

# Implementação de Gerenciador de Submissão de Tarefas em *Cluster* Educacional

Orientando: Luigi Jacometti de Oliveira

Orientador: Aleardo Manacero Jr.

Departamento de Ciências da Computação e Estatística  
IBILCE/UNESP

O ensino de computação de alto desempenho, mesmo em cursos de graduação, tem aumentado significativamente nos últimos anos. Esse crescimento se dá por duas razões principais: a primeira delas é o aumento na utilização de sistemas de alto desempenho em aplicações comerciais, enquanto a segunda razão está ligada ao surgimento dos *clusters* de PCs, que podem ser montados a um custo bastante baixo. No caso desses *clusters*, o custo final é definido pelos custos do hardware e do software (quase todo ele obtido a partir de plataformas de software livre e gratuito).

Os problemas com tais sistemas aparecem no momento de fazer um uso mais controlado de seus computadores, assumindo que o *cluster* será acessado remotamente por usuários diversos. Nessa situação pode haver disputa pelo acesso aos recursos do ambiente (desde os processadores até o suporte físico de comunicação), que é naturalmente de tempo compartilhado. Esse problema é facilmente resolvido em ambientes destinados à produção (entendida como processamento de tarefas que efetivamente buscam resultados para algum problema específico), através de ferramentas como Condor [1], Maui [2] ou Libra [3], uma vez que tais sistemas recebem geralmente um aporte maior de recursos de software.

O mesmo não ocorre em sistemas pequenos, destinados ao desenvolvimento de aplicações ou ao ensino de computação de alto desempenho, que é o caso examinado nesse trabalho. Nesses sistemas é difícil implementar o controle através das ferramentas mencionadas no parágrafo anterior por serem essas, em geral, bastante pesadas do ponto de vista de consumo de processamento, memória e discos. Isso demanda a implementação de soluções particulares a cada sistema, caso se pretenda garantir acesso privativo aos recursos do ambiente.

Embora sistemas de pequeno porte, não destinados à produção, não necessitem de um controle mais restrito sobre o compartilhamento de recursos do *cluster*, é interessante que exista alguma forma de controle para situações específicas. Essas situações incluem principalmente a execução de *benchmarkings* sobre os programas executados no *cluster*. Nesse tipo de situação a existência de compartilhamento compromete as medições realizadas, possivelmente invalidando qualquer estudo sobre *speedup*, escalabilidade ou qualquer outra medida de desempenho.

Torna-se evidente, portanto, que a implementação de um controle sobre o compartilhamento de recursos, que esteja adaptado ao contexto do ambiente, é desejável. Adicionalmente, uma ferramenta que seja dedicada permite ao administrador do *cluster* implementar controles sobre outros aspectos do sistema. Um desses aspectos é a separação física e lógica entre os sistemas de arquivos da rede em que o *cluster* está funcionando e os de cada uma de suas máquinas. Outro aspecto envolve a notificação dos usuários sobre a conclusão da execução das tarefas submetidas.

Nesse texto apresenta-se inicialmente um exame sobre as necessidades de gerenciamento em um *cluster* de aplicação educacional. Segue-se então com a descrição dos problemas para se atender tais necessidades e como os mesmos podem ser resolvidos. No final apresentam-se alguns resultados obtidos com a implementação de um aplicativo que atende as especificações indicadas nas próximas seções.

## 1 Processamento em *clusters* remotos

Como dito na seção anterior, o acesso de múltiplos usuários em um *cluster* remoto pode resultar, na falta de um gerenciamento explícito, no compartilhamento simultâneo dos recursos do sistema. Isso

implica que medições de desempenho realizadas nessas condições serão fortemente influenciadas pela concorrência entre os processos paralelos que estejam executando. Assim, torna-se essencial a existência de alguma forma de gerenciamento sobre o *cluster*, de forma a permitir que qualquer medição realizada seja feita livre de interferências.

De forma geral, ao gerenciar um *cluster* deve-se controlar as seguintes atividades:

- Serialização das tarefas, de forma a evitar a concorrência entre submissões;
- Controle sobre os sistemas de arquivos, evitando acessos desnecessários aos canais de comunicação, bem como necessidade de espaços muito grandes em disco;
- Notificações sobre a execução de cada tarefa, liberando o usuário de um controle *on-line*.

Ao controlar essas atividades é possível obter um uso bastante eficiente do *cluster*. A seguir apresenta-se mais detalhadamente a motivação para se implementar cada um desses controles.

