REVISTA CIENTÍFICA ELETRÔNICA DE SISTEMAS DE INFORMAÇÃO - ISSN 1807-1872
PUBLICAÇÃO CIENTÍFICA DA FACULDADE DE CIÊNCIAS JURÍDICAS E GERENCIAIS DE GARÇA/FAEG
ANO II, NÚMERO, 04, FEVEREIRO DE 2006. PERIODICIDADE: SEMESTRAL

PERFORMANCE EM AMBIENTES DISTRIBUÍDOS USANDO MPI PARA PROCESSAMENTO DE IMAGENS MÉDICAS

TAMAE, Rodrigo Yoshio
ROSA, Adriano Justino
MUZZI, Fernando Augusto Garcia
Docentes da Faculdade de Ciências Gerenciais e Jurídicas de Garça – FAEG/Garça
rytamae@yahoo.com.br; adriano@faef.br; fagmuzzi@yahoo.com.br

PRIMO, André Luis Gobbi
Departamento de Internet e Docente da FEF – Fundação Educacional de Fernandópolis

MUCHERONI, Marcos Luiz
Centro Universitário Eurípides de Marilia, PPGCC-UNIVEM

RESUMO

Este trabalho tem por objetivo apresentar os resultados iniciais obtidos em testes de desempenho em ambientes distribuídos em *clustering* utilizando arquiteturas Beowulf com implementações MPI para processamento de imagens médicas.
Palavras-chave: MPI, Computação distribuída e Imagens Médicas.

ABSTRACT

The objeticve of this work is presents initial results obtained in tests to available the performance gain in clustering distributed environments using Beowulf architecture with MPI implementations to solve medical images processing.
Keywords: MPI, Distributed Computing and Medical Images.

1. INTRODUÇÃO

A computação paralela surgiu a partir da necessidade crescente por obter-se alto desempenho computacional. Com a evolução dos processadores e demais componentes que possibilitaram a interconexão destes processadores e, assim, coordenar seus esforços computacionais, atualmente torna-se possível a construção de um sistema computacional paralelo a relativo baixo custo,

melhorando a relação custo e desempenho, normalmente atrelado a sistemas de alto desempenho.

Existem diversas arquiteturas escalares de computação paralela, a citar: *Massively Parallel Processors* (MPP), *Symmetric Multiprocessors* (SMP), *Cache-Coherent Nonuniform Memory Access* (CC-NUMA), *Distributed Systems* e *Clusters.* Dentre estes, destacam-se os *Clusters* que surgiram a partir de um movimento iniciado no início dos anos 90 com a finalidade de encontrar uma alternativa em relação aos caríssimos sistemas paralelos proprietários através da interconexão de PCs por meio de redes de alta performance.

Embora os computadores estejam evoluindo e tornando-se cada vez mais velozes, existem limites físicos e a velocidade dos circuitos não poderá continuar melhorando indefinidamente, a menos que surjam novos avanços tecnológicos ou que ocorra a quebra do paradigma computacional tradicional. Por outro lado, tem-se observado grande aceitação quanto ao uso de implementações em sistemas paralelos nas aplicações de alto desempenho, motivados pelo surgimento de novas arquiteturas que integram dezenas de processadores de baixo custo, tais como o *clustering* e o *grid-computing*.

Tanto nas áreas de pesquisa acadêmica quanto nas áreas de aplicação comercial é notória a necessidade de obtenção de desempenho computacional a um custo relativamente mais baixo.

Dentro deste contexto, podem ser citados os complexos sistemas que exigem alto poder de processamento, armazenamento e transmissão, como é o caso dos ambientes de radiologia digital PACS (*Picture Archiving and Communication Systems*) de imagens médicas.

Um ambiente denominado SISPRODIMEX (Sistema de Processamento Distribuído de Imagens Médicas com XML), utilizado para distribuição de imagens médicas, tem como um dos objetivos a redução no tempo de resposta aos clientes deste sistema, prejudicada, ainda, devido às limitações dos meios de interconexão remota entre servidor e clientes, e faz uso de um *cluster* experimental baseado na arquitetura Beowulf.

Os resultados iniciais coletados a partir destas implementações são apresentados a seguir.

