INTERFACE WEB PARA GERENCIAMENTO E UTILIZAÇÃO DE CLUSTERS PARA PROCESSAMENTO PARALELO

Elaine Patricia Quaresma Xavier Lett

Dissertação apresentada ao Instituto de Física de São Carlos, Universidade de São Paulo, para obtenção do título de Mestre em Ciências "Física Aplicada – Opção Física Computacional"

Orientador: ***Prof. Dr. Gonzalo Travieso***

São Carlos

2003

USP/IFSC/SBI

8-2-001504

IFSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

UNIVERSIDADE
DE SÃO PAULO
Instituto de Física de São Carlos

Caixa Postal 369
13560-970 São Carlos, SP
Av. Trabalhador São-carlense, 400,
13566-590 São Carlos, SP

Fone/Fax 16 273 9777
www.ifsc.usp.br
secdepg@ifsc.usp.br

MEMBROS DA COMISSÃO JULGADORA DA DISSERTAÇÃO DE MESTRADO DE ELAINE PATRÍCIA QUARESMA XAVIER, APRESENTADA AO INSTITUTO DE FÍSICA DE SÃO CARLOS, DA UNIVERSIDADE DE SÃO PAULO, EM 17/02/2003.

COMISSÃO JULGADORA:

Prof. Dr. Gonzalo Travieso (Orientador) – IFSC/USP

Prof. Dr. Guilherme Matos Sipahi – IFSC/USP

Prof. Dr. Onofre Trindade Júnior – ICMC/USP

Aos meus pais Manoel e Maria Elena, pelo amor, paciência e compreensão

Agradecimentos

Ao Prof. Dr. Gonzalo Travieso, pela paciência, compreensão, confiança e pela excelente e competente orientação do trabalho, além do otimismo, que sempre se fez necessário.

Ao meu marido Alexandre, pelo constante apoio, companheirismo e paciência, mesmo nas horas mais difíceis.

À minha irmã Catarina por compreender minha distância, mesmo sem muitas vezes entender o motivo.
Aos meus pais Manoel e Maria Elena, pelo amor, apoio e paciência, e pelas oportunidades que me propiciaram.

Agradeço ao Bruno pelas discussões e longas conversas, que muito ajudaram na realização deste trabalho.

Agradeço a Thatyana, que se revelou uma grande amiga, companheira e confidente em todos os momentos, mesmo a distância. Agradeço à Eloiza e André, pela amizade. Agradeço a todas as pessoas que contribuíram para a realização deste trabalho, direta ou indiretamente.

Agradeço a CNPq pelo apoio financeiro

Sumário

INTRODUÇÃO 1

PROCESSAMENTO PARALELO 3

2.1 CLUSTERS 10
2.2 PORQUE UTILIZAR CLUSTER? 12

GERENCIAMENTO DE CLUSTER 15

3.1 SUPORTE A HETEROGENEIDADE 19
3.2 AMBIENTE BATCH 19
3.3 SUPORTE A COMPUTAÇÃO PARALELA 19
3.4 CHECKPOINTING 19
3.5 MIGRAÇÃO DE PROCESSOS 19
3.6 REESCALONAMENTO E MONITORAÇÃO DE PROCESSO 20
3.7 BALANCEAMENTO DE CARGA 20
3.8 ESTABELECIMENTO DE LIMITES 20
3.9 INTERFACE GRÁFICA COM O USUÁRIO (GUI) 20
3.10 FONTE ABERTA 21
3.11 INTERFACE WEB 21

SOFTWARES PARA CLUSTERS 22

4.1 SISTEMAS DE GERENCIAMENTO DE CLUSTERS 22
4.1.1 Portable Batch System (PBS) *23*
4.1.2 Distributed Queuing System (DQS) *24*
4.1.3 Condor *25*
4.1.4 Codine *26*
4.1.5 Connect:Queue *27*
4.1.6 Load Balancer *28*
4.1.7 LoadLeveler *29*
4.1.8. Load Sharing Facility (LSF) *29*
4.1.9 Task Broker *31*
4.2 AMBIENTES DE COMPUTAÇÃO EM CLUSTER 33
4.2.1 Amoeba *33*
4.2.2 Beowulf *34*

4.2.3 The Oxford BSP Library *35*
4.2.4 Distributed Computing Environment (DCE) *36*
4.2.5 Distributed Object Migration Environment (Dome) *37*
4.2.6 GLU Parallel Programming System *40*
4.2.7 Local Area Multicomputing (LAM) *41*

SOFTWARE PROPOSTO **42**

5.1 DESCRIÇÃO 42
5.2 IMPLEMENTAÇÃO 45

CONCLUSÃO **52**

Resumo

Este trabalho descreve um projeto simples de gerenciamento de *clusters* que apresenta uma interface de usuário para as tarefas mais comuns de uso e gerenciamento de um *cluster* utilizado como máquina paralela. A partir do estudo de vários softwares existentes hoje, o sistema projetado foi adequado às necessidades do *cluster* do Laboratório de Processamento Paralelo Aplicado do Instituto de Física de São Carlos. O sistema é baseado em páginas HTML e scripts CGI. O uso de HTML e CGI se demonstrou apropriado para o desenvolvimento desse tipo de sistemas.

Abstract

This work describes a simple cluster management system that operates as a user interface for some common user and manager tasks performed on a cluster used as parallel machine. We studied some cluster management systems from the literature and then designed a system with the needs of our research laboratory in mind. The system was implemented using HTML pages and CGI scripts. The use of HTML and CGI was found adequate for this type of systems.

Capítulo 1

Introdução

Processamento paralelo tem se mostrado uma ferramenta extremamente importante, e muitas vezes única, para a solução de problemas de várias áreas científicas. Isso porque o processamento paralelo permite um ganho no desempenho considerável se comparado ao desempenho quando se é utiliza apenas um processador.

Uma das formas disponíveis de utilização para processamento paralelo é o *cluster*, ou seja, um conjunto de máquinas interligadas em rede utilizado como um único recurso computacional. A principal desvantagem na utilização do *cluster* é o gerenciamento do mesmo, uma vez que ele é formado por diversos nós, isolados e tratados individualmente. Como os nós são tratados individualmente, qualquer comando ou programa a ser executado deve ser executado máquina a máquina. Também se torna trabalhoso o controle de acesso aos nós do *cluster*.

Existem duas possíveis soluções para este problema. A primeira é o desenvolvimento de um sistema operacional que gerencie as diversas máquinas envolvidas e apresente ao usuário uma visão única do sistema. A outra solução seria o desenvolvimento de um conjunto de ferramentas que sirvam de apoio aos programadores e usuários em suas tarefas.

O objetivo deste trabalho é mostrar que é possível realizar operação e gerenciamento de *clusters* por interface Web através de um software simples e facilmente adaptável às características próprias de cada *cluster* individual.

Este trabalho está dividido da seguinte forma:

O primeiro capítulo descreve o que é e a história do desenvolvimento de processamento paralelo, o que são *clusters*, suas vantagens e desvantagens.

O segundo capítulo tem por objetivo mostrar o que é gerenciamento de *clusters* e algumas características importantes a serem apresentadas pelos softwares utilizados para o gerenciamento do *cluster*.

O terceiro capítulo descreve alguns softwares utilizados para o gerenciamento de *clusters* e suas características, incluindo as citadas no segundo capítulo.

No quarto capítulo propomos um software de gerenciamento de *cluster* que pode ser desenvolvido com ferramentas simples e de fácil acesso.

Capítulo 2

Processamento paralelo

O *cluster* que iremos usar como modelo durante este trabalho é utilizado para processamento paralelo. É de extrema importância que o leitor saiba o que é processamento paralelo. Descreveremos abaixo o que é processamento paralelo e sua importância. Também apresentaremos um histórico do desenvolvimento de arquiteturas paralelas e seqüenciais. Não poderemos deixar de citar a importância de *cluster* e descrever para o leitor o nosso *cluster*.

O termo "processamento paralelo" é utilizado para definir a aceleração da execução de um programa através de sua divisão em múltiplos fragmentos que podem executar simultaneamente, cada um em um processador distinto. Portanto, um programa executado em N processadores poderia executar N vezes mais rápido [22].

A idéia de paralelismo vem desde o surgimento da computação, mas a tecnologia da época foi um grande obstáculo. Com o desenvolvimento de microprocessadores, o uso de paralelismo se tornou viável.

