SPD: UM NÚCLEO DE PROGRAMAÇÃO DISTRIBUÍDA EM REDES DE COMPUTADORES

Wilton S. Caldas Daniel Calvo Renato A. C. Ferreira
Wagner Meira Jr* Osvaldo S. F. Carvalho†

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Caixa Postal 702, 30161 Belo Horizonte - MG - Brasil
Telefone: (031)443-4088 FAX: (031)443-4352
E_mail: wilton@dcc.ufmg.br ou vado@dcc.ufmg.br

Resumo

Este artigo descreve o SPD, um sistema de programação distribuída para redes de estações de trabalho, que funciona em plataformas UNIX onde se tenham disponíveis o protocolo TCP/IP e a biblioteca TLI. O sistema é composto basicamente por três ferramentas: um servidor, que é o responsável pela gerência do sistema e das trocas de mensagens; uma biblioteca de funções, que fornece ao usuário o subsídio necessário para a utilização dos recursos oferecidos pelos servidores; e, finalmente, um programa que instancia as aplicações do usuário no sistema distribuído.

Abstract

This paper describes SPD, a system for distributed programming over computer networks, that runs over a UNIX background provided with TCP/IP protocol and the TLI library. The system is composed basically by three tools: a server, which performs the system management and the message exchanges; a library, that provides the user with the functions required for using the system; and, finally, a program for the startup of user aplications over the distributed system.

*Mestrandos em Ciência da Computação pelo DCC-UFMG

†Bacharel em Física, 1974, UFMG. Doutor de Estado, 1985, Université Pierre et Marie Curie (Paris VI) e Professor Adjunto do Departamento de Ciência da Computação da UFMG

1 Introdução

Nos últimos anos, a utilização do processamento paralelo e distribuído surgiu como uma solução promissora para o atendimento das necessidades computacionais do futuro. Avanços significativos em algoritmos e arquiteturas paralelas têm demonstrado o potencial de técnicas de programação paralela para uma grande variedade de problemas. Uma possibilidade atraente é o uso de redes locais, presentes em muitas universidades e empresas brasileiras, como um multiprocessador. Ciclos ociosos das estações podem ser aproveitados em aplicações distribuídas, proporcionando em princípio uma grande potência de computação a um custo quase nulo.

Entretanto, o desenvolvimento de programas distribuídos, além das dificuldades próprias de aplicações paralelas, exige do programador um conhecimento profundo dos recursos de comunicação existentes entre as máquinas disponíveis no sistema. Isso torna a implementação de aplicações distribuídas um trabalho penoso para o programador, além de demandar muito tempo e ser extremamente propenso a erros. Uma maneira de facilitar esse trabalho é oferecer uma abstração da comunicação entre máquinas, de forma que o programador possa concentrar seus esforços na aplicação em si, não se preocupando com problemas de comunicação.

O SPD, sistema de programação distribuída, desenvolvido no DCC/UFMG, é uma ferramenta para a criação de aplicações distribuídas em redes de computadores. O sistema necessita de uma plataforma UNIX contendo o protocolo de comunicação TCP/IP e a interface TLI (*Transport Layer Interface*) [6, 7]. O SPD provê um conjunto de primitivas de comunicação que podem ser incorporadas a um programa C. Esta comunicação pode ser feita tanto de forma assíncrona como síncrona. O sistema consiste essencialmente de um conjunto de protocolos destinados a garantir uma transferência de dados segura.

A interligação de processo no SPD foi inspirado no sistema de programação e configuração de transputers [3]. Neste modelo, uma aplicação é um conjunto de processos seqüenciais que se comunicam entre si somente através de canais. Um canal conecta dois processos, ou dois grupos de processos cujos componentes sejam distintos. Contrastando com esta abordagem pode-se endereçar mensagens a processos e não a canais, como é feito no sistema PVM [8].

Um exemplo do primeiro modelo é a arquitetura Transputer [3], onde até a comunicação entre processadores é feita via canais. Já no segundo caso podemos citar o PVM [8], onde todas as mensagens são endereçadas a processos ou a instâncias de processos.