2. CONSIDERAÇÕES INICIAIS SOBRE AMBIENTES PARALELOS E O MPI

Uma grande alternativa de se obter máquinas paralelas a baixo custo é utilizar a capacidade ociosa de máquinas disponíveis em ambientes industriais ou educacionais. Pode-se, por exemplo, interconectá-las viabilizando os projetos de *clustering*. Estes tipos de arquiteturas já estão disponíveis e, em geral, compõem ambientes com memória distribuída. Um dos exemplos são as estações de trabalho ou PCs conectados em rede (FILHO, 2002).

Uma outra característica, importante em sistemas distribuídos são as tecnologias utilizadas para se interligar os computadores. Um ambiente que se destaca é o PVM (*Parallel Virtual Machine*), que é um software que permite que um conjunto heterogêneo ou homogêneo de computadores seja visto como uma única máquina, sendo a portabilidade uma de suas características principais – as bibliotecas de rotinas de comunicação entre processos são "*standard*" de fato. A independência de plataforma que o PVM disponibiliza é indubitavelmente interessante; um software pode ser executado em ambientes diferentes, este fato gera segurança para desenvolvedores de software criarem aplicações paralelas, tendo em vista a portabilidade possível. Uma outra tecnologia importante no conceito de computação distribuída é o padrão MPI (*Message Passing Interface*), que é um padrão de interface para a troca de mensagens em máquinas paralelas com memória distribuída. Não se deve confundi-lo com um compilador ou um produto específico. O projeto do MPI teve início em 1992 com um grupo de pesquisadores de várias nacionalidades e fabricantes de computadores do mundo todo. Ele é uma tentativa de padronizar a troca de mensagem entre equipamentos.

Um exemplo de implementação do padrão MPI é o MPICH. Foi projetada para ser portável e eficiente. O "CH" referenciado no nome vem de *Chameleon* (Camaleão), símbolo de adaptabilidade – e, portanto, de portabilidade - para o ambiente onde está sendo utilizado. As duas características acima foram desenvolvidas nas linguagens C e FORTRAN. Contudo, com o surgimento da linguagem Java, também foram desenvolvidas ferramentas para computação distribuída nesta linguagem. O JMPI é um exemplo disso, sendo um projeto de propósito comercial da *MPI Software Technology, Inc.*, implementado a partir do projeto de mestrado de Steven (MORIN, 2000) com o intuito de desenvolver um sistema de passagem de mensagem em ambientes paralelos utilizando a

linguagem Java. O JMPI combina as vantagens da linguagem Java com as técnicas de passagem de mensagem entre processos paralelos em ambientes distribuídos (DINCER, 1998).

Serão apresentados resultados experimentais da execução de uma aplicação Java utilizando o JMPI-PLUS e uma aplicação em C utilizando MPICH, ambos em uma arquitetura de *cluster* Beowulf.

3. MATERIAIS E MÉTODOS

O JMPI-PLUS é o resultado da pesquisa de André L.G. Primo (PRIMO, 2005) em sua dissertação de mestrado e consiste em uma implementação do padrão MPI baseada em Java que teve como base as classes do mpiJava desenvolvido por Bryan Carpenter e Geoffrey Fox da NPAC, Syracuse University, Syracuse, USA; Vladimir Getov da School of Computer Science, University of Westminster, London, UK; Glenn Judd da Computer Science Department, Brigham Young University, Provo, USA; Tony Skjellum da MPI Software Technology, Inc., Starkville, USA. (CARPENTER, 2000). O objetivo da implementação do JMPI-PLUS foi o aprimoramento, no que diz respeito aos métodos, de envio de mensagens da implementação mpiJava. Para os teste e coleta de dados iniciais realizou-se o tratamento do transporte de mensagens usando serialização de objetos (CORREIA, 2005).

Para isso, foi construído um *cluster* baseado na arquitetura Beowulf que se encontra nas dependências da Fundação Educacional de Fernandópolis – FEF e pode ser observado na figura 1, composto por três nós processadores, sendo os escravos chamados de “escravoN”, (onde “N” varia de acordo com o número de escravos) e o servidor de “master”. As máquinas do *cluster* são equipadas com CPU’s Pentium IV de 2.4GHz, memória RAM de 256MB, HD de 40GB, placa de rede Intel(R) PRO 10/100 VE Network Connection, com floppy Disk de 1.4MB e unidade de CD-ROM. Foi utilizada a versão do sistema operacional Linux Red Hat 9.0.

