Aplicação em Cluster com o SUN Grid Engine

Fernando Spencer spencer@cbpf.br
Marita Maestrelli marita@cbpf.br

Novembro 2002

Prefácio

Atualmente, o mundo vem voltando atenções para o processamento paralelo, cada vez mais, grandes empresas vêm estudando e pesquisando sistemas operacionais capazes de detectar e executar programas desenvolvidos de forma que possam ser fatiados, tendo esses pedaços executados simultaneamente.

Índice

1 – Introdução 3
2 – Instalação 5
2.1 – Diretórios 5
2.2 - Requerimento de espaço de disco 5
2.3 – Download 5
3 – Conceitos 6
3.1 – Batch 6
3.2 – Filas 7
3.3 - Processamento Paralelo 7
3.4 – Cluster 8
3.5 – Células 8
4 – Componentes 8
5 – Configuração 9
5.1 - Instalação de Contas 9
5.2 - Instalando o Master Host 9
5.3 - Instalando um Execution Host 9
5.4 - Instalando um Administration e Submit Hos 10
6 – Planejamento 10
7 – Comandos 11
8 – Execução 13
9 - Interface Gráfica 14
10 – Estatísticas 18
11 – Conclusão 19
12 – Referências 21
13 – Glossário 22
Anexo 1 - Programa para o cálculo de Π (Pi) 23

1 - Introdução

A *Grid*, rede de supercomputadores, pode, e vai se espalhar pelo mundo, lançando seus tentáculos em empresas, universidades e centros avançados de pesquisa. A grade, como está sendo chamada, parte do pressuposto de que o planeta tem um poder computacional ocioso que pode ser usado para implementar projetos e alavancar os negócios de micro-empresas. A *Grid* consiste numa imensa máquina virtual, com capacidade de processamento multiplicada pelo número de dispositivos que ela possuir. Um exemplo pode deixar mais claro do que é computação em grade é o projeto *Seti@home*, da universidade de Berkeley, consiste em usar computadores ociosos para pesquisas, a maioria das milhões de pessoas que baixaram o programa *peer-to-peer*, não sabe que ele é muito mais do que um mero *screen saver*. Depois de instalado, o software passa a usar o poder do micro e da rede na qual ele está alocado para buscar vida alienígena, transferindo as informações para uma central, que capta os dados e os decodificam.

A grande diferença entre o projeto *Seti@home* e a *Grid* é que, no primeiro caso, existe uma tarefa única sendo compartilhada por computadores. No caso da *Grid*, diversos blocos podem trabalhar com funções específicas, como por exemplo, pesquisas sobre o genoma humano. As máquinas recebem um pacote de tarefas e, dependendo da capacidade que necessitam para realizar a tarefa, buscam recursos computacionais em outras na rede *Grid*.