Este projeto foi desenvolvido visando servir como laboratório para experimentação e pesquisa de algoritmos distribuídos e sistemas operacionais, além de conhecer e utilizar melhor os recursos computacionais existentes no DCC-UFMG. O SPD vem se juntar a outros ambientes de processamento paralelo desenvolvidos no DCC como o Têmis[2], Linda [5] e APPAT [4].

Este trabalho descreve, a seguir, a camada do SPD com a qual o usuário tem contato. Esta seção compreende o modelo de criação de tarefas pelo SPD e a biblioteca disponível. A seguir, será descrito o modelo de tratamento interno do sistema, onde se terá a ênfase no servidor. Após isso, apresenta-se alguns resultados de experiência de comparação do SPD com o PVM, e finalmente, as conclusões e as perspectivas futuras.

2 O SPD

Uma *aplicação*, no SPD, é um conjunto de *programas* que cooperam para a solução de um dado problema. Os programas que compõem a aplicação trocam informações entre si através de *canais de comunicação*. Uma aplicação, quando está executando, é composta por diversos

processos cooperantes, que são *instâncias* dos programas que a compõem. O SPD permite a existência de diversas instâncias de um mesmo programa durante uma mesma *sessão de execução*, ou seja, podem existir vários processos que correspondem a um mesmo programa.

Além disso, o SPD permite a execução de diversas aplicações simultaneamente na rede. Daí surge o conceito de *grupo*, que é o nome dado ao conjunto de processos de uma mesma aplicação, internamente ao SPD. Cada grupo é tratado independentemente e paralelamente no sistema, de forma que a execução de um não interfira na execução de outro, exceto por questões de desempenho. Os erros também são tratados a nível de grupo, ou seja, o sistema, quando detecta um erro em um dado grupo, notifica todos os outros processos do mesmo sem interferir nos outros grupos que, porventura, estejam em execução no momento.

A distribuição da aplicação entre as máquinas que compõem a rede, bem como o modo de conexão dos diversos canais dos vários programas, são definidos em um arquivo, escrito pelo usuário, de *configuração da aplicação*, que consiste de: identificação dos programas que compõem a aplicação, identificação das máquinas que serão utilizadas durante a execução, as instâncias de cada um dos programas com as respectivas máquinas onde devem estar executando e as interconexões dos canais. A figura 2 mostra um pequeno exemplo deste arquivo de aplicação. Através do exemplo, pode-se notar que a linguagem de configuração é uma ferramenta suficientemente poderosa para permitir ao usuário relocar programas em máquinas, alterar o número de instâncias de cada programa para execução e as conexões dos programas com certa facilidade, sem que seja necessária a recompilação dos mesmos. Este arquivo será descrito com mais detalhes posteriormente.

A arquitetura do sistema é ilustrada na figura 1. As máquinas envolvidas podem ser heterogêneas, desde que exista uma versão dos programas (o próprio SPD e os programas do usuário) compilados para cada uma das arquiteturas utilizadas. É importante notar que deve existir um servidor do SPD executando em cada máquina onde se deseja disparar processos. Os servidores são conectados entre si, cada um em uma máquina, utilizando o protocolo TCP/IP através da TLI. Toda a comunicação entre processos é realizada via servidor, donde se conclui que ele é o componente principal de todo o sistema. Esses dois fatores, embora tornem o sistema um pouco mais lento em relação a alguns similares, possibilitam a execução de programas em redes, inclusive de longas distâncias, além de facilitarem a portabilidade, visto que o SPD utiliza, em sua implementação, funções de alto nível padronizadas.

Além do servidor, os outros dois componentes que, integrados, compõem o SPD são: uma biblioteca de funções que oferecem os recursos de comunicação necessários aos usuários do sistema e um programa capaz de receber o arquivo de configuração da aplicação e disparar sua execução.

Essas três ferramentas, bem como o arquivo de aplicação, são detalhados a seguir.

2.1 O arquivo de configuração da aplicação

Este arquivo é a principal ferramenta utilizada pelo usuário na descrição de suas aplicações, especificando a configuração desejada na execução.