Um computador paralelo é um conjunto de processadores que são aptos a trabalhar cooperativamente para solucionar um problema. Essa definição envolve supercomputadores paralelos que possuem centenas ou milhares de processadores, redes de computadores, estações de trabalho de multiprocessadores e *cluster*s. Computadores paralelos são interessantes por oferecer o potencial para concentrar recursos paralelos - tais como, processadores, memória ou largura de banda I/O - em problemas computacionais importantes.

Não apenas a tecnologia é um aspecto relevante, mas também as aplicações. As primeiras aplicações que utilizaram paralelismo foram as numéricas, tais como previsões do tempo e simulação de clima. Entre as comerciais que utilizam o conceito de paralelismo, podemos citar os caixas eletrônicos e a produção de filmes de desenhos animados.

A tecnologia e as aplicações têm evoluído lado a lado. De um lado existem aplicações que precisam de uma tecnologia melhor. Então são criadas novas tecnologias para satisfazer tais necessidades. As aplicações são executadas e surgem mais aplicações que necessitam de melhor tecnologia.

Como os computadores tornam-se constantemente mais rápidos, pode-se tentar supor que eles eventualmente se tornarão rápidos o suficiente e que o interesse por aumentar o poder computacional será satisfeito dessa forma. No entanto, a história sugere que quando uma tecnologia particular satisfaz aplicações conhecidas, novas aplicações surgirão e demandarão o desenvolvimento de novas tecnologias.

Tradicionalmente, o desenvolvimento da computação de alto desempenho tem sido motivado por simulações numéricas de sistemas complexos tais como: previsão de tempo, clima, dispositivos mecânicos, circuitos eletrônicos, processos de fabricação e reações químicas. Entretanto, as forças mais significativas dirigindo o desenvolvimento de computadores mais rápidos atualmente são as aplicações comerciais que requerem que um computador seja capaz de processar grande quantidade de dados de modo sofisticado. Essas aplicações incluem vídeo-conferência, ambientes de trabalho colaborativos, diagnósticos auxiliados por computador em medicina, bases de dados paralelas usadas para apoio à decisão e imagens gráficas e realidade virtual de alta qualidade, particularmente na indústria de entretenimento. Por exemplo, a integração da computação paralela, rede de alto desempenho e a tecnologia de multimídia está levando ao desenvolvimento de servidores

de vídeo, computadores projetados para servir centenas ou milhares de pedidos simultâneos para vídeo em tempo real. Cada fluxo de vídeo pode envolver taxas de transferências de dados de muitos megabits por segundo e grande quantidade de processamento para a codificação e decodificação.

Em resumo, a necessidade de computadores mais rápidos está direcionada por uma demanda tanto de aplicações de dados intensivos no comércio quanto de aplicações de computação intensiva na ciência e engenharia. Progressivamente, as necessidades desses campos estão convergindo, quando aplicações científicas e de engenharia se tornam de dados intensivos e aplicações comerciais desempenham computações mais sofisticadas.

O desempenho dos computadores mais rápidos tem crescido exponencialmente desde 1945, proporcionalmente a um fator de dois a cada dezoito meses [17]. Enquanto os primeiros computadores realizavam poucas dezenas de operações de ponto flutuante por segundo, os computadores paralelos atuais atingem trilhões de operações por segundo. Tendências similares podem ser observadas nos computadores de baixo custo das diferentes épocas: calculadoras, computadores pessoais e estações de trabalho. Há pouco para sugerir que este crescimento não continuará. Entretanto, as arquiteturas de computadores usadas para sustentar este crescimento estão mudando radicalmente, do seqüencial para o paralelo.

O desempenho de um computador depende diretamente do tempo necessário para realizar uma operação básica e do número dessas operações básicas que podem ser realizadas concorrentemente. O tempo para realizar uma operação básica é, na verdade, limitado pelo ciclo de *clock* do processador, ou seja, o tempo necessário para realizar a operação mais primitiva. No entanto, a taxa de redução dos tempos de ciclo de *clock* está diminuindo lentamente e parece aproximar-se dos

limites físicos, tal como a velocidade da luz. Não podemos depender de processadores mais rápidos para oferecer aumento no desempenho computacional indefinidamente. Por exemplo, a execução de problema com 10^{16} operações num computador de 500 Mflops dura mais de sete meses. Precisamos esperar sete anos para ter computador capaz de executar este problema em 10 dias (de acordo com a Lei de Moore).

Os projetistas de computadores usam uma variedade de tecnologias para superar essas limitações no desempenho de computadores simples, incluindo o *pipeline* (diversos estágios de várias instruções executam concorrentemente) e múltiplas unidades funcionais (vários multiplicadores, somadores, etc, são controlados por um único fluxo de instrução). Cada vez mais os projetistas estão incorporando múltiplos computadores, cada um com seu próprio processador, memória e lógica de interconexão associada. Esta abordagem é facilitada pelos avanços na tecnologia VLSI que continua a diminuir o número de componentes requeridos para implementar um computador. Como o custo de um computador é proporcional ao número de componentes que ele contém, a integração aumentada também aumenta o número de processadores que pode ser incluído em um computador por um custo menor. O resultado é o crescimento contínuo no número de processadores em um mesmo sistema [17].

Outra importante tendência que está mudando a computação é o grande aumento nas capacidades das redes que conectam os computadores. Há pouco tempo, redes de alta velocidade funcionavam a 1.5 Mbits/s; atualmente, uma largura de banda acima de 1000 Mbits/s é muito comum. Melhorias significativas na confiabilidade também já são esperadas. Essas tendências tornam viável desenvolver aplicações que usam recursos distribuídos fisicamente como se eles fizessem parte do mesmo computador. Uma aplicação típica pode utilizar processadores em múltiplos computadores remotos, acessar uma seleção de base de dados

remota, realizar interpretação em um ou mais computadores gráficos, e oferecer saída em tempo real e controle na estação de trabalho.

É interessante enfatizar que a computação em computadores em rede (computação distribuída) não é apenas uma subárea da computação paralela. A computação distribuída está profundamente relacionada com problemas tais como confiabilidade, segurança, heterogeneidade, que são geralmente considerados como tangenciais em computação paralela. A tarefa básica de desenvolvimento de programas que podem executar em vários computadores de uma vez ainda é um problema da computação paralela. O aumento no uso de *clusters* e *grid computing* tem provocado uma convergência entre estes dois campos.

A rápida penetração dos computadores no comércio, ciência e educação deve muito à primeira padronização de um modelo de máquina simples, o computador de von Neumann. Um computador de von Neumann compreende uma unidade central de processamento (CPU) conectada a uma unidade de armazenamento (memória). A CPU executa um programa que especifica uma seqüência de operações de escrita e leitura na memória. Este modelo simples tem provido robustez notável. Sua persistência por mais de quarenta anos tem permitido o estudo de tópicos importante como algoritmos e linguagens de programação. Conseqüentemente, os programadores podem ser treinados na abstrata arte de programação ao invés do ofício de programação numa máquina específica qualquer e podem projetar os algoritmos para uma máquina de von Neumann abstrata, convictos de que esses algoritmos executarão na maioria dos computadores com razoável eficiência.

O estudo de programação paralela será mais compensador se nós pudermos identificar um modelo de máquina paralela que seja tão geral e útil quanto o modelo de máquina seqüencial de von Neumann. Este modelo de máquina deve ser simples e realístico para assegurar que

programas desenvolvidos para o modelo executem com razoável eficiência em computadores reais.

Um modelo de máquina paralela, chamado de multicomputador, preenche esses requisitos [22]. Um multicomputador compreende um número de computadores de von Neumann ou nós interligados por uma rede de interconexão. Cada computador executa seu próprio programa. Este programa pode acessar a memória local e pode enviar e receber mensagens pela rede. As mensagens são usadas para comunicação com outros computadores ou equivalentemente ler e escrever em memórias remotas. Na rede idealizada, o custo de emissão de uma mensagem entre dois nós é independente da localização do nó e outro tráfego de rede, mas depende do comprimento da mensagem.

Um atributo do modelo de multicomputador é que acessos a memória local (mesmos nós) são menos custosos que acessos a memória remota (nós diferentes), isto é, leitura e escrita são menos custosos que enviar e receber. Portanto é desejável que acessos a dados locais sejam mais freqüentes que acessos a dados remotos.