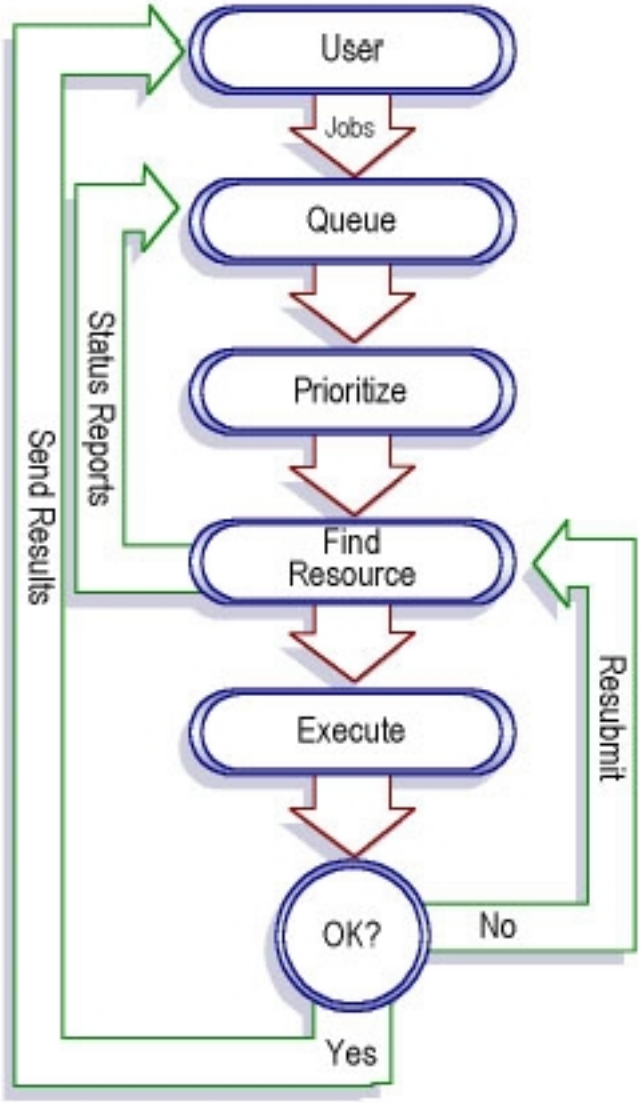
Embora a área científica seja apenas uma das pernas da *Grid*, o projeto foi concebido pensando nela. No Laboratório Nacional Argonne, nos EUA, foi desenvolvido uma rede que associasse laboratórios mundiais para troca de capacidade computacional e de conhecimento entre pesquisadores. Mas o projeto só decolou quando a ONG Globus adotou o modelo de grade.

Sun Grid Engine é um software que providencia um eficiente método de distribuição de processos por meio de múltiplos computadores, proporcionando um aumento na produtividade das máquinas e aumentando o número de processos que podem ser executados simultaneamente.

O *Sun Grid Engine* recebe pedidos para execução de processos dos usuários, colocando-os em uma área de espera (filas), até que eles possam ser executados.

Nesta área de espera são formadas as filas de acordo com a prioridade de cada processo, e estes vão sendo liberados à medida que haja recursos para serem executados.

A seguir o esquema de execução de um job no SGE:



Na prática, o *Sun Grid Engine* diminui o tempo de execução dos processos, dividindo-os em partes e distribuindo estes pedaços pelo cluster. Esta função chama-se paralelismo, caracteriza-se pela possibilidade de uma aplicação ser executada por mais de um processador ao mesmo tempo. Nesse tipo de ambiente, só haverá

ganho real de tempo caso a aplicação possa ser dividida em partes independentes para execução simultânea.

Um cluster pode ser formado por duas ou varias máquinas, cada máquina pode receber uma ou mais funções.

2 – Instalação

2.1 – Diretórios

É necessária a definição de um diretório, para a instalação e para uma melhor administração, recomenda-se o uso do mesmo diretório em todas as maquinas.

Por padrão, o SGE_ROOT será localizado no /gridware/sge. Caso isto seja alterado, os administradores e usuários deverão alterar a variável SGE_ROOT para a nova locação do sge.

2.2 - Requerimento de espaço de disco para os diretórios do *Grid*:

- 40 MB para o kit de instalação, incluindo a documentação, sem os binários.

- Entre 10 e 15 MB para cada binário, com exceção para arquitetura Cray, onde os binários costumam ter aproximadamente 35 MB.

Recomendamos providenciar espaço de disco para os *logs* do *Grid*.

- 30 - 200 MB para os diretórios de *spool*, dependendo do tamanho do cluster.

- 10 - 20 MB para cada máquina de execução.

2.3 – Download

O *Gridware* pode ser encontrado na *web*, http://www.sun.com/gridware, verificar a versão apropriada para sua máquina (Intel ou Sparc, 32 ou 64 bits).

Após o *download* é necessário descompactar os arquivos:

```
# gzip –d sge-5_3beta2-bin-solsparc32.zip

# gzip –d sge-5_3beta2-common.zip
```

Após a descompactação, deve-se instalar os binários com os seguintes comandos:

```
# pkgadd -d . SDRMsp32

# pkgadd -d .  SDRMcomm
```

3 - Conceitos

Para uma melhor compreensão do funcionamento do *Grid*, é necessária a revisão de alguns conceitos.