A estrutura desse arquivo é detalhada a seguir:

- Primeiramente, todas as estações que serão utilizadas pela aplicação devem ser discriminadas, uma por linha, permitindo ao SPD conhecer e verificar a acessibilidade da estação desejada antes de prosseguir. O comando P no arquivo de configuração faz a definição dos processadores, obedecendo à seguinte sintaxe:

  P *<estação>*

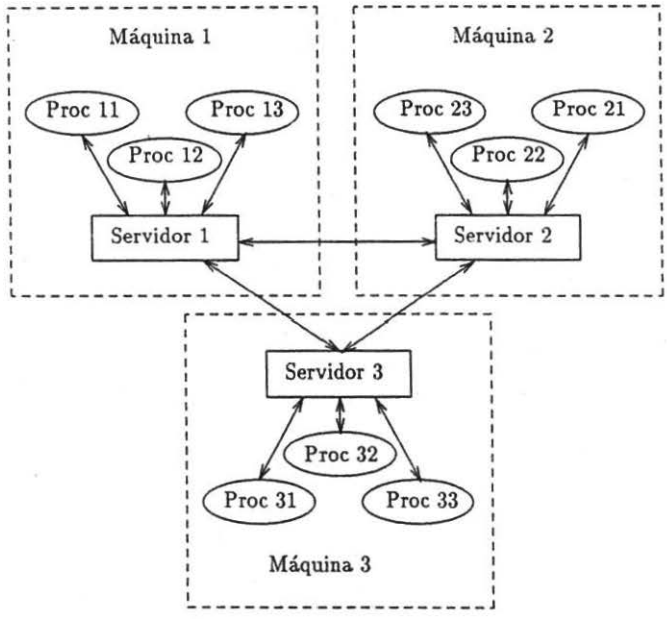

Figura 1: Arquitetura do SPD

```
P rubi
P topazio
T xmandel 1 1 1
T gmandel 2 1 1
L xmandel 0 aplic/mandel/ rubi -2.5 0.5 -1 1
L gmandel 0 aplic/mandel/ rubi
L gmandel 1 aplic/mandel/ topazio
C xmandel 0 0 = gmandel 0 0 , gmandel 1 0 .
C gmandel 0 0 , gmandel 1 0 = xmandel 0 0 .
```

Figura 2: Arquivo de configuração da aplicação SPD

Rotina	Função
open_spd();	abertura da comunicação
close_spd();	fim da comunicação
sendc(canal, mensagem);	envio síncrono de mensagem
receivec(canal, mensagem);	recepção síncrona de mensagem
a_sendc(canal, mensagem);	envio assíncrono de mensagem
a_receivec(canal, mensagem);	recepção assíncrona de mensagem

Tabela 1: Rotinas disponíveis no SPD

- A seguir, descreve-se todos os programas que compõem a aplicação, juntamente com o número de instâncias que serão criadas. O número de canais de entrada e de saída também deve ser fornecido. O comando T realiza essa descrição, segundo a sintaxe abaixo:

  T *<tarefa> <# instâncias da tarefa> <# canais de entrada> <# canais de saída>*

- O próximo comando, L, destina cada instância a uma determinada máquina. São fornecidos o nome do programa, o número da instância, o diretório onde se encontra o executável, a máquina alvo e, quando necessário, os parâmetros da linha de comando. As instâncias passam a ser identificadas pelo nome do programa e pelo número da instância, devendo existir uma versão compilada para a máquina a ser utilizada. A estrutura do comando é dada a seguir:

  L *<nome > <índice > <diretório> <estação> [parâmetros]*

- Finalmente, descreve-se a ligação dos canais de comunicação. O comando C é usado para conectar os canais de entrada a canais de saída. Observe que listas podem ser especificadas em ambos os lados da atribuição, dando flexibilidade a essa especificação. Não é permitido a utilização de listas em ambos os lados da atribuição ao mesmo tempo, mas apenas um dos dois lados de cada vez. Desta forma, conexões podem ser de três tipos: 1 para 1, 1 para n e n para 1. A sintaxe do comando é mostrada a seguir:

  C *<tarefa índice canal_entrada>* [, ...] = *<tarefa índice canal_saída>* [, ...]