O multicomputador é muito similar aos computadores com arquiteturas MIMD (*Multiple Instruction Multiple Data*) de memória distribuída. Este tipo de arquitetura envolve múltiplos processadores executando de forma autônoma diferentes instruções em conjuntos de dados distintos. Exemplo de computadores que se utilizam desta arquitetura: SP2 da IBM e Cray T3D da Cray. Essa classe inclui máquinas com processadores e unidades de memória especificamente projetados para serem componentes de uma arquitetura paralela, máquinas paralelas construídas com processadores comerciais, computadores simples como o IBM PC interconectados por uma rede local e sistemas distribuídos. A principal diferença entre um multicomputador e computador MIMD de memória distribuída é que no

segundo o custo de emissão de uma mensagem entre dois nós não é independente da localização do nó e do tráfego de rede.

Outra importante classe de computadores paralelos é o multiprocessador ou computador MIMD de memória compartilhada. Neste, todos os processadores compartilham o acesso a uma memória comum, tipicamente, via barramento. Assim o tempo de acesso a dados é constante entre os nós do multiprocessador. Os programas desenvolvidos para multicomputadores também podem ser executados eficientemente em multiprocessadores, pois a memória compartilhada permite uma implementação eficiente da passagem de mensagem. Exemplos de computadores com este tipo de arquitetura: Challenge da Silicon Graphics e Cray Y-MP, da Cray.

Duas classes de sistemas de computadores que são usados como computadores paralelos são as redes locais (LAN) e as redes não locais (WAN). Embora os sistemas deste tipo introduzam preocupações adicionais tais como confiabilidade e segurança, eles podem ser vistos para muitos propósitos como multicomputadores com alto custo de acesso remoto.

O multicomputador é um modelo adequado para a grande maioria das arquiteturas paralelas existentes, desde que a aplicação não gere uma quantidade muito grande de comunicações ou tarefas muito pequenas (o que em qualquer caso seria deletério para o desempenho da aplicação nas arquiteturas atuais). Espera-se, portanto, que programas desenvolvidos para esse modelo tenham boa eficiência em uma ampla gama de arquiteturas atuais, incentivando o desenvolvimento de programas paralelos pela sua maior portabilidade.

IFSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

2.1 *Clusters*

Agora que já sabemos o que é processamento paralelo, quais os tipos de máquinas disponíveis para este tipo de processamento e sua importância, descreveremos o que é um *cluster.*

Considerando o crescimento exponencial na velocidade de execução, verificado nos últimos anos, dos processadores utilizados em computadores pessoais, associado ao significativo crescimento na largura de banda de redes de interconexão comerciais nesse mesmo período, o uso de *clusters* para execução de tarefas de processamento paralelo tem se tornado cada vez mais freqüente. *Clusters* aparecem em várias formas: *clusters* de alto desempenho, *clusters* de alta disponibilidade, *clusters* dedicados, *clusters* não-dedicados, entre outros. É possível "montar" *clusters* com uma grande variedade de recursos, como supercomputadores, sistemas de armazenamento, recursos de dados e classes especiais de dispositivos geograficamente distribuídos e usá-los com um recurso único[31].

Um *cluster* é um tipo de sistema paralelo ou distribuído que consiste de uma coleção de computadores interligados; é utilizado com um único e unificado recurso de computação.

Também é possível a utilização de *Grid Computing* [6], que tem por objetivo o uso cooperativo de recursos distribuídos geograficamente formando um único computador poderoso. Neste caso, é utilizada também a Internet como recurso computacional.

Neste trabalho enfocamos o uso de *clusters* fechados como sistemas de processamento paralelo, tendo em vista o destino principal do *cluster* adquirido por nosso grupo de pesquisa.

Várias são as razões para o sucesso de *clusters*:

- Um *cluster* oferece um ambiente agradável para pesquisas em computação paralela. Antigamente, programação paralela só podia ser realizada em computadores caros (propriedades aqui significam sistemas tais como CMS, Paragon, T3D, e outros).
- Ciclos de computação livres podem ser utilizados, os quais oferecem capacidades de computação adicionais sem custo.
- *Clusters* aumentam o fator custo/benefício de desempenho por utilizar sistemas de microprocessadores.
- Os sistemas de software robustos/estáveis da primeira geração para *cluster* estão geralmente disponíveis para acesso público, dos quais a maioria é encontrada via Internet. Neste caso classificamos estes softwares como softwares de acesso público. Esta característica será descrita no próximo capítulo.

Clusters oferecem uma flexibilidade única para protótipos e experiências com programação paralela. Antigamente, para utilizar programação paralela era preciso investir em computadores desenvolvidos e comercializados para este propósito específico e com custo elevadíssimo. Agora, qualquer pesquisador com acesso a múltiplas estações de trabalho pode desenvolver e fazer experiências com programas paralelos. Tanto empresas quanto laboratórios de pesquisas podem ser configurados como um ambiente de computação paralela. Duas abordagens são utilizadas para implementar paralelizações em um cluster:

- Conceder às linguagens seqüenciais existentes tais como C e FORTRAN tratar as necessidades de comunicação e sincronização (por exemplo, passagem de mensagens, memória compartilhada distribuída, etc.), através de bibliotecas especiais.

- Definir novas linguagens de programação baseadas em paradigmas lógicos, funcionais ou orientados a objeto.

2.2 Porque utilizar *cluster*?

A utilização de *cluster* para processamento paralelo oferece importantes vantagens:

- Cada uma das máquinas no *cluster* pode ser um sistema completo, útil para um grande conjunto de outras aplicações de computação. Isso leva muitas pessoas a sugerir que *clusters* para computação paralela podem simplesmente aproveitar todos os ciclos gastos em estações de trabalho que permaneceram ociosos.
- A explosão corrente em sistemas de rede significa que a maioria dos hardwares para construção de um *cluster* está sendo negociada em grandes volumes, com preços razoavelmente baixos como resultado. Como comparação, podemos lembrar que sistemas SMP e de processadores anexos são um mercado menor, tendendo a ter altos preços com um desempenho menor.
- *Clusters* são altamente escaláveis. Enquanto é difícil encontrar sistemas SMP com mais de quatro processadores, os hardwares de rede disponíveis para construir um *cluster* com mais de 16 computadores são facilmente encontrados. Com pouco trabalho, centenas ou até milhares de máquinas podem se tornar uma rede. Portanto, a Internet pode ser vista como um enorme *cluster*.

A primeira vista, imaginamos que *clusters* são as melhores soluções para processamento paralelo. Mas eles também apresentam problemas:

- Com poucas exceções, os hardwares de rede não são projetados para processamento paralelo. Tipicamente apresentam alta latência e largura de banda relativamente baixa comparada às dos sistemas SMP e de processadores anexos. Por exemplo, a latência de um SMP não é geralmente maior que poucos microssegundos, mas é normalmente centenas ou milhares de microssegundos quando falamos em latência de um *cluster*. A largura de banda de comunicação de SMP é freqüentemente mais que 100 Mbytes por segundo, enquanto que a conexão de rede ATM mais rápida é cinco vezes menor.
- Existem poucos softwares para *clusters* comparados aos softwares existentes para sistemas SMP.

Mas uma das principais desvantagens da utilização de *clusters* para processamento paralelo, tanto para usuários como para programadores, provém do fato de que eles se constituem de diversos nós que aparecem como recursos isolados que, portanto, devem ser tratados individualmente, complicando o processo de desenvolvimento e controle da execução dos programas. Isto é uma decorrência do fato de que os diversos elementos de hardware e software do sistema foram projetados com vista ao uso como sistemas individuais, interligados à Internet.

Uma solução para este problema é o desenvolvimento de um sistema operacional que gerencie as diversas máquinas envolvidas e apresente ao usuário uma visão única do sistema. Esta solução, no entanto, é muito custosa em termos de esforço de desenvolvimento e

apresenta a desvantagem adicional de tornar o comportamento do sistema paralelo durante a execução de um programa menos previsível (em termos de desempenho), devido às sobrecargas introduzidas para possibilitar o gerenciamento dos recursos por parte do sistema operacional distribuído.