3.1 - Batch:

Quando um processo é executado em algum computador, pode ser feito de duas formas: Interativamente ou por batch.

Interativamente, significa executar o processo em tempo real, ou seja, conectar na máquina, rodar o programa e esperar o resultado. Este procedimento é ideal para *jobs* com respostas rápidas (ls, por exemplo).

Tratando-se de processos que levam tempo (mais de 3 minutos) para obterem uma saida, é interessante roda-los por meio de batch, ou seja, não executa-los diretamente, mas sim, enviando uma requisição ao computador para faze-lo.

Neste modo de processamento, os programas a serem executados são reunidos em grupos antes de serem processados. É gerada uma fila em

que os programas são liberados seqüencialmente pelo *Sun Grid Engine*. Durante o processamento em lote, o usuário não poderá interagir com a máquina e os resultados podem levar minutos, horas ou até mesmo dias.

Após o *job* ser submetido, não é necessário que o usuário fique conectado na máquina, esperando o resultado, pois será enviado por e-mail ou escrito em um arquivo.

3.2 - Filas

Conforme os processos são submetidos, de acordo com suas especificações e prioridades, são organizados em filas e executados em suas fatias de tempo. Todas essas definições são feitas pelo gerente do sistema.

3.3 - Processamento Paralelo

Na maioria das vezes, processamento paralelo é confundido com sistemas multitarefa, por mais que a impressão seja de que um computador esteja rodando dois programas ao mesmo tempo, na realidade não está, a não ser que este possua dois ou mais processadores trabalhando paralelamente, possibilitando a divisão de processos, porém, isto só será possível se o aplicativo que está rodando poderá ser fatiado em partes independentes para a execução simultânea.

O processamento paralelo é a possibilidade de uma aplicação ser executada por mais de um processador ao mesmo tempo, tanto em apenas uma máquina, quando em uma rede.
Caso um sistema possua 10 processadores e todos estejam dedicados à mesma aplicação, cada processador pode inicializar uma posição diferente de um programa ao mesmo tempo, tornando um processamento muito mais rápido que uma execução seqüencial.

O maior problema dos sistemas que implementam processamento paralelo é, justamente, determinar quando existe a possibilidade de implementação do paralelismo em um programa. O paralelismo pode ser indicado pelo próprio programador (paralelismo explícito) através de comandos da linguagem utilizada, onde ele determina as sub-rotinas que podem ser executadas em paralelo. Nesse método, o programador pode não perceber algum código possível de ser executado em paralelo,

sendo, também, comum o programa se tornar mais complexo e difícil de modificar.

Atualmente, muito esforço tem sido empreendido no desenvolvimento de compiladores e sistemas operacionais que detectam automaticamente o paralelismo nos programas (paralelismo implícito). Quando o paralelismo é determinado pelo próprio sistema, é provável que se obtenham programas mais eficientes e confiáveis.

Independentemente da forma com que o paralelismo é implementado, essa técnica é, ainda hoje, muito pouco utilizada, devido à característica puramente seqüencial das metodologias de programação e à grande dificuldade de depuração e prova de correção dos programas. Além disso, não surgiram linguagens de programação próprias para o desenvolvimento de aplicações paralelas, sendo utilizadas, atualmente, as linguagens Ada, Pascal, FORTRAN e Modula 2.

3.4 - Cluster

Um cluster é o conjunto de computadores interligados por rede e gerenciados por um software. O *Sun Grid Engine* vai tratar o cluster como se fosse um único computador com vários processadores, porém com memória distribuída.

3.5 – Células

Célula é o nome dado a cada computador, integrante do *Sun Grid Engine*, individualmente.

4 - Componentes

O *Sun Grid Engine* possui quatro componentes:

- Master Host: O master host é o centro das atividades do cluster, nele vai rodar o qmaster e o scheed, ambos daemons que controlam todos os componentes, como as filas e os jobs.

- Executin Host: O execution host é quem tem a permissão para executar os jobs, é nele que ficam hospedadas as filas.

- Administration Host: Este daemon vai controlar a parte administrativa, como, por exemplo, adicionar ou remover hosts de execução.