Esse arquivo permite que o usuário faça diversas configurações para a execução de sua aplicação de uma maneira fácil e eficiente. Um exemplo de arquivo de configuração é mostrado na figura 2.

2.2 A biblioteca do sistema

Os programas que compõem as aplicações acessam o SPD através de seis funções, disponíveis na biblioteca do sistema e que são mostradas na tabela 1.

As duas primeiras funções da tabela são, respectivamente, a função de abertura das comunicações e a de encerramento. A função de abertura deve ser chamada no programa do usuário antes que seja feita a primeira troca de mensagem. Ela indica ao servidor que o processo está em execução e irá iniciar as transmissões de dados. A função de encerramento indica ao servidor que o processo terminou suas trocas de mensagens, ou seja, não irá necessitar de mais serviços. Estas duas funções permitem ao servidor acompanhar o estado dos diversos processos, de forma que este possa detectar alguns erros de execução, forçar a

terminação da execução de todos os outros processos do grupo, além de notificar ao usuário o ocorrido.

As outras quatro funções são as que realmente realizam as trocas de mensagens entre os processos. Estas funções servem para enviar e receber dados em duas modalidades: síncrona e assíncrona. Na modalidade assíncrona, o processo envia a mensagem e continua sua execução normal. O servidor armazena a mensagem até que seja recebida, garantindo a sua entrega independentemente de qualquer atraso na transmissão ou recepção. Na recepção assíncrona, o processo recebe a mensagem ou uma indicação de erro, caso ela não esteja disponível. Por outro lado, no modo síncrono, o envio bloqueia o processo até que a mensagem seja recebida, enquanto a recepção faz o processo aguardar até que exista uma mensagem disponível. As funções de envio e recepção devem ser usadas apropriadamente, ou seja, síncrono com síncrono e assíncrono com assíncrono. O sistema não permite o cruzamento destas funções.

Os parâmetros das funções de troca de mensagem são iguais e correspondem respectivamente ao canal onde se deseja enviar ou receber (o processo parceiro na comunicação é definido no arquivo de configuração da aplicação, onde se estabelecem as interconexões dos canais), e a mensagem propriamente dita. O tipo da mensagem é uma estrutura padrão do tipo:

```
typedef struct {
                  unsigned short len;
                  char * data;
                } msg_type;
```

O campo `len` representa o tamanho da mensagem e `data` corresponde ao texto transmitido.

Um último detalhe a respeito das funções de troca de dados é que não se permite mensagens *broadcast*. Mesmo que o canal esteja conectado a mais de um processo, apenas uma cópia da mensagem existirá no sistema e, conseqüentemente, apenas um processo poderá recebê-la.

2.3 O disparador de aplicações

Este é o programa que deve ser executado pelo usuário para disparar a execução de uma aplicação do SPD. Sua função é basicamente interpretar as informações contidas no arquivo de configuração, enviando-as para o servidor da máquina onde a aplicação está sendo disparada.

Quando se dispara uma aplicação, o servidor local assume a condição de *servidor mestre* da mesma, passando a acompanhar, gerenciar e centralizar todas as informações referentes a sua execução. O programa disparador tem o trabalho de produzir, a partir do arquivo de configuração, as tabelas com todas as informações necessárias da aplicação, conectar ao servidor local e lhe enviar as tabelas, para que este possa iniciar a execução.

Quando a aplicação termina, o servidor mestre envia de volta ao disparador as informações sobre a execução, para que possam ser retornadas ao usuário.

3 Organização interna: o servidor do SPD

O servidor é o principal elemento do SPD. É através dele que os processos são disparados remotamente e que circulam todas as mensagens pelo sistema. Como mostrado na figura 1, existe um servidor em cada máquina da rede e eles são conectados entre si.

Esse programa é uma grande máquina de estados que, a cada ciclo, verifica a ocorrência de eventos e os trata apropriadamente. Pode-se dividir as funções do servidor em quatro grupos básicos, a saber: gerenciamento e disparo de processos, comunicação com processos, intercomunicação de servidores e roteamento de mensagens. Cada uma dessas funções é descrita a seguir:

3.1 Gerenciamento e disparo de processos

Esse subconjunto do servidor é o responsável pela interface com o disparador. Ele inclui as funções de inicialização e controle dos diversos processos que compõem cada uma das aplicações.