Uma outra solução é o desenvolvimento de um conjunto de ferramentas que sirvam de apoio aos programadores e usuários em suas tarefas. Estas ferramentas poderiam ser um conjunto de programas (ou *scripts*) que se devem apresentar uma interface única a partir da qual o sistema paralelo possa ser operado, sem necessidade de conexão em máquinas remotas, anotação de números de identificação de processos, etc.

Junto com a popularização dos *clusters* estão surgindo pacotes de sistemas de gerenciamento de clusters, para apoio ao usuário e gerente do sistema, ajudando também no balanceamento de carga, aproveitamento de ciclos de CPU livres, oferecimento de sistemas de tolerância a falhas, gerenciamento de acessos do usuário ao sistema, etc.

Alguns desses sistemas serão discutidos no próximo capítulo.

Capítulo 3

Gerenciamento de *cluster*

Já foi mencionado a grande dificuldade de se trabalhar com *cluster* que é o gerenciamento das máquinas como um sistema único. Para solucionar este problema, uma das soluções é desenvolver um software com esta finalidade. Mas para desenvolver este software algumas perguntas devem ser respondidas:

- Que tipo de software será desenvolvido?
- Quais as suas características?

Para responder estas perguntas, descreveremos abaixo quais os dois grupos de softwares e quais as características relevantes.

Podemos classificar os softwares de gerenciamento de *Clusters* em dois grupos:

- *Cluster Management Software* (CMS) - este tipo de software é projetado para administrar e gerenciar processos submetidos em *clusters* de estações de trabalho. Ele abrange o tradicional *batch* e sistemas de filas.
- *Cluster Computing Environments* (CCE) - com este tipo de software o *cluster* é tipicamente utilizado com um ambiente de aplicação, similar aos sistemas de memória compartilhada distribuída.

Para entender softwares de *cluster* é necessário saber como eles interagem com o sistema operacional de uma plataforma particular.

Os CMS trabalham completamente fora do Kernel e em cima do sistema operacional existente nas máquinas. Isto significa que sua

instalação não requer modificações no kernel; basicamente o CMS é instalado como qualquer outro pacote de software na máquina.

A situação é muito diferente com o CCE. Neste caso, tipicamente, um micro kernel com serviços básicos customizados é instalado ao invés do kernel existente para suportar o ambiente desejado. A razão para esta solução mais radical é a necessidade de apoiar funcionalidades como memória virtual compartilhada. Isto não pode ser alcançado eficazmente fora do kernel.

Existe um número significativo de Softwares de gerenciamento de *clusters* (CMS). A maioria destes softwares se originou de projetos de pesquisa, mas muitos foram adotados por representantes comerciais. Ainda não está evidente se os CMS estão sendo utilizados para tirar proveito dos ciclos de CPU excedentes, mas é evidente o esforço em melhorar o balanceamento de carga. A maioria dos softwares de CMS é projetada para ser executada em *clusters* que tenham como sistema operacional Linux e sistemas de MPP. Alguns softwares de domínio público podem ser executados em PCs. O Codine, um software comercial, que será descrito posteriormente, pode ser executado em plataforma Linux. JP1 da Hitachi Ltd. e é desenvolvido para ser executado em plataformas Windows NT.

Podemos dizer que sistemas para *clusters* e softwares para administrar e programar aplicações para executar nestes sistemas estão se tornando comuns. Pacotes de CMS surgiram por várias razões, entre elas balanceamento de carga, utilização de ciclos de CPU excedentes, oferta de sistemas de tolerância a falhas, gerenciamento de acessos ao sistema e assim por diante. Mas acima destas razões, a principal razão para a existência destes softwares é a sua habilidade em oferecer um aumento de processamento das aplicações dos usuários no sistema que ele gerencia.

Para executar um processo em um sistema *batch* CMS é normal gerar algum tipo de arquivo de descrição de recurso. Este arquivo normalmente é um arquivo texto ASCII que contém um conjunto de palavras chaves para ser interpretado pelo CMS. A natureza e o número de palavras chaves disponíveis depende do pacote de CMS, mas inclui pelo menos o nome do processo, o tempo máximo de execução e a plataforma desejada.

Uma vez completo, o arquivo de descrição do processo é enviado para o software cliente numa estação de trabalho do usuário, para um gerente de agendamento. O gerente de agendamento, como o próprio nome diz, é a parte do CMS que tem por tarefa verificar os recursos disponíveis do *cluster*: as filas que podem ser configuradas e a carga computacional no *cluster* que ele está administrando. Em cada uma das máquinas do *cluster*, *daemons* estão presentes para comunicar seu estado em intervalos regulares para o gerente de agendamento. Uma das tarefas do gerente de agendamento é eventualmente balancear a carga nos recursos que ele está gerenciando. Assim, quando um novo processo é submetido ele não apenas tem que comparar os recursos pedidos com os que estão disponíveis, mas também precisa assegurar que os recursos a serem utilizados estão com suas cargas balanceadas.

Tipicamente, um sistema *batch* possui múltiplas filas, cada uma apropriada para diferentes tipos de processo. Por exemplo, uma fila pode ser configurada para um *cluster* homogêneo que é primeiramente utilizado para executar processos paralelos; alternativamente filas podem ser configuradas num servidor poderoso para processos intensos de CPU, ou pode haver uma fila para processos que precisam de um retorno rápido. O número de configurações possíveis de fila é grande e irá depender do processamento típico de processo no sistema que será utilizado.

O gerente de agendamento também é responsável por assegurar que os processos serão completados com sucesso. Ele faz o monitoramento dos processos até que eles terminem com sucesso. No entanto, se um processo falha, devido a diferentes problemas em tempo de execução, cabe ao gerente replanejar a execução do processo novamente.

Alguns CMSs possuem características semelhantes às de um *Middleware*. *Middleware* é considerada uma camada de software entre o sistema operacional e as aplicações. Os *Middleware* oferecem vários serviços necessários para que uma aplicação funcione corretamente. Nos últimos anos, os *middlewares* estão sendo integrados com aplicações de softwares para serem executados em ambientes heterogêneos. O *Middleware* também é um software que reside sobre o kernel e oferece serviços para aplicações. Ele tem a habilidade de ajudar aos desenvolvedores de aplicações para ambientes heterogêneos na solução de problemas, tais como, a integração do software de diferentes origens, acesso ao software interno e externo e desenvolvimento rápido de aplicações. No entanto, os serviços que o *middleware* oferece não estão restritos a desenvolvimento de aplicações. Eles também oferecem serviços para o gerenciamento e administração de sistemas heterogêneos. A característica especial dos CMS é que eles são projetados para administrar e gerenciar processos submetidos em *clusters* de estações de trabalho e não apenas para integrar diferentes sistemas.

Descreveremos agora algumas características importantes para um CMS. Os critérios aqui descritos são baseados nos critérios originais levantados por Kaplan e Nelson[31], com adições para refletir as visões e experiências com novos sistemas.

3.1 Apoio à heterogeneidade

Um *cluster* heterogêneo consiste de alguns computadores de diferentes arquiteturas executando diferentes sistemas operacionais. Para ser utilizado nos ambientes computacionais atuais, um sistema de *cluster* deve suportar diferentes arquiteturas e sistemas operacionais.

3.2 Ambiente *batch*

Um processo *batch* é definido como um *shell script* contendo comandos que podem ser executados sem a intervenção do usuário. Ambientes *batch* oferecem as ferramentas de software necessárias para permitir ao usuário submeter, monitorar e manter os processos *batch*.

3.3 Apoio à computação paralela

Oferecer apoio para executar programas paralelos no *cluster*, por exemplo, PVM, MPI ou HPF. Esta característica é mais relevante para *clusters* de pesquisa que para *clusters* comerciais. Entretanto, apoio a computação paralela irá freqüentemente significar que o CMS é mais flexível onde ele for configurado que um CMS que suporte apenas processos seqüenciais.

3.4 *Checkpointing*

Habilidade para criar um arquivo de reinicialização contendo todas as informações necessárias de estado para reinicializar o processo quando ocorrer uma interrupção.

Esta característica é útil em caso de falha enquanto o processo está sendo executado. Mas não é vantagem utilizá-la para processos pequenos, devido ao seu custo.

3.5 Migração de processos

Habilidade para mover um processo de um sistema de computador para outro sem reinicializar o processo.