- Submit Host: No submit host é onde os usuários vão enviar os pedidos de execução dos jobs por meio de batch.

5 - Configuração

5.1 - Instalação de contas

Instalando o *Sun Grid Engine* como *root*, há a possibilidade de outros usuários poderem usar o *Grid*. Instalando como um usuário não privilegiado, as demais contas não terão acesso. Porém, a permissão de *root* é necessária para a instalação completa.

5.2 - Instalando o Master Host

Para instalar o *Master Host* é necessário estar conectado como *root*, ir ao diretório do SGE_ROOT e executar o seguinte comando.

```
# ./inst_sge –m
```

Durante a instalação o *Sun Grid Engine* fará algumas perguntas e talvez seja necessário executar alguns comandos administrativos.

O *Master Host* possuirá os recursos de administração e submissão.

Caso algo de errado ocorra, recomece a instalação.

5.3 - Instalando um Execution Host

Para instalar um *Execution Host* é necessário estar conectado como *root,* ir ao diretório do SGE_ROOT e executar o seguinte comando.

```
# ./inst_sge –x
```

O procedimento de instalação do Execution Host é similar ao do Master Host.

5.4 - Instalando o Administration e Submit Hosts

O *Master Host* está habilitado para ser também uma máquina de administração e submissão, monitorar e remover jobs. Não é necessária a instalação de mais componentes, porém, esta adição, pode ser feita com simples comandos:

```
# ./qconf -ah admin_host
```

```
# ./ qconf -as submt_host
```

Estes comandos precisam ser executados em um *host* de administração ou no *master*.

6 – Planejamento

6.1 - Decidir se o *Sun Grid Engine* será um simples cluster ou um conjunto de *sub-clusters*.

6.2 - Selecionar as máquinas que serão usadas e quais os seus tipos: *master, shadow master, administration, submit* e/ou *execution*.

6.3 - Verificar se todos os *ids* de usuários são os mesmos nos *executions* e *sumbmit hosts*.

6.4 - Decidir quais serviços de rede serão usados, NIS ou localmente em cada máquina no /etc/services.

7 – Comandos

Sun Grid Engine possui comandos para enviar e remover processos, monitorar e controlar filas. A seguir os exemplos:

. qacct:

Extrair informações dos logs do cluster.

. qalter:

Modificar atributos de um processo já enviado, porém, ainda não executado.

. qconf:

Providencia uma interface gráfica da configuração das filas.

. qdel:

Remove jobs.

. qhold:

Faz com que os jobs em execução, voltem para a fila de espera.

. qhost:

Mostra informações sobre os hosts de execução.

. qlogin:

Inicializa uma sessão similar ao telnet.

. qmake:

Funciona como o tradicional make, porém, tem a habilidade de distribuir independentes partes pelo cluster.

. qmod:

Permite ao dono suspender ou habilitar a fila.

. qmon:

Abre uma interface gráfica para monitorar o cluster.

. qresub:

Cria novos jobs copiados de outros que estejam na área de espera ou execução.

. qrls:

Reenvia jobs, que foram bloqueados pelo qhold, da área de espera para área de execução.

. qrsh:

Pode ser usado para vários propósitos, como providenciar uma execução remota de uma aplicação.

. qselect:

Aplicar ações em uma fila selecionada.

. qsh:

Abre um shell no xterm.

. qstat:

Mostra uma lista de todos os jobs e filas associadas ao cluster.

. qsub:

Interface para enviar um job ao cluster.

. qtcsh:

Shell compatível ao shell csh do Unix.

8 – Executando um job

Executando um simples *Job*

Para rodar um processo, primeiro é necessário que se ajuste o PATH e esta é a maneira mais fácil de fazer-lo.

```
# . sge_root_dir/default/common/settings.sh
```

Agora, pode-se tentar submeter o seguinte *script* de exemplo: (<SGE_ROOT>/jobs/simple.sh)

```
#!bin/sh
#This is a simple example of a Sun Grid Engine batch script
#
# Print date and time
date
```

```
# Sleep for 20 seconds
sleep 20
# Print date an time again
date
# End of script file
```

Use o seguinte comando:

```
# ./qsub simple.sh
```

9 – Interface Gráfica

O *Sun Grid Engine* possui uma interface gráfica para usuários acostumados com outros sistemas operacionais, como o Windows, facilitando a administração e execução de jobs. Para acessa-la execute o seguinte comando:

```
# ./qmon
```

A seguinte janela irá aparecer caso você esteja no modo gráfico (CDE, GNOME, etc).