Figura 1 – Cluster baseado na Arquitetura Beowulf utilizado no projeto (PRIMO, 2005).

Segundo (WALKER, 2001), a arquitetura Beowulf é uma categoria especial de *cluster* construído a partir de máquinas e dispositivos de baixo custo facilmente encontrados no mercado. A idéia principal é que seja feita uma configuração de *hardware* e *software* para que se tenha um aumento de performance na execução paralela de aplicações. A arquitetura Beowulf pertence ao grupo dos sistemas SSI (*Single System Image*), pois o usuário vê o sistema como se fosse uma máquina única.

Uma característica importante que se tem em arquiteturas Beowulf é a centralização dos pacotes de programas e contas de usuários em uma máquina servidora (*master*) que é a responsável pelas tarefas administrativas do *cluster*. Quando uma tarefa é enviada ao *cluster*, a máquina servidora recebe e divide a tarefa entre os nós processadores, também chamados de escravos (*slave*). Sendo assim, um *cluster* Beowulf de dois nós, na verdade possui um total de três máquinas, sendo uma máquina servidora e duas máquinas escravas (STERLING, 1999).

No nível de sistema operacional um dos aspectos mais importante da configuração Beowulf é a configuração do NFS (*Network File System*) e do NIS (*Network Information Service*). O objetivo da configuração NFS é permitir que a máquina servidora compartilhe a sua partição que contém os softwares usados no *cluster*, normalmente instalados no diretório /usr/local, fazendo uma exportação para os nós escravos.

Normalmente, o diretório /home também é exportado. Os nós escravos por sua vez fazem à montagem desses *file systems* em suas áreas locais com os mesmos nomes, /usr/local e /home. Com isso, a atualização e administração dos softwares do *cluster* se torna bem mais fácil e elimina um dos problemas: o de controle das versões dos pacotes instalados. Dessa maneira o pacote JDK (*Java Development Kit*), por exemplo, seria instalado no diretório /usr/local/j2sdk1.4. Os nós escravos tendo o *file system* montado precisam acrescentar apenas as informações do JDK na variável PATH e CLASSPATH e podem usar os pacote da *Sun Microsystems* para desenvolver e rodar suas aplicações, sem a necessidade de instalar localmente o pacote (STERLING, 1999) (SOUZA, 2003).

O NIS é utilizado porque é o responsável por fazer a autenticação centralizada dos usuários. O usuário criado na máquina servidora, também tem que estar criado em todos os nós, pois o usuário utilizado na máquina servidora, automaticamente tem que ter acesso a todas as máquinas do cluster, pois suas informações são repassadas para todos os nós da rede através do servidor NIS.

Para tentar-se melhorar o envio de mensagens trabalhou-se diretamente na alteração das rotinas dos métodos do pacote mpiJava.

- public byte[] Object_Serialize
- public void Object_Deserialize
- public void Rsend
- public void Object_Ibsend
- public Request Irecv

A figura 2 ilustra o diagrama de classe mpiJava, que foi a implementação do padrão MPI que deu origem a JMPI-PLUS.