Como o *checkpointing*, migração de processos pode ser uma característica muito útil para os CMS. Tipicamente ela é usada para minimizar o impacto no próprio *cluster* (um processo irá ser suspenso e eventualmente migrado para um outro recurso enquanto o recurso original é utilizado por um processo com maior prioridade) e também como um meio de balanceamento de carga. O impacto da utilização de migração de processo é similar ao *checkpointing*, com a desvantagem adicional de movimentar pela rede de conexão do *cluster* grandes arquivos de *status*, reduzindo a largura de banda disponível na rede.

3.6 Reescalonamento e monitoração de processo

Habilidade para monitorar os processos que estão trabalhando e, no caso de uma falha no processo, reescaloná-lo para executar novamente.

3.7 Balanceamento de carga

Capacidade de balancear a carga entre os recursos disponíveis. É útil para o administrador do sistema customizar a configuração padrão de modo a utilizar melhor a capacidade de processamento do *cluster*.

3.8 Estabelecimento de limites

Permite ao administrador configurar um limite de tempo de CPU, memória, disco e outros recursos e impõe estes limites no processo batch, evitando um uso excessivo e abuso de recursos computacionais dos usuários, possibilitando a utilização de acordo com políticas pré-estabelecidas.

3.9 Interface gráfica com o usuário (GUI)

Todos os sistemas de *cluster* suportam interface de comando de linha (CLI) em seus ambientes. Fornecer mecanismos CLI para submeter, monitorar e manter processos *batch* em todo o *cluster* é uma característica básica destes sistemas. Fornecer uma interface gráfica para o usuário (GUI) pode aumentar significativamente a produtividade do usuário do *cluster* (especialmente usuários novatos) e está rapidamente tornando-se uma característica requerida dos sistemas populares.

3.10 Fonte aberta

Sistemas de *cluster* de domínio público fornecem características essenciais para *clusters* que requerem a habilidade de fazer pequenas ou significantes mudanças no software para implementar requisitos básicos ou dependências locais.

3.11 Interface Web

Característica para permitir ao usuário submeter, monitorar e manter processos *batch* através de navegadores Web.

Capítulo 4

Softwares para *clusters*

Como comentado anteriormente, já foram desenvolvidos alguns softwares para facilitar o gerenciamento de *clusters*.

Neste capítulo descreveremos alguns destes softwares, suas principais características e seu funcionamento. As características comentadas no capítulo anterior servirão como principais parâmetros. Caso o software apresente características ainda não comentadas, elas serão apenas citadas.

4.1 Sistemas de Gerenciamento de *Clusters*

Os CMS são divididos em dois grupos: os comerciais e os acadêmicos.

Um CMS comercial requer um alto grau de estabilidade, robustez e serviços de apoio mais abrangentes que os softwares acadêmicos. O software é basicamente um gerenciador de *cluster*, o qual é não apenas crucial para a produtividade interna, mas pode ser crítico para os serviços que são oferecidos pela entidade comercial. No caso de softwares acadêmicos, existe mais liberdade e flexibilidade sobre os níveis de serviço que são oferecidos e esperados. É também possível que um software acadêmico não tenha funções que sejam de desenvolvimento custoso, devido à falta de verbas disponíveis.

Vários são os conjuntos de ferramentas, como o acima descrito, já desenvolvidos, entre eles: PBS, DQS, Condor, Codine e outros [7].

Os CMS foram desenvolvidos com o intuito de administrar e gerenciar processos de aplicações específicos para *cluster*. Existem alguns CMS que estão sendo estendidos para gerenciar outros recursos assim como controle de licença de softwares. Os processos podem ser

aplicações paralelas ou seqüenciais que possam ser executados interativamente ou em *background.*

Os CMS podem ser usados para gerenciar cluster de várias formas:

- Otimizar o uso de recursos disponíveis para processos paralelos ou seqüenciais;
- Priorizar a utilização de recursos disponíveis;
- Gerenciar mecanismos para usar ciclos disponíveis de CPU das máquinas do *cluster*;
- Habilitar *checkpointing* e migração de tarefas;
- Oferecer mecanismo para assegurar que as tarefas sejam completadas com sucesso.

4.1.1 Portable Batch System (PBS)

O objetivo do projeto Portable Batch System [29, 35], ou simplesmente PBS, foi inicialmente criar um sistema de processamento de *batch* para satisfazer demandas únicas de uma rede de computadores heterogêneos. O propósito do PBS é oferecer controles adicionais além de inicialização e escalonamento de execução de processos *batch*, e fazer o balanceamento de carga destes processos da melhor maneira possível. O módulo de escalonamento independente do PBS permite ao administrador do sistema definir que tipo de recursos e quanto de cada recurso poderá ser usado por cada processo. Este módulo tem total conhecimento das filas de processo disponíveis, processos trabalhando, e recursos do sistema em uso. O PBS também oferece um mecanismo que permite aos usuários especificar recursos únicos necessários para um processo ser completado.

O PBS oferece:

- Comandos de usuários - para manipulação de processos, tais como, submissão, exclusão, estado, etc;

- Comandos de operação - para configuração, inicialização e finalização de PBS.
- Comandos de administração - para gerenciamento do PBS (configuração, inicialização/finalização de filas);
- Interface de programação de aplicações - projeto de aplicações para serem executadas com PBS;
- Gerenciamento de recursos - para manipulação do PBS para o melhor uso de recursos disponíveis;
- Roteamento de processos - o módulo de movimento de processos de um nó para outro.
- Inicialização de processos - o módulo de seleção e execução de certos processos;
- Estágio de armazenamento - módulo de gerenciamento de espaço de disco e movimentação de arquivos para e/ou do sistema de execução de processos;
- Estado do processo - monitoração do estado dos processos submetidos remotamente;
- Arquivo de histórico - arquivo mantido pelo PBS para estatística do uso dos recursos pelo processo;
- Detecção de erros e recuperação;
- Software cliente pode ser instalado em todas as estações de trabalho do *cluster*.

4.1.2 Distributed Queuing System (DQS)

O DQS [25, 19] (Distributed Queuing System) é um sistema de fila baseado em UNIX, desenvolvido pelo Supercomputer Computations Research Institute (SCRI) na Florida State University. O DQS foi desenvolvido como uma ferramenta de gerenciamento para ajudar na distribuição de recursos computacionais pela rede. DQS oferece

arquitetura transparente tanto para usuários como para administradores num ambiente heterogêneo.

Algumas características do DQS:

- Interface GUI;
- Controlador GUI;
- Utilitários de paralelização;
- Apoio a pacotes paralelos - PVM, TCG/MSG, P4/P5 e MPI;
- Apoio à interatividade.
- Escalonamento baseado em Filas-complexas[1];
- Ponto não único de falhas, redundância de filas principais são suportadas.

4.1.3 Condor

Condor [8, 16] é um software para execução de processos *batch* em estações de trabalho que possam estar ociosas. A maioria das ferramentas do Condor é de localização automática e alocação de máquinas ociosas, *checkpointing* e migração de processos. Todas essas operações são realizadas sem qualquer modificação nas linhas de kernel do UNIX. Não é preciso modificar o código fonte para executar o Condor, embora os programas devam estar ligados especificamente com bibliotecas do Condor. O Condor monitora a atividade de todas as estações de trabalho localizadas na rede local. As máquinas que são determinadas como ociosas são colocadas no conjunto de recursos. Máquinas são então alocadas do conjunto para a execução de processos. O conjunto é uma entidade dinâmica - estações de trabalho funcionam quando elas se tornam ociosas, e desligam quando se tornam ocupadas.

Algumas características:

---

[1] Conjunto de filas com um servidor de batch. O propósito de uma fila complexa é oferecer controle adicional sobre o uso de recursos.

- O usuário apenas precisa projetar seu código para executar sobre o Condor;
- O ambiente de execução local da máquina que submeteu o processo é apresentado ao processo mesmo em execução remota;
- O Condor é responsável pela localização e alocação de estações de trabalho ociosas;
- Os "proprietários" de estações de trabalho têm total prioridade nas suas máquinas;
- Espaço de disco local não é usado por processos do Condor executados remotamente;
- Condor trabalha fora do kernel, e é compatível com BSD e outros derivados UNIX.