Fig. 01

A partir do QMON, pode-se administrar o *Sun Grid Engine*, executar programas, adicionar hosts, etc.

Vejamos o Job Submission, é nele que selecionamos os jobs que irão para as filas, ajustar sua prioridade, hora de execução...

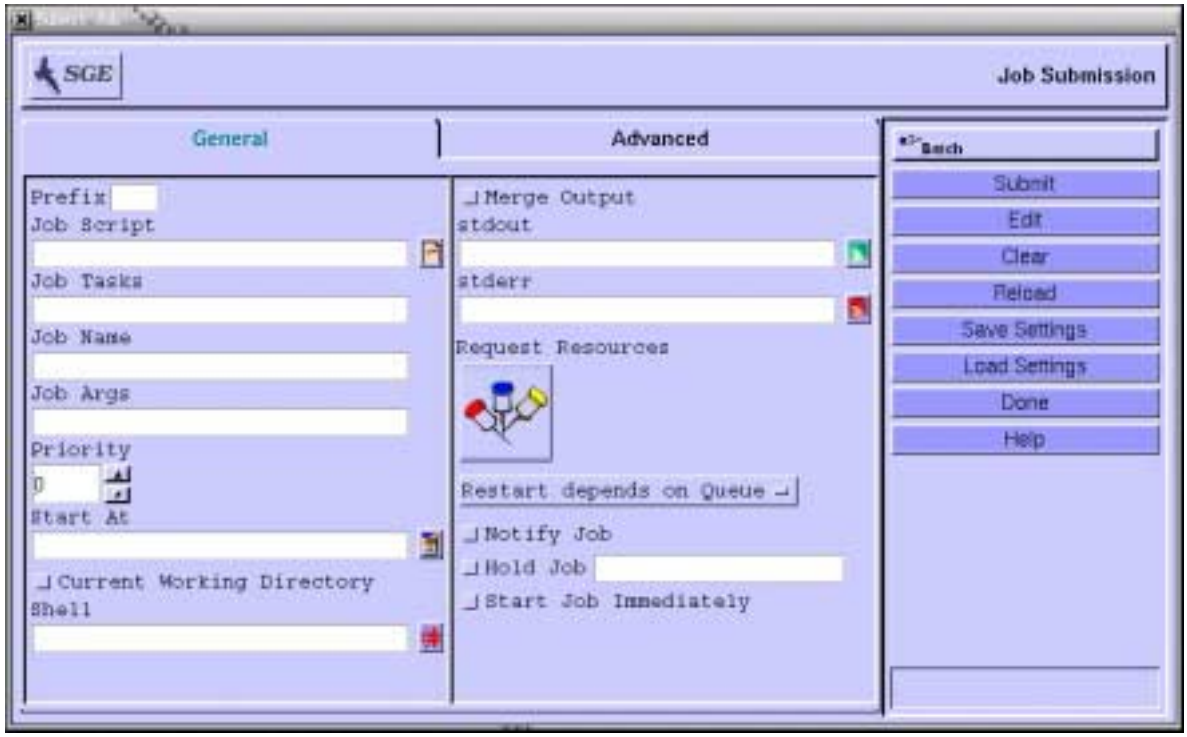


Fig. 02

o Host Configuration é uma das áreas mais importantes, é nele que controlamos as funções de cada máquina.