O servidor possui um porto especial (uma fila de mensagens) do IPC, *Interprocess Comunication* [6], através do qual recebe do disparador a descrição da aplicação que será executada, na forma de duas tabelas. A primeira contém todas as instâncias com as respectivas máquinas e o *path* onde serão encontrados os programas executáveis, e a outra contém todas as conexões existentes entre os canais das instâncias.

Assim que o servidor recebe uma aplicação, é iniciado um novo grupo de processos para o ambiente e este servidor recebe a denominação de servidor mestre daquela aplicação. Esse grupo recebe um identificador único na rede. É também distribuída, para as todas máquinas participantes da aplicação, a tabela de conexões dos canais, juntamente com as instâncias que serão executadas em cada uma.

De posse de todas as informações necessárias, cada servidor envolvido pode disparar os processos solicitados. Os processos da aplicação enviam mensagens informando o servidor do início e fim da utilização dos canais (`openspd()` e `closespd()`, respectivamente), permitindo o acompanhamento do estado atual de cada um deles. O recebimento de uma mensagem ou de um *signal* (morte do processo) inesperados configura, para o servidor, um erro no grupo. Caso aconteça, o procedimento seguido é o término dos processos locais da respectiva aplicação e o envio de uma mensagem de erro aos outros servidores envolvidos, para que também possam terminar a aplicação. Esta mensagem de erro garante a término de todos os processos do grupo e que o usuário será informado do erro observado.

Caso todos os processos terminem normalmente, enviando as respectivas mensagens de encerramento, o servidor anuncia ao servidor mestre o término normal, o qual, após receber esta notificação por parte de todos os servidores participantes, comunicará ao disparador o fim da execução.

3.2 Comunicação com processos

Essa parte do servidor tem como função realizar a troca de dados entre processos e servidor. Como pode ser visto na figura 1, processos só se comunicam com o servidor local. Por este motivo, a implementação dessas transações baseou-se em filas de mensagens (IPC).

O servidor possui uma fila de mensagens conhecida pela biblioteca do SPD que é usada no processo de comunicação local. Mensagens nesta fila são recebidas pelo servidor e identificadas através do PID (*Process ID* - número do processo retornado pelo UNIX quando da sua criação) do processo que envia. O servidor conhece o PID de todos os processos que está gerenciando, visto que todos foram disparados pelo próprio servidor. É através do PID que o servidor identifica o processo que enviou cada mensagem e é capaz de enviar mensagens para determinados processos.

3.3 Intercomunicação de servidores

A comunicação entre servidores é um pouco mais complicada que entre servidor e processo, já que os servidores estão em máquinas separadas. Para realizar essa comunicação, o servidor utiliza o protocolo TCP/IP e a biblioteca TLI.

O uso de um protocolo e uma interface padronizados é de grande valia, no sentido que facilita a portabilidade do sistema. O protocolo utilizado, TCP, exige que se estabeleça as conexões necessárias antes do início da comunicação.

Além de haver um procedimento para o estabelecimento de conexão, que deve ser implementado, existe um outro problema no que diz respeito ao número de conexões abertas simultaneamente, que é limitado pela implementação da TLI. Desta forma, não é possível, para todos os casos, o estabelecimento de conexões ponto a ponto entre todos os servidores executando na rede.

Em vista disso, o SPD deve manter um número limitado de conexões abertas simultaneamente, o que força uma certa disciplina no estabecimento das conexões, no sentido de evitar dois canais abertos entre o mesmo par de servidores. Além disso, quando se esgota o número limite de conexões abertas, o servidor deverá iniciar uma política de desconexão. Foi adotada a política LRU (*Least Recently Used*) para a escolha da conexão a ser encerrada pelo sistema. Ainda está em fase de estudos a ocorrência ou não de *trashing* na implementação realizada, com a política adotada estando sujeita a alterações.