4.1.4 Codine

Os softwares acima descritos são de pesquisa, sem fins comerciais (com exceção do PBS, que possui versão comercial). Já o Codine [15] é desenvolvido pela Cray, como um pacote comercial. Seu objetivo é ser utilizado em ambientes de redes heterogêneos, em particular em grandes *clusters* de estações de trabalho com servidores de computadores integrados, como computadores vetoriais e paralelos. Codine oferece um ambiente de enfileiramento de *batch* para uma grande variedade de arquiteturas via uma ferramenta de administração baseada em um ambiente gráfico (GUI). Ele também oferece um balanceamento de carga dinâmico e estático, *checkpointing* e apoio *batch*, processos interativos e paralelos.

Principais características do Codine:

- Apoio à *batch*, processos interativos e paralelos (Express, p4 e PVM);

- Apoio à filas múltiplas;
- Apoio à *checkpointing*;
- Balanceamento de carga estático;
- Balanceamento de carga dinâmico por *checkpointing* e migração de processos;
- Interface NQS - usado para interação com sistemas existentes NQS (Network Queueing System [18]);
- Estatística de utilização.

4.1.5 Connect:Queue

Outro pacote comercial é vendido pela Sterling Software Inc.: o Connect:Queue. Este pacote é uma variação do software de pesquisa NQS. Suas características e funcionalidades são muito similares ao GNQS (Generic NQS [28, 23]). Sua plataforma é homogênea, ou seja, oferece facilidades apenas para plataformas baseadas em Unix. Possui três tipos de filas:

- Fila *batch*, oferecendo um mecanismo de escalonamento e execução de processos;
- Fila de dispositivos, que oferece ao *batch* acesso aos dispositivos físicos;
- Fila de *pipe*, o qual transporta pedidos de outra fila de *batch*, dispositivo ou *pipe* para locais remotos.

Um sistema inteligente de escalonamento de processos *batch* oferece um balanceamento de carga de processos através das estações de trabalho do *cluster* que são gerenciados. O balanceamento de carga é baseado em um algoritmo de escalonamento que utiliza três estatísticas:

- Porcentagem de memória física utilizada.
- Utilização de CPU.
- Limite de fila de processos.

4.1.6 Load Balancer

O Load Balancer tende a otimizar o uso dos recursos do computador por distribuição de carga de trabalho pelos sistemas UNIX disponíveis através da rede. O Load Balancer determina a disponibilidade baseado na:

- Capacidade e carga de trabalho de cada computador.
- Recursos pedidos pelo processo submetido.
- Usuários definidos e outras restrições.

O Load Balancer tenta aumentar acima de tudo o *throughput* do processo fazendo uso dos recursos ociosos do computador. Ao mesmo tempo, ele tenta prevenir que os sistemas se tornem super carregados pela distribuição de carga mesmo através de computadores disponíveis.

Suas principais características são:

- Não modificação das aplicações ou kernel.
- Controle e administração central.
- Opções de políticas e restrições.
- Os critérios de distribuição de processos/aplicações incluem:
    - Velocidade de CPU;
    - Média de carga corrente;
    - Hora;
    - Espaço de RAM e SWAP;
    - Tempo de CPU;
    - Presença e ausência de usuários interativos;
    - Escalabilidade e resistência;
    - Outros critérios definidos para usuários;
- Trabalha tanto com processos *batch* quanto com aplicações interativas;

- Possui características de segurança para restringir acessos não autorizados.

4.1.7 LoadLeveler

O software descrito agora é um escalonador de processos. O LoadLeveler distribui os processos do UNIX. Os processos são mantidos em filas até que o LoadLeveler possa alocar os recursos necessários para executar o processo. Uma vez que o processo foi completado, o LoadLeveler irá notificar o usuário. O usuário não precisa especificar em que máquina quer que o processo seja executado, pois o LoadLeveler escolhe as máquinas apropriadas.

Características:

- Escalonador de processos distribuído;
- *Batch* serial/paralelo e carga de trabalho interativo;
- Balanceamento de carga de trabalho;
- Apoio a estações de trabalho heterogêneas;
- Compatibilidade com o NQS;
- Ponto de controle central;
- Escalabilidade (filas e escalonador).

Quando um processo é escalonado, seus pedidos são comparados com todos os recursos disponíveis pelo LoadLeveler. Os pedidos do processo podem ser uma combinação de memória, espaço de disco, arquitetura do sistema operacional e programas de aplicação. O gerente central do LoadLeveler coleta informações de recursos e executa o processo assim que ele localiza os recursos satisfatórios.

4.1.8. Load Sharing Facility (LSF)

Um software utilizado em ambientes UNIX heterogêneos é o LSF (Load Sharing Facility). O LSF é um pacote de sistema de enfileiramento de

batch e compartilhamento de carga distribuída. Esse pacote gerencia o processamento de tarefas oferecendo uma visão simples e transparente de todos os recursos de hardware e software, sem se preocupar com os sistemas individuais no *cluster*. LSF suporta *batch*, processos paralelos e interativos e gerencia esses processos no *cluster* fazendo uso de estações de trabalho ociosas e servidores.

Especificamente, o LSF oferece:

- Escalonamento e enfileiramento de batch;
- Distribuição interativa de processo pela rede;
- Balanceamento de carga através dos nós;
- Acesso transparente aos recursos heterogêneos através da rede;
- Balanceamento de carga de rede monitorado;
- Carga inclusiva e informação de recursos;
- Execução remota transparente. O mesmo ambiente de execução (identificação do usuário, *umask*, limites de recursos, variáveis de ambiente, diretórios de trabalho, estado de saída de tarefa, sinais, parâmetros de terminais, etc) é mantido quando são enviados processos para executar remotamente.
- Processamento interativo e *batch*;
- Processamento paralelo e seqüencial;
- Operações heterogêneas;
- Interface de programação de aplicação (API);
- Tolerância a falhas - LSF esta disponível mesmo que apenas um nó esteja ligado e nenhum processo será perdido mesmo que todos os nós estejam desligados;
- Opções de políticas locais - oferece opções de configuração para customizar os recursos locais;

- Dados de contagem de processos e ferramentas de análise.

4.1.9 Task Broker

Outro software utilizado em sistemas de computação baseados em UNIX heterogêneos é o Task Broker, distribuindo processos entre esse sistema. Ele executa suas distribuições computacionais sem necessitar de qualquer mudança na aplicação. Esse software irá reescalonar um processo e seus dados de acordo com as regras configuradas na inicialização. Outras capacidades do Task Broker são:

- Habilidade para balanceamento de carga computacional em um grupo de sistemas de computadores;
- Interoperabilidade com o DCE (ver adiante).

O Task Broker automatiza a distribuição de computações por:

- Reunir conhecimentos baseados na máquina de cada usuário;
- Analisar conhecimentos específicos de cada máquina e selecionar o servidor mais disponível;
- Conectar um servidor selecionado via telnet, rsh (remote shell) ou crp (create remote process).
- Copiar programas e arquivos de dados para um servidor selecionado via ftp ou NFS.
- Executar aplicações pela rede;
- Copiar os arquivos de dados de resultados retornados do servidor via ftp ou NFS.

As características que podemos destacar são:

- Execuções de aplicações sem exigir recompilação do código;

- Inclusão de servidores e clientes para uma rede simplesmente os enganchando a uma LAN, atualizando o arquivo de configuração central para incluir o servidor ou cliente novo e reinicializando o *daemon* do Task Broker;
- Contagem de serviços utilizados em um determinado servidor;
- Permissão de limitação do número de aplicações que são executadas em um computador em qualquer momento. Isto previne a degradação do desempenho do servidor;
- Controle do uso de computadores como servidor. Por exemplo, um usuário pode especificar que sua máquina só será acessada no período da noite;
- A interface gráfica para o usuário (GUI) oferece uma interface visual da maioria dos comandos de configuração de informações do Task Broker. Além disso, monitoramento e controle do estado das tarefas são oferecidos para o usuário;
- Sistema de ajuda em tempo real.

Para resumir as características que os CMSs possuem dentro das descritas no capítulo anterior, segue a tabela com o relacionamento Software/Característica. Identificamos com *?* as características sobre as quais não encontramos informações nas bibliografias.

Tabela 4.1 Comparação entre os softwares e as características de um sistema de gerenciamento de cluster.