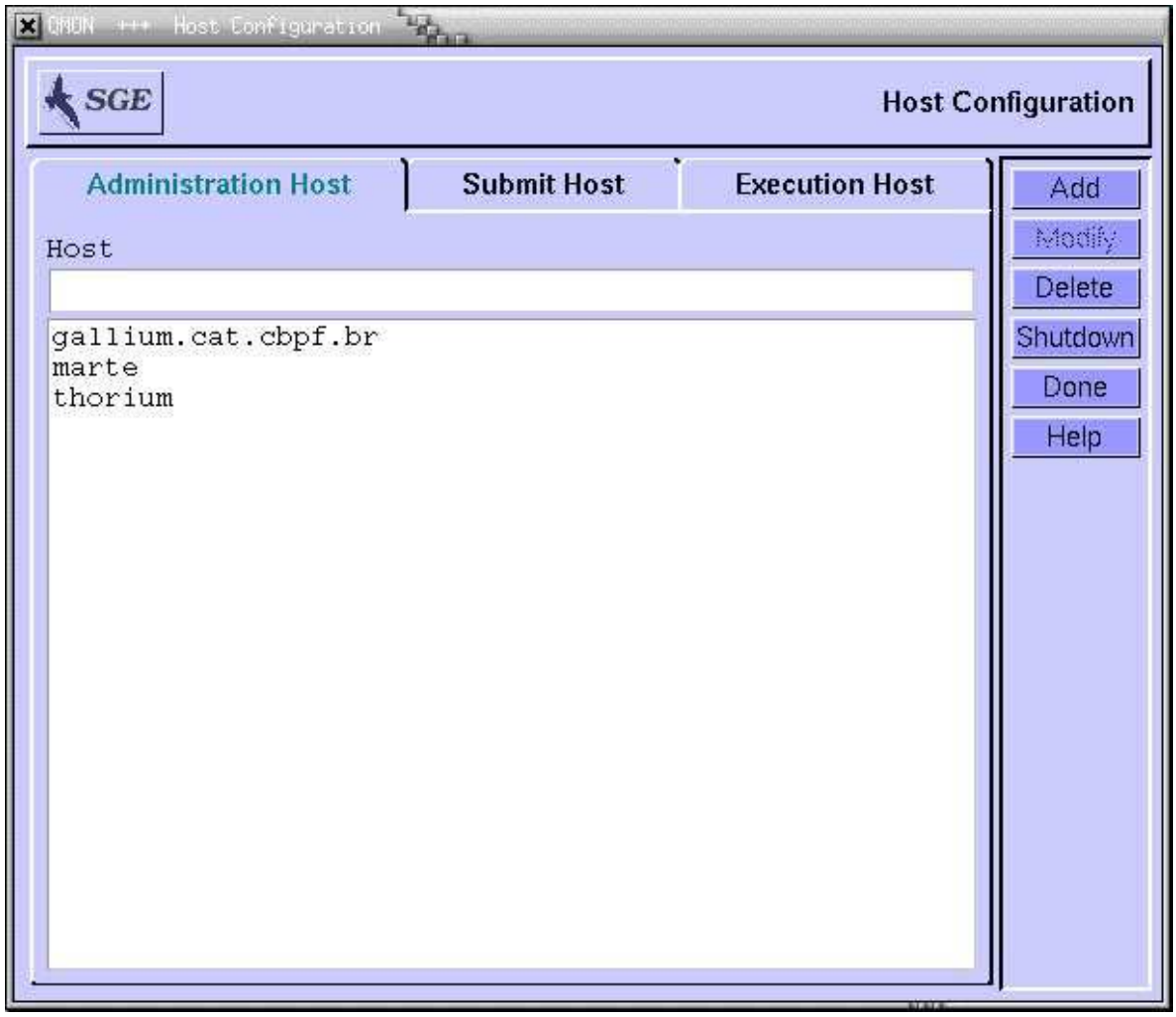


Fig. 04

O Queue Control é onde organizamos as filas e suas particularidades, como também podemos controla-las.

Fig. 05

O User Confuguration é a parte onde adicionamos os usuários que iram administrar, operar ou simplesmente usar o Grid.