### 1.1 Serialização de tarefas

O objetivo de uso de um *cluster* é a aceleração no processamento de tarefas através de sua paralelização, normalmente com o uso de bibliotecas de trocas de mensagens, como MPI ou PVM. Ocorre que nessas situações cada processo deve executar de forma exclusiva no sistema, uma vez que o compartilhamento do ambiente com outros processos reduz consideravelmente o tempo de CPU disponível para cada um deles.

Assim, como o objetivo é acelerar a execução dos programas, o acesso de cada um deles ao *cluster* deve ser feito de forma serializada. Para que isso ocorra o sistema deve oferecer, portanto, algum mecanismo que impeça a execução simultânea de dois programas no *cluster*. Isso pode ser obtido, de forma relativamente simples, através de um *daemon* que controle o disparo de processos no sistema.

### 1.2 Controle do sistema de arquivos

Um dos problemas no gerenciamento de usuários diversos dentro de um *cluster* é o tratamento do sistema de arquivos interno ao ambiente. Essa situação demanda por uma solução alternativa, em que se consiga simultaneamente uma redução na ocupação dos discos e no volume de comunicação necessário. Uma estratégia é separar os sistemas de arquivos em dois ambientes. Um, interno ao *cluster*, contendo apenas os recursos mínimos necessários para a execução dos programas paralelos, e outro, visível apenas pelo *front-end* do *cluster*, contemplando os arquivos de cada usuário.

Nessa estrutura, o ambiente de submissão de tarefas teria também a incumbência de gerenciar o seu sistema de arquivos interno.

### 1.3 Notificação de execução

Assumindo-se que o controle de submissão de tarefas trabalhe na forma especificada anteriormente, torna-se natural que ele também estabeleça um mecanismo para a notificação do usuário. Esse mecanismo de notificação é, em geral, estabelecido através de avisos por correio eletrônico, que é uma forma simples e eficaz de notificação. Nessa estrutura o controle de submissão deve ter a capacidade de geração de mensagens e acesso aos serviços de envio de mensagens que esteja estabelecido no *front-end* do sistema.

## 2 O controle de submissão

Para a implementação do par cliente-servidor adotou-se a comunicação através de *sockets*. O modelo geral de funcionamento desses programas aparece na figura 1. Nela é possível observar que tanto o cliente como o servidor foram estruturados em duas camadas, uma implementada através de *scripts*

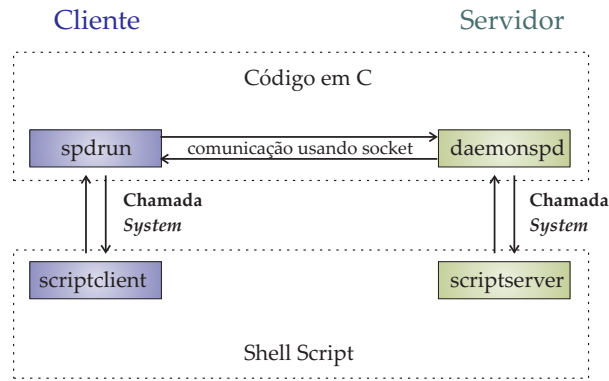


Figura 1: Modelo de comunicação do daemon de controle.

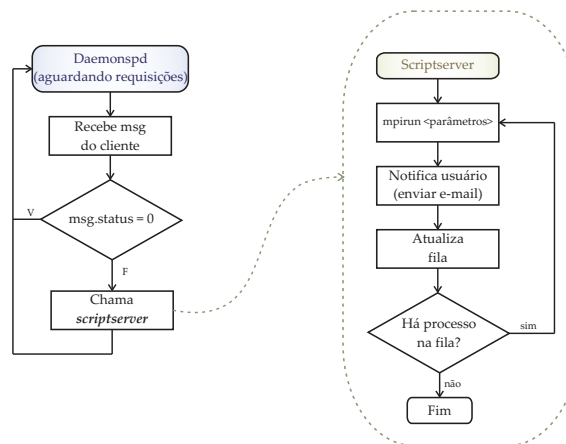


Figura 2: Fluxograma para o módulo servidor.

Unix e outra através de um par de programas em C, com comunicação via *sockets*. O funcionamento desse conjunto é explicado a seguir.

## 2.1 Funcionamento do sistema de controle

Como já mencionado, o controle é feito pelo programa servidor, a partir de chamadas feitas pelo cliente. O servidor atua como um processo *daemon*, que permanece à espera de requisições de clientes. Essas requisições são enviadas por meio da chamada do programa cliente, que faz a execução do *script* cliente.