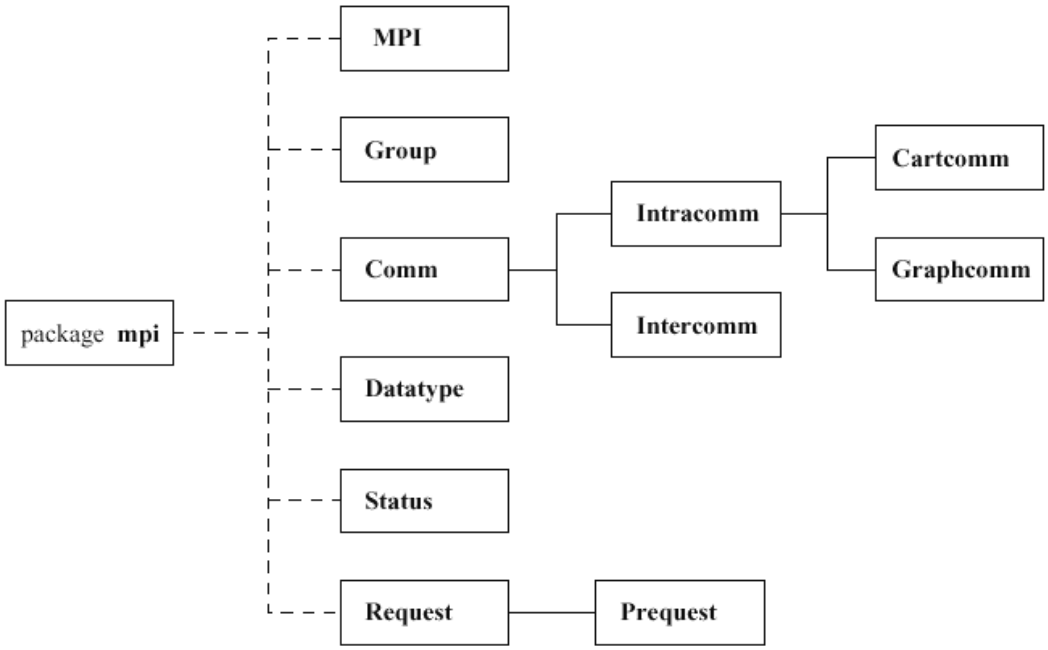

Figura 2 – Estrutura de classes do mpiJava (CARPENTER, 2000), que originou o JMPI-PLUS.

4. RESULTADOS E DISCUSSÕES

Para executar os testes e coletar os dados iniciais apresentados neste trabalho foi desenvolvida uma aplicação em linguagem Java utilizando os conceitos de JMPI-PLUS que efetua consulta ao banco de dados de imagens médicas (em formato de objetos DICOM – *Digital Image and Communication in Medicine*) e, em seguida, submete-a a processamento do *cluster* para serialização, obtendo-se assim, o tempo resultante do processo.

Para efeito de comparação de resultados, o mesmo processo foi repetido, desta vez, utilizando os conceitos do MPICH (com imagens em formato JPEG). Os testes foram realizados com três tamanhos diferentes de imagens. O gráfico apresentado na Figura 3 ilustra o resultado obtido com a realização entre o processamento de uma imagem no padrão DICOM que foi processada utilizado a implementação do JMPI-PLUS e uma imagem no formato JPEG utilizando a implementação do MPICH.