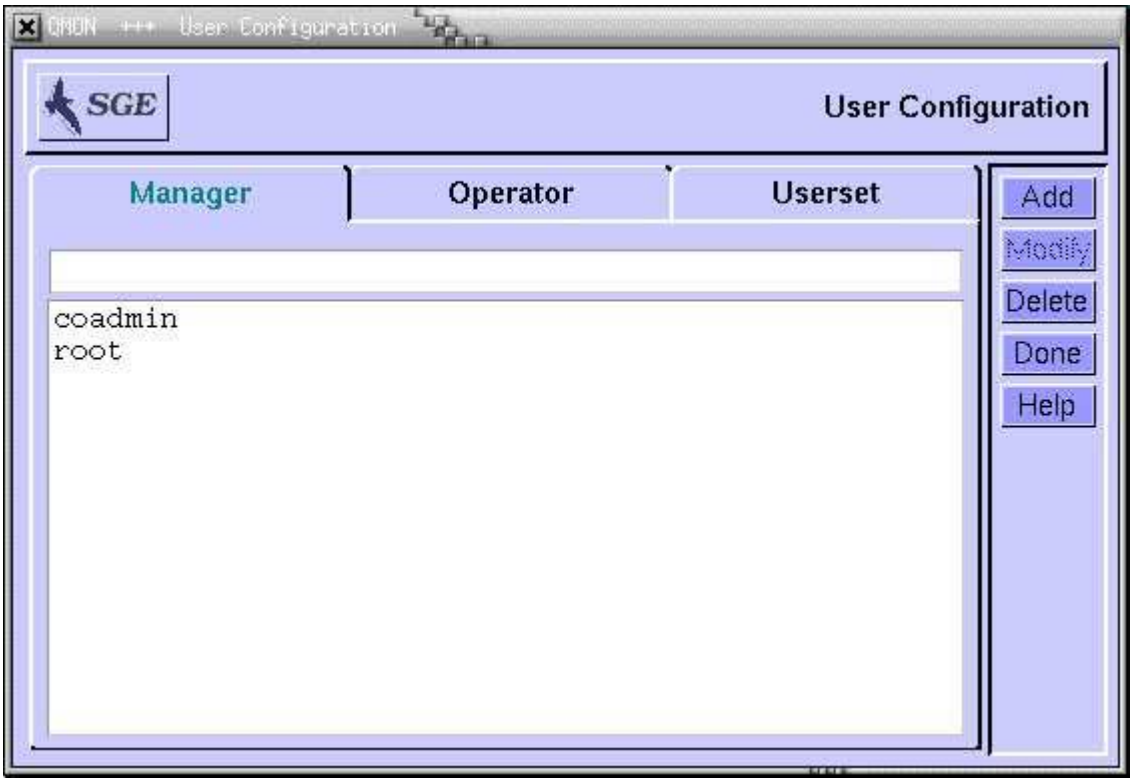


Fig. 06

10 – Estatísticas (sun.com/gridware – 09/2002)

- Aproximadamente 6000 instalações.

- Cada instalação tem em média 40 CPUs -> SGE controla ~240.000 CPUs pelo mundo.

- Cada site tem em torno de 1.8 masters executadando.

- 92% das pessoas consideram o SGE bom, ótimo ou excelente.

- 60% das pessoas usam esta tecnologia pela primeira vez.

11 – Conclusão

Em nosso estudo, utilizamos três máquinas do tipo *SPARStation*, com o sistema operacional Solaris 2.8 rodando em cada uma delas. Tinham em média 4GB de espaço em disco e 64 de ram.

Uma das máquinas era responsável por ser a *master* e *administration host,* sendo as outras duas apenas para execução de processos.

Estas máquinas estavam interligadas à rede do CBPF a 10 megabits por segundo de velocidade.

Afim de realizar testes, usamos um exemplo de MPI para calcular o valor do "PI" - Π. O programa fonte pode ser visto no anexo 1.

MPI é uma biblioteca de Message Passing desenvolvida para ambientes de memória distribuida, máquinas paralelas massivas, NOWs (network of workstations) e redes heterogênias.

MPI define um conjunto de rotinas para facilitar a comunicação entre processos paralelos.

A biblioteca MPI é portável para qualquer arquitetura, tem aproximadamente 125 funções para programação e ferramentas para se analisar a performance.