3.4 Roteamento de mensagens

O roteamento das mensagens é feito baseado na tabela de conexões que o servidor recebe do disparador e é criada a partir do arquivo de aplicação. Um detalhe na implementação do servidor é que ele só conhece mensagens assíncronas. Desta forma, o protocolo de transferência síncrona é implementado na biblioteca do sistema.

Para ligações do tipo de um para um, o procedimento de roteamento é extremamente simples. O servidor apenas verifica o canal de origem da mensagem e, através de uma consulta na tabela, determina o canal para onde a mensagem deve ir. A partir daí, o servidor encaminha a mensagem para o *buffer* interno, referente ao canal destino. Se este estiver em outra máquina, o servidor envia a mensagem para o servidor remoto, que fará o encaminhamento da mensagem para o canal correto.

Porém, quando as relações de canal fonte para canal destino diferirem de um para um, podendo ser n para 1 ou 1 para n, o servidor irá posicionar a mensagem no buffer referente ao canal do lado um da relação, que pode ser tanto o origem quanto o destino. Desta forma, a busca das mensagens pela função da biblioteca é facilitada. Novamente, caso a mensagem tenha que ser transmitida para uma máquina remota, o servidor local irá transmitir a mensagem para o servidor remoto, que fará a alocação correta da mensagem.

No caso de conexões de n para n canais, o sistema irá concentrar as mensagens em um *buffer* conhecido por todos. A escolha do ponto de concentração é feito pelo disparador enquanto se está interpretando o arquivo da aplicação. O servidor já recebe as tabelas de roteamento prontas.

4 Resultados obtidos

Uma preocupação quando da avaliação do desempenho do sistema é a forma como este se comportaria diante de variações dos seus parâmetros básicos de execução: número de processos, número de máquinas, tamanho e quantidade de mensagens transmitidas.

Desta forma, procurou-se uma aplicação que permitisse alto paralelismo e uma fácil alteração dos parâmetros a serem avaliados. No presente trabalho, será usado uma aplicação que envolve a geração e exibição dos fractais do conjunto de Mandelbrot [1].

Todas as avaliações foram feitas com base em tempos reais de execução de cada uma das diferentes configurações da aplicação. Estas medidas de tempo foram usadas para calcular o *speedup* do sistema e para compará-lo com outro sistema similar, para este trabalho, o PVM [8].

4.1 A aplicação

Conforme descrito acima, a geração dos fractais de Mandelbrot envolve duas tarefas básicas, a geração e a exibição das imagens. Do ponto de vista de esforço computacional, estas duas tarefas são bastante diferentes, pois a primeira basicamente consiste de um grande número de cáculos envolvendo números complexos enquanto a segunda lida apenas com a interface que nesta implementação é uma aplicação *X-Windows*.

A adequação deste programa às implementações paralelas vem da total localidade de referência inerente ao algoritmo, pois o cálculo de um ponto do fractal não depende de nenhum outro dado a não ser as suas próprias especificações.

Para aproveitar esta característica do algoritmo, a aplicação foi dividida em dois programas:

xmandel: Este programa recebe os parâmetros do fractal a a ser gerado, especifica os quadros (subdivisões do fractal), os distribui. Ele também aguarda o retorno dos quadros calculados por parte dos processos geradores(vide a seguir) e os imprime na tela.

gmandel: Recebe os limites e resolução de um quadro a ser gerado e, após o cálculo, retorna o quadro a ser exibido para o servidor.

4.2 Ambiente de Simulação

Para todos os testes feitos, o servidor executou em uma *SUN Sparc station 2* e os processos geradores sempre executaram em máquinas *SUN Sparc station SLC* do DCC-UFMG. Em todos os testes, o tempo considerado exclui o tempo de exibição da imagem, levando em conta apenas o tempo de geração. Isso se deve aos atrasos do *X-Windows* sobre os quais não se tinha controle. Para as medidas, desconsidera-se então, a máquina que está rodando o processo xmandel.

4.3 Speedup

Para o cálculo do *Speedup*, usou-se uma implementação seqüencial do algorítmo executando-o em uma estação SLC. O resultado obtido na comparação das execuções pode ser visualizado no grafico 3.

