIDToday, July 20, 2002

[3] PVM – Parallel Virtual Machine, Computer Science and Mathematics Division do Oak Ridge National Laboratory. URL: www.csm.ornl.gov/pvm/pvm_home.html

[4] The Message Passing Interface (MPI) Standard, Argonne National Laboratory, University of Tenesee, URL: http://www-unix.mcs.anl.gov/mpi/

[5] L. E. S. Ramos, L. F. W. Góes, C. A. P. S. Martins, "Prober: Uma Ferramenta de Análise Funcional e de Desempenho de Programas Paralelos e Configuração de *Cluster*", 2º WSCAD, Brasil, 2001, pp 127-134.

[6] Góes, L. F. W., Ramos, L. E. S., Martins, C. A. P. S., "Performance Analysis of Parallel Programs using Prober as a Single Aid Tool". 14th Symposium on Computer Architecture and High Performance Computing (SBAC - PAD), Vitória, 2002.

[7] Gerndt, M., Wismüller, R., "Performance Tools for the Grid: State of the Art and Future", Institut für Informatik, LRR, Technische Universität München, 2004

[8] Miller, B. P., Hollingsworth, J. K., Callaghan, M. D., "The Paradyn Parallel Performance Tools and PVM", Environments and Tools for Parallel Scientific Computing, SIAM Press, 1994.

[9] Rose, L. D., Reed, D. A., "SvPablo: A Multi-Language Architecture-Independent Performance Analysis System", Departament of Computer Science, University of Illinois, Urbana, USA. URL: www-pablo.cs.uiuc.edu/

[10] Junior, W. M., LeBlanc, T. J., Hardavellas, N., Amorim, C., Understanding the Performance of DSM Applications, Proceedings of CANPC '97 Workshop on Communication and Architectural Support for Network-Based Parallel Computing, Feb 1997.

[11] Performance Application Programming Interface (PAPI) Library, Innovative Computing Laboratory, URL: http://icl.cs.utk.edu/papi/

[12] Netbeans open-source Project, Sun Microsystems - http://www.netbeans.org

[13] Ferrari, A., "JPVM: Network Parallel Computing in Java", Department of Computer Science, University of Virginia, 1997. URL: www.cs.virginia.edu/~ajf2j/jpvm.html