Características	*CODINE*	*Condor*	*DQS*	*Load Leveler*	*LSF*	*Connect Queue*	*Load Balancer*	*Task Broker*	*PBS*
Apoio heterogêneo	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Não	Sim
Ambiente Batch	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Apoio a computação paralela	Sim	Não	Sim	Sim	Sim	Sim	Não	Não	Sim
Checkpointing	Sim	Sim	Sim	Sim	Não	Não	Sim	Não	Não

Migração de Processos	Sim	Sim	Não	Sim	Sim	Não	Sim	Não	Não
Reescalonamento e monitoração de processos	Sim	Não	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Balanceamento de carga	Sim	Não	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Limite de tempo	Sim	Não	Sim	Sim	Sim	Sim	Sim	Não	Sim
GUI	Sim	Não	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Fonte aberta	Não	Sim	Sim	Não	Não	Não	Não	Não	Sim
Interface Web	?	?	?	?	?	?	?	?	Sim

4.2 Ambientes de Computação em *Cluster*

Os pacotes de softwares descritos abaixo estão classificados com Cluster Computing Environments (CCE). A descrição consiste em informações geradas a partir de guias de usuários e documentos *on-line*.

4.2.1 Amoeba

Começaremos descrevendo o Amoeba [1] . Amoeba é uma proposta de sistema operacional distribuído desenvolvido e apoiado pela Vrije Universiteit in Amsterdan. Amoeba é projetado para transformar uma coleção de máquinas em um único sistema integrado, fazendo com que elas atuem juntas. Para o usuário, o número de máquinas e a sua locação são transparentes, ou seja, o usuário não tem conhecimento destes valores.

Amoeba é um projeto de pesquisa em andamento e deve ser considerado como uma plataforma para desenvolvimento e pesquisa em sistemas paralelos e distribuídos, linguagens, protocolos e aplicações. Ele oferece apenas emulação do Unix, mas não é um substituto do Unix.

Amoeba foi desenvolvido sobre uma arquitetura microkernel - aqui cada máquina executa um kernel que apóia o processo básico, comunicações e objetos primitivos. Ele também controla saídas I/O e gerenciamento de memória. Tudo é feito embasado nestes fundamentos.

Um sistema Amoeba típico irá consistir de três classes de máquinas fundamentais. Primeiro, cada usuário tem uma estação de trabalho para executar a interface ao usuário, baseado no sistema X Window. Segundo, há um conjunto de processadores que são dinamicamente alocados para os usuários que o requisitarem. Terceiro, há servidores especializados, como servidores de arquivos e servidores de diretórios que executam o tempo todo. Todos estes componentes são conectados via uma rede local rápida - no momento apenas Ethernet é suportada.

4.2.2 Beowulf

O Beowulf [26, 9] é um projeto para produzir software para controlar *clusters* baseado em hardware PC e sistema operacional Linux. Beowulf agrupa o baixo custo, desempenho moderado de subsistemas PC comuns com o surgimento de padrões em sistemas de hardware e software para passagem de mensagem, utilizando a capacidade de armazenamento de arquivo local e largura de banda. Este sistema experimental é guiado pelas exigências das aplicações científicas da NASA Earth and Space, incluindo simulação de dados, visualização e amostragem de conjunto de dados e simulação de sistemas físicos naturais.

A arquitetura de *cluster* paralelo do Beowulf inclui subsistemas de processadores PC, cada um com um disco, controlador e 16 Mbytes de DRAM. Os elementos de processamento estão interconectados por duas redes Ethernet paralelas com uma capacidade de 10 Mbps por rede. Dois dos PCs tem interface Ethernet separada para LANs externas para acesso remoto e transferência de dados. Dois subsistemas adicionais têm controladores de vídeo de alta resolução, um destes também possui um teclado e um mouse para acesso convencional de um único usuário.

A maioria das aplicações paralelas atualmente é executada utilizando PVM para comunicação, ou RPC.

4.2.3 The Oxford BSP Library

A biblioteca Oxford BSP [10] oferece um mecanismo para desenvolvimento de código portátil e eficiente através de um grande conjunto de plataformas distribuídas. O modelo de programação é baseado no modelo arquitetônico do Bulk Synchronous Parallel (BSP) [40]. Este modelo geral abstrato parametriza características do sistema para medir desempenho através de plataformas distribuídas sem ligar o código de aplicação a uma arquitetura específica. Previsão de desempenho e determinação de estratégias paralelas ótimas para aplicações são baseados em um modelo de custo correspondente.

A biblioteca consiste de um pequeno número de sub-rotinas para implementar a criação de processos, acesso a dados remotos e sincronização. A biblioteca pode ser ligada a programas escritos em linguagens seqüenciais padrão como Fortran, C e Pascal.

A especificação das rotinas da biblioteca é portátil, mas suas implementações não o são necessariamente. Instâncias específicas de máquina podem ser criadas para utilizar as facilidades do paralelismo para cada ambiente. Assim a biblioteca, ao invés do programa, é afinada para a máquina, alcançando códigos de aplicação de fonte única.

A biblioteca de BSP adota um modelo de programação SPMD (Single Program, Multiple Data) estático, onde todos os processos paralelos executam o mesmo texto de programa. Os processos procedem juntos através de uma seqüência de passos, fazendo com que assim todos os processos terminem os passos juntos. Cada processo tem seu próprio espaço de dados; não há dados globais compartilhados, mas um processo pode acessar explicitamente as variáveis de qualquer outro processo. O acesso remoto a dados é assíncrono; mas não é garantido ser satisfeito até que o passo corrente termine, quando todos os processos são sincronizados.

O primeiro protótipo da biblioteca foi desenvolvido utilizando as bibliotecas de passagem de mensagem PVM e PARMACS. Esta implementação foi portátil, mas não eficiente. Versões genéricas de alto desempenho foram implementadas utilizando *sockets* TCP/IP (para clusters) e primitivas do Sistema V para memória compartilhada (para multiprocessadores de memória compartilhada). Para o aumento de desempenho, há versões nativas da biblioteca para máquinas específicas, tais como IBM SP1/SP2, SGI Power Challenge, Meiko CS-2 e Cray T3D.

4.2.4 Distributed Computing Environment (DCE)

O DCE [18] é um conjunto abrangente e integrado de serviços para apoio a desenvolvimento, uso e gerenciamento de aplicações distribuídas. Ele oferece um conjunto uniforme de serviços, em qualquer lugar na rede, habilitando as aplicações usarem o poder das redes de computadores heterogêneas.

O objetivo do DCE é se tornar independente de sistema operacional e rede, oferecendo compatibilidade com os ambientes de usuário existentes. DCE inclui um conjunto abrangente de serviços distribuídos que oferecem qualidade comercial e interoperabilidade em ambiente computacional distribuído.

A arquitetura do DCE tenta mascarar a complexidade física dos ambientes de rede e oferece uma camada de simplicidade lógica para o usuário. Esta camada é composta por um conjunto de serviços que podem ser utilizados, separadamente ou em combinação, para formar um ambiente distribuído. Os serviços são organizados em duas categorias:

- Fundamental Distributed Services (FDS), que oferece ferramentas para os desenvolvedores de softwares criar as aplicações para os usuários necessárias para computação distribuída. Entre elas:

    - Chamada remota a funções;
    - Diretório de serviço;
    - Tempo de serviço;
    - Segurança no serviço;
    - Serviços de *threads*.
- Data Sharing Services oferecem aos usuários a capacidade de construir um FDS. Este serviço não necessita de programação por parte do usuário e facilita o uso da informação. Este serviço inclui:
    - Arquivo de sistema distribuído;
    - Apoio a estações sem disco.

O DCE permite que o poder de processamento inerente à rede seja estendido a um grande número de nós. O DCE estende o poder da rede - e todos os seus recursos - individualmente aos usuários em PCs e Macintosh. Como resultado, PCs e Macintosh interconectados por LAN ou servidores de rede não são mais isolados. Com DCE, eles são confiados a sistemas clientes por nível que pode alcançar serviços de processamento e informação fora do servidor.

O DCE pode ser utilizado com uma variedade de redes e sistemas operacionais por vendedores de sistemas, desenvolvedores de software ou usuários finais. Ele pode ser usado com qualquer hardware de rede e software de transporte, incluindo TCP/IP, OSI, X.25, e outros produtos similares.

O DCE tem uma implementação portátil, escrita em C, que faz uso de interfaces padrões para serviços de sistema operacional, tais como POSIX e X/Open. Ele pode ser utilizado facilmente por OSF/1, AIX, DOMAIN OS, ULTRIX, HP-UX, SINIX, SunOS, e UNIX.