Os parâmetros da execução dos programas foram: número de quadros de 30×30 para serem executados de cada vez. A versão seqüencial contava, obviamente, com apenas um gerador enquanto que a versão paralela contava com 8 processos em 8 máquinas diferentes (excluindo a nona máquina que está executando o xmandel, como em todos os exemplos apresentados). Foi variado o tamanho da janela total para a geração do fractal, assumindo os seguintes valores: 90×90, 150×150, 300×300, 450×450 e 600×600, obtendo-se então, um a variação na quantidade de mensagens transmitidas.

Figura 3:

4.4 Número de processos geradores

Pode ser observado no gráfico 4 a adequação do algoritmo a uma implementação paralela, pois a velocidade de execução daquela cresce com o aumento do número de processos geradores. Isto é válido tanto para o SPD quanto para o PVM, entretanto o primeiro tem um desempenho um pouco superior em todas as medidas. Isso se deve a outros parâmetros que serão tratados a seguir.

Todos os testes foram feitos mantendo a janela de 300×300 e o quadro para a troca de informações é de 30×30. Foi mantido um gerador por máquina e variado o número de geradores no total (conseqüentemente, o número de máquinas).

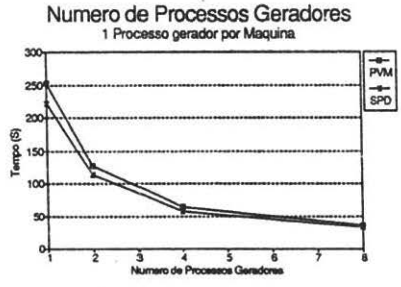

Figura 4:

4.5 Número de processos por máquina

Esta medida tem por finalidade avaliar a influência do número de processos da aplicação existentes por máquina, ou, no caso do SPD, o número de processos que o servidor local a cada máquina deve atender.

Pelo gráfico 5 percebe-se que quando se aumenta o número de processos por máquina o PVM se comporta melhor. Isso se deve ao fato de que todas as mensagens do SPD passam pelo servidor local, transformando-o em "gargalo" do sistema, enquanto no PVM, a conexão é feita diretamente com o processo destinatário [8]. Entretanto, quando são poucas as mensagens transmitidas, o SPD se comporta melhor, mostrando que o custo de transmissão do PVM é maior.

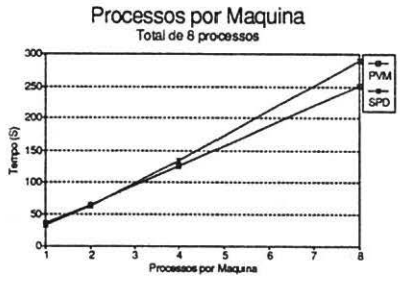

Figura 5:

4.6 Quantidade de Informação Transmitida na Mensagem

Aqui avalia-se o custo dos dois ambientes para realizar uma mesma aplicação variando-se o tamanho da mensagem. Cabe ressaltar que quanto menor a mensagem, maior será o tráfego destas para a realização da aplicação.

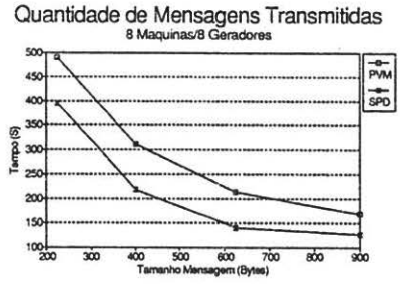

Figura 6:

Novamente pode-se observar pelo gráfico 6 a influência da passagem de todas as mensagens pelo servidor local no SPD, pois para mensagens maiores (e portanto em menor número), o SPD apresenta melhores resultados.

5 Conclusões e perspectivas futuras

Os resultados obtidos indicam um bom desempenho, justificando a continuação de experimentos e a adição de novas ferramentas ao sistema, para que o mesmo possa ter sua utilização mais difundida no meio acadêmico e, até mesmo, no meio comercial.