Nesse *script* exige-se a passagem de dados (parâmetros de execução) para a construção de um arquivo de configuração. Esses parâmetros incluem a configuração a ser utilizada (quantidade de nós, quais máquinas serão envolvidas no processamento, o nome do arquivo executável, dos arquivos de entrada e saída, bem como o endereço eletrônico para notificação do usuário).

O programa cliente faz então uma requisição ao servidor, passando o arquivo de configuração e copiando os demais arquivos para uma área temporária no *front-end*. O módulo servidor inicia então a sua parte do trabalho, verificando se existe outro processo em execução. Quando não existe outro processo, o processo requisitado é imediatamente disparado. Caso contrário a requisição é inserida em uma fila de espera, sendo atendida na ordem temporal de sua chegada.

A figura 2 apresenta um fluxograma do funcionamento do módulo servidor para aplicações MPI. Para aplicações PVM basta mudar a execução do comando *mpirun*, que aparece em *Scriptserver*, para a chamada do código principal do programa (ele próprio em SPMD ou o mestre numa estrutura mestre-escravo).

No *script* do servidor, ao término da execução de um processo, os resultados são copiados para um diretório de saída, no mesmo sistema de arquivos do usuário, específico para o processo que foi

executado. Esse diretório é de acesso exclusivo do usuário que solicitou a execução de tal processo. Uma notificação é enviada por correio eletrônico informando o término da execução, a criação do diretório de resultados, bem como o caminho desse diretório no sistema de arquivos a que o usuário tem acesso.

### 3 Resultados

Um sistema de controle, na forma descrita na seção anterior, foi implementado para o controle de um pequeno *cluster Beowulf* construído para a condução de pesquisas em avaliação de desempenho e, simultaneamente, execução de projetos em disciplinas de graduação e pós-graduação com conteúdos que envolvam programação paralela. Esse *cluster* é composto por oito nós de processamento e um *front-end*, conectados por um *switch* padrão ethernet.

Para a verificação do funcionamento correto do controle de submissão foram executados testes diversos, em que se procurou verificar se os usuários obtinham efetivamente as funcionalidades previstas. Isso inclui a serialização e a notificação das submissões.

Em todos os testes realizados o controle se comportou da maneira prevista, disparando as tarefas submetidas na ordem e momentos corretos. As notificações, da mesma forma, ocorreram nos contextos específicos de cada um dos programas submetidos.

Por fim, a separação entre os sistemas de arquivos, bem como as cópias dos arquivos de entrada e os de resultados, foi verificada como correta em todos os casos. Não foram realizadas medições para verificar se a estratégia adotada permitiu uma redução efetiva nos tempos gastos na comunicação com o sistema de arquivos, mas não se entende ser esse um problema, uma vez que o controle de submissão não implica em atividades extras nesse mecanismo.

### 4 Considerações finais

Na realização desse trabalho foram examinadas diversas tecnologias para controle e comunicação de processos em ambientes Unix. Isso permitiu um bom aprendizado na programação com *sockets*, desenvolvimento de programas paralelos e configuração de *clusters Beowulf*.

O controle de submissão de tarefas aqui implementado permite aos usuários do *cluster* uma grande facilidade na execução de *benchmarks* de seus programas paralelos. Como uma das atividades principais desse grupo de pesquisas é a construção de ferramentas de avaliação de desempenho, torna-se importante que os programas consigam executar sem a interferência de outros processos em execução, o que é viabilizado por esse controle. Adicionalmente, também é facilitada a execução de programas por alunos que estejam aprendendo programação paralela, incluindo-se o *benchmarking* de seus programas, que é uma atividade importante no processo de aprendizagem.

### Referências

- [1] THAIN, D.; TANNENBAUM, T.; LIVNY, M. Condor and the grid. In: BERMAN ANTHONY J.G. HEY, G. F. F. (Ed.). *Grid Computing: Making The Global Infrastructure a Reality*. [S.l.]: Wiley, 2003.
- [2] INC., C. R. Maui cluster scheduler. [Http://www.clusterresources.com/products/maui/](http://www.clusterresources.com/products/maui/). 2005.
- [3] SHERWANI, J. et al. Libra: a computational economy-based job scheduling system for clusters. *Softw. Pract. Exper.*, John Wiley & Sons, Inc., New York, NY, USA, v. 34, n. 6, p. 573–590, 2004. ISSN 0038-0644.

**Bolsa:** Fapesp