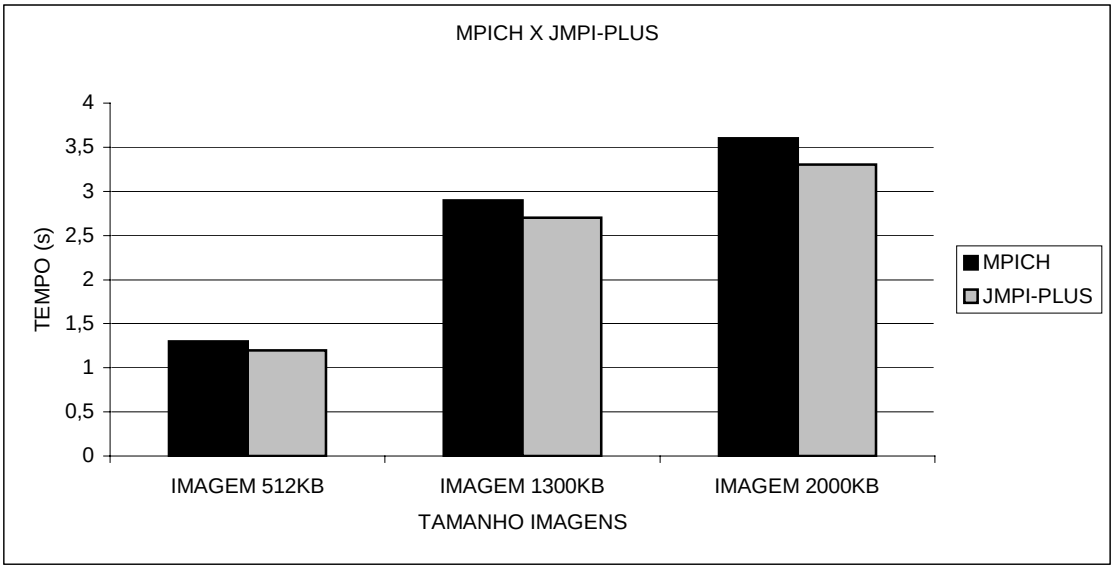

Figura 3 – Comparação entre o processamento utilizando MPICH e JMPI-PLUS

Pode-se observar no gráfico acima que a implementação do MPI baseada em Java apresentou melhor resultado que a implementação MPI baseado em C. Normalmente uma aplicação feita utilizando a linguagem C, que é a linguagem que o MPICH foi implementado, é mais rápida que uma mesma implementação desenvolvida em Java. O resultado, inicialmente polêmico deste teste, pode ser explicado pelo fato do código em linguagem C ser compilado e o código Java ser interpretado (SOUZA, 2003). Um código compilado normalmente é mais rápido que um código interpretado. Mas, levando-se em consideração que o mpiJava, e, portanto, o JMPI-PLUS também, implementa a serialização de objetos e o *marshalling* de dados, conseguiu-se com isso um ganho considerável na troca de mensagem entre os nós processadores, chegando a aproximadamente 29% de ganho na troca de mensagem em relação ao MPICH.

Pode-se observar também que a distribuição e tarefas de pequeno porte no sistema não houve um grande ganho na performance, isso ocorreu devido ao tempo perdido com o *overhead* entre a troca de mensagens.

5. CONCLUSÕES

As implementações e testes iniciais com o *cluster* e o uso da metodologia JMPI-PLUS vêm a ratificar a proposta deste trabalho, ou seja, garantir poder de processamento e minimizar o impacto nas respostas solicitadas pelos clientes, pois evita o uso isolado de caríssimos sistemas de *Workstations*, e demonstra claramente as contribuições tanto para a área de pesquisa baseada em

informações e imagens médicas advindas de ambientes PACS quanto para as instituições que tem a nítida necessidade de obter um melhor gerenciamento e aproveitamento de suas informações.

Para a área médica, em particular, este sistema é de grande auxílio ao diagnóstico e poderá disponibilizar recursos em rede a um custo relativamente mais baixo que sistemas semelhantes na área da saúde.

6. REFERÊNCIAS BIBLIOGRÁFICAS

FILHO, Virgílio J. M. Ferreira. **MPI-Implementação Paralela**, Universidade Federal do Rio de Janeiro, Departamento de Engenharia Industrial, Rio de Janeiro – Rio de Janeiro, 2002.

MORIN, Steven Raymond. **JMPI: Implementing the Message Passing Interface Standard in Java**, University of Massachusetts, graduate degree in Master of Sience in Electrical and Computer Enginnering, september 2000, disponível em: http://www.ecs.umass.edu/ece/realtime/ publications/steve-thesis.pdf.

DINCER, Kivanc. **jmpi and a Performance Instrumentation Analysis and Visualization Tool for jmpi**. First UK Workshop on Java for High Performance Network Computing, EUROPAR-98, Southampton, UK, September 2-3, 1998.

CORREIA, Vasco Martins**: Serialização dos dados de Imagens Médicas usando troca de mensagens**, LES - Laboratório de Engenharia de Software - Centro Universitário Eurípides Soares da Rocha – UNIVEM, 2005.

WALKER, B. J. **Introdution to Single System Image Clustering** (2001). Disponivel em http://www.souceforge.net acessado em 15 de julho de 2005.

STERLING, Thomas L. Salmon, John. Becker, Donald J. e Savaresse, Daniel F. **How to build a Beowulf: a guide to the implementation and aplication of PC clusters**. Massachusetts Institute of Technology. 1999.

SOUZA, G.P.; PFITSCHER, H.; MELO, A.C.M.A. **Computação Distribuída Baseada em Java Rodando em Arquiteturas Beowulf e Arquiteturas Heterogêneas**. Departamento de Computação – Universidade de Brasília (UNB). 2003.

CARPENTER, Bryan, Mark Baker, Geoffrey Fox, Sung Hoon Ko and Sang Lim**. mpiJava: An Object-Oriented Java interface to MPI**. june 2000.

PRIMO, A.L.G. **Serialização e Performance em Ambientes Distribuídos Usando MPI**. 2005. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-graduação em Ciência da Computação (PPGCC) Centro Universitário Eurípides de Marília – UNIVEM. 2005.