4.2.5 Distributed Object Migration Environment (Dome)

O Dome [3, 20] está sendo usado para desenvolver aplicações "Grand Challenge", tais como dinâmica molecular, física nuclear, simulação distribuída, sequenciamento de genes e reconhecimento de fala, em redes de *cluster* heterogêneos. Dome torna possível aproveitar as vantagens dos multicomputadores (redes de *cluster*), MPPs (*massively parallel processors*), sistemas operacionais avançados, e redes gigabit que são desenvolvidas para o programa US HPCC.

O Dome oferece esta capacidade através de bibliotecas de objetos distribuídos (por exemplo, software escalável) que pode ser utilizado por programas em redes de computadores heterogêneos como um recurso único, obtendo desempenho que não pode ser alcançado em máquinas individuais.

Algumas características do Dome:

- Solução para problemas de balanceamento de carga, facilidade de programação, e tolerância a falhas;
- Os objetos se distribuem automaticamente entre os múltiplos computadores;
- Balanceamento de carga da aplicação a ser executada;
- Apoio a mecanismos de recuperação de falhas em recursos.

O Dome está disponível em oito plataformas: DEC Alpha OSF/1, HP HP-UX, Intel Paragon, Sparc SunOS, SGI Irix, DEC Ultrix, IBM AIX, e Cray C90 Unicos.

Incorpora uma técnica de executar operações de comunicações coletivas. Esta técnica automaticamente deriva as topologias de rede e usa esta informação par otimizar as operações coletivas. Isto é realizado por escolha de uma rotina de mensagem que mapeie eficazmente uma rede subjacente.

Possui um *checkpointing* independente de arquitetura e reinicia a inclusão de seqüência de chamadas, tornando o mecanismo mais flexível.

O sistema atual permite que os *checkpointing* sejam inicializados em qualquer ponto numa seqüência de chamadas.

O Dome faz uso de técnicas de balanceamento de carga dinâmico para incluir informações de carga global e local.

4.2.6 GLU Parallel Programming System

O GLU (Granular Lucid) [30, 24] é um sistema de programação de alto nível para construção de aplicações paralelas e distribuídas para executar em diversos sistemas de computação de alto desempenho. GLU é baseado no modelo de programação híbrido que combina fluxo de dados multidimensional (Lucid) com modelos imperativos. GLU habilita o desenvolvimento rápido de programas paralelos usando códigos seqüenciais existentes, ajudando a produzir programas portáteis, eficientes e tolerantes a falha.

O GLU é um conjunto de ferramentas para construir e executar aplicações de alto desempenho em sistemas distribuídos incluindo *clusters* heterogêneos (com WAN, LAN e redes sem fios), multiprocessadores e processadores paralelos. O conjunto de ferramentas GLU consiste de uma notação de descrições de aplicações de alto nível, um compilador, uma interface para usuário gráfica de execução de aplicações, e uma ferramenta de análise de desempenho. O GLU pode ser utilizado para desenvolver novas aplicações paralelas ou conversões de aplicações seqüenciais.

O conjunto de ferramentas GLU tem sido utilizado para desenvolver aplicações de alto desempenho para algumas aplicações seqüenciais já existentes, onde as aplicações originais foram escritas em C, Fortran e C++. O sistema GLU atual está disponível para os seguintes sistemas:

- *Cluster*: Sparc/Solaris 1.1 e 2.3, SGI/IRIX v4.0.5F, IBM RS6000/AIX v3.2, DEC AXP/OSF1 v1.3 e x86/Linux v1.1.58.
- Multiprocessadores simétricos: SparcMP/Solaris 1.1 e SGI ONYX/IRIX v4.0.5F.
- Grandes processadores paralelos: CM5/SunOS 4.1.3.

4.2.7 Local Area Multicomputing (LAM)

Outro CCE a citar é o LAM [11, 32], que executa em cada computador um *daemon* Unix simples como um microkernel e processadores virtuais *hand-threaded*. Ele oferece um serviço simples de passagem de mensagens. Os processos *daemon* formam um subsistema de rede de comunicação, que transfere mensagens de e para outros *daemons* LAM em outras máquinas. O subsistema de rede adiciona características como empacotamento e armazenamento para sincronização básica. Outros processos *daemon* são servidores para capacidades remotas, tais como execução de programa e acesso a arquivo paralelo. Os usuários podem configurar seus serviços locais ou remotos, quando necessário.

As características da LAM são transparentes ao usuário e ao administrador do sistema, que apenas vêem um *daemon* Unix convencional. Os desenvolvedores de sistemas podem agrupar o *daemon* em um *daemon* contendo apenas um nano-kernel e vários processos clientes completos de Unix. A camada de rede no LAM é uma camada documentada, primitiva e abstrata para implementar um padrão de comunicação mais poderoso como o MPI (PVM também pode ser implementado).

Uma característica importante da LAM é o controle manual do multicomputador. Existe uma pequena parte que não pode ser vista ou mudada em tempo de execução. Os programas residem em qualquer lugar onde possam ser executados de qualquer outro lugar, parados, resumidos, finalizados ou vistos a qualquer momento. A sincronização de processos e mensagens pode ser vista facilmente, erros podem facilmente ser encontrados. Estes e outros serviços estão disponíveis como bibliotecas de programação e programas utilitários.

Capítulo 5

Software proposto

Agora que já descrevemos alguns softwares existentes no mercado, descreveremos um protótipo por nós projetado, baseado nas necessidades do nosso *cluster* e dos nossos usuários. Salientamos que nossa intenção não é desenvolver uma ferramenta completa para *cluster* e sim um protótipo para futuros projetos procurando demonstrar a possibilidade de desenvolver uma ferramenta para operação de *clusters* usando HTML e CGI Scripts[13, 14, 21].

Nosso protótipo é do tipo CMS, citado no capítulo *Gerenciamento de clusters*.

5.1 Descrição

O software proposto está dividido, inicialmente, em três páginas Web. A primeira página que o usuário recebe é a página de identificação, onde ele fornece seu nome de usuário. Após fornecer seu nome, ele é direcionado para o seu diretório UNIX, que tem acesso controlado pelo método *fstaccess* do servidor Apache [4] utilizado e, portanto, o usuário deve conectar com sua senha. É neste ponto que os usuários são identificados como usuários ou administrador do sistema. Caso seja um usuário ele entrará em uma página para usuários.

A página dos usuários possui opções para:

- Submeter processos;
- Verificar recursos, como espaço em disco e memória em uso;
- Finalizar processos;
- Verificar andamento de uma tarefa;
- Verificar status das máquinas;

- Gerenciar diretório do usuário;
- Verificar máquinas disponíveis para execução.

A opção Submeter processos permite ao usuário submeter os processos instantaneamente ou programar, dentro da sua permissão de horário, o horário em que ele deseja que o processo comece a trabalhar. Também é possível escolher em quais nós o usuário deseja que os processos sejam executados (dentre os nós habilitados pelo administrador).

Caso o usuário queira saber a ocupação dos discos, deve utilizar a opção "verificar discos", onde o sistema retornará uma página com a demonstração gráfica da utilização de disco de cada nó. Se desejar saber a ocupação de memória, deve usar a opção "verificar memória", que demonstrará graficamente a utilização da memória de cada nó.

Para finalizar processos já iniciados, o usuário utiliza a opção "finalizar processos", onde aparecerá uma lista dos processos inicializados pelo usuário, lista esta utilizada para escolher qual(is) processo(s) deseja finalizar.

A opção "verificar andamento de uma tarefa", exibe para o usuário os programas, caso estes existam, em execução no sistema e os processos que já finalizaram a execução.

Os processos em execução podem ser verificados utilizando a opção "status das máquinas", que mostra todos os processos em execução submetidos pelo usuário.

Também é possível verificar se os programas a serem iniciados se encontram no diretório do usuário. Para tanto, basta utilizar a opção conteúdo do diretório, que mostrará todos os arquivos armazenados no diretório.

Para saber quais os nós disponíveis para ele, o usuário deve utilizar a opção "máquinas disponíveis".

Se o usuário em questão é o gerente do sistema, será mostrada uma página específica para o gerenciamento do sistema.

As opções desta página são as seguintes:

- Incluir usuários;
- Remover usuários;
- Atribuir