O SPD se mostrou bastante adequado ao desenvolvimento de aplicações distribuídas, devido à simplicidade na utilização dos recursos de comunicação oferecidos, juntamente com a facilidade e flexibidade de configuração, que pode ser modificada sem nenhuma alteração nos programas fonte.

Destacamos que este artigo não encerra o trabalho, mas o inicia. Como dito no início, esta é uma ferramenta de experimento e avaliação e ainda há muitas alternativas a serem pesquisadas.

Como primeiro tópico de melhoramento no sistema, podemos citar o desenvolvimento do servidor, no sentido de torná-lo mais robusto e mais tolerante a falhas, incluindo funções de tratamento e recuperação de erros, principalmente aqueles decorrentes da comunicação entre servidores.

O próximo passo será transformar o servidor em um *daemon*, podendo ser ativado durante o processo de *boot* das máquinas da rede. Assim, os servidores teriam permissão de superusuário, tornando-se capazes de executar diversas aplicações de diversos usuários simultaneamente.

O problema do *trashing* no esquema de desconexão de servidores deve ser estudado e corrigido, se necessário. Talvez, uma mudança na política de seleção ainda seja necessária. Um outro enfoque seria o sistema procurar rotas alternativas já existentes para enviar suas mensagens ao invéz de sempre tentar estabelecer uma conexão direta. Isso eliminaria o *overhead* de desconexão e conexão em troca de um caminho mais longo e um algorítmo de roteamento para as mensagens.

Um outro ponto de pesquisa e desenvolvimento do sistema seria a implementação de um esquema de relocação automática de processos nas máquinas da rede, baseando-se na carga corrente de cada uma. Desta forma, a alocação de processos ás máquinas seria feita dinamicamente, ficando a alocação feita pelo usuário opcional. Também, haveria um ganho de desempenho, visto que os processos estariam sempre executando nas máquinas com menor carga disponíveis na rede.

O SPD não oferece nenhum recurso para a centralização da entrada e saída dos diversos processos. Assim sendo, está restrita a utilização de processos interativos no sistema. Pode-se implementar um esquema de aberturas de janelas para cada um dos processos na máquina local (por exemplo, janelas *X-Windows*) de forma a permitir a interação com o usuário.

Aproveitando a idéia do *X-Windows* e interfaces amigáveis, uma outra proposta seria a de implementar uma interface amigável para a descrição dos arquivos de aplicação, ao invéz de utilizar o arquivo de aplicação. Este nova interface poderia oferecer recursos gráficos, facilitando a configuração das aplicações por parte do usuário.

Finalmente, um recurso que será imprescindível é um que permita depurar as aplicações. Deve-se implementar no sistema, ferramentas para reproduzir execuções paralelas e acompanhar a execução de processos. Este tipo de ferramenta é de utilidade inquestionável e tornará o SPD um sistema muito mais poderoso.

Referências

[1] Michael Barnsley. *Fractals Every Where.* Academic Press Inc., 1988.

[2] S. V. A. Campos and O. S. F. Carvalho. Têmis: um núcleo para o desenvolvimento de programas paralelos. In *Anais do X Congresso da SBC*, Vitória, ES, Julho 1990. Sociedade Brasileira de Computação.

[3] INMOS Corporation. *TRANSPUTER DATABOOK.* Second edition, 1989.

[4] Wagner Meira Jr., Márcio L. B. de Carvalho, and Osvaldo S. F. Carvalho. Programação paralela em arquiteturas transputers. Relatório Técnico 002/92 - UFMG, 1992.

[5] Dorgival Olavo Guedes Neto and Osvaldo S. F. Carvalho. Um núcleo "linda" para o desenvolvimento de aplicações distribuídas em uma rede unix. 10° *Simpósio Brasileiro de Redes de Computadores*, pages 573–585, Dezembro 1992.

[6] W. Richard Stevens. *Unix Network Programming.* Software Series. Prentice-Hall, Englewood Cliffs, New Jersey 07632, 1990.

[7] SUN microsystems. *Network Programming Guide*, revison a of 27 march edition, 1990.

[8] V. S. Sunderam. *PVM: A Framework for Parallel Distributed Computing.* Department of Math and Computer Science, Emory University, Atlanta.