A biblioteca MPI possui rotinas para programas em Fortran 77 e ANSI C, e portanto pode ser usada também para Fortran 90 e C++. Os programas são compilados e linkados a biblioteca MPI.

Todo paralelismo é explicito, ou seja, o programador é responsável por identificar o paralelismo e implementar a algoritmo utilizando chamadas aos comandos da biblioteca MPI.

Comparação dos resultados:

1 - Processando em uma única máquina

Process 0 on marte.startrek.kirk
pi is approximately 3.1416009869231254, Error is 0.0000083333333323
wall clock time = 0.004504

2 - Processando em uma única máquina que possui 10 processadores

Process 0 on sol.startrek.kirk
pi is approximately 3.1416009869231254, Error is 0.0000083333333323
wall clock time = 0.000551

No caso 1, o processamento compartilha com os demais processos, enquanto que no caso 2 , o SGE escolhe o(s) processador(es) disponíveis e consequentemente o tempo de execução é bem menor.

12 – Referências

Sun Grid Engine – Manual do Grid.

http://www.sun.com/gridware - Home Page oficial do *Sun Grid Engine*.

MPI

http://www-unix.mcs.anl.gov/mpi/

http://www.mpi-forum.org/

http://www.erc.msstate.edu/misc/mpi/

Livros:

1 - Arquitetura de Sistemas Operacionais
Francis B. Machado
Luiz Paulo Maia
Segunda Edição
Livros Técnicos e Científicos Editora

2 - Introdução a Organização de Computadores
Mário A. Monteiro
Terceira Edição
Livros Técnicos e Científicos Editora

13 – Glossário

Master – Mestre, aquele que vai comandar.

Host – Qualquer máquina ligada à rede.

Submit – Submeter um programa para execução.

Job – Programa, processo, algo que vai ser ou está sendo executado.

Shadow – Sombra, a copia do original.

Célula – Ou host, qualquer máquina ligada à rede.

Grid – Conjunto de computadores ativos no cluster.

User – Um usuário qualquer cadastrado no sistema.

Queue – Fila de processos a serem executados.

Script – Programa em formato texto com comandos de Shell.

Shell – Local onde o usuário pode inserir comandos lidos pelo sistema.

PATH – Caminho de procura de binários.

Make – Comando para inicializar a compilação de um programa através de um script.

Telnet – Programa de comunicação entre dois computadores em modo texto.

Logs – Arquivos que guardam mensagens geradas pelo sistema ou por programas.

Root – Usuário com total acesso ao sistema.

ANEXO 1

Programa para o cálculo de PI - Π, usando linguagem C

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f(a)
double a;
{
   return (4.0 / (1.0 + a*a));
}

int main(argc,argv)
int argc;
char *argv[];
{
   int done = 0, n, myid, numprocs, i;
   double PI25DT = 3.14159265358979323846264 3;
   double mypi, pi, h, sum, x;
   double startwtime, endwtime;
   int  namelen;
   char processor_name[MPI_MAX_PROCESSOR_NAME];

   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);
   MPI_Get_processor_name(processor_name,&namelen);

   fprintf(stderr,"Process %d on %s\n",
           myid, processor_name);

   n = 0;
   while (!done)
   {
      if (myid == 0)
      {
/*
         printf("Enter the number of intervals: (0 quits) ");
         scanf("%d",&n);
*/
```

```
            if (n==0) n=100; else n=0;

            startwtime = MPI_Wtime();
        }
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
        if (n == 0)
           done = 1;
        else
        {
           h   = 1.0 / (double) n;
           sum = 0.0;
           for (i = myid + 1; i <= n; i += numprocs)
           {
              x = h * ((double)i - 0.5);
              sum += f(x);
           }
           mypi = h * sum;

           MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);

           if (myid == 0)
              {
              printf("pi is approximately %.16f, Error is %.16f\n",
                    pi, fabs(pi - PI25DT));
                    endwtime = MPI_Wtime();
                    printf("wall clock time = %f\n",
                          endwtime-startwtime);
              }
        }
     }
     MPI_Finalize();

     return 0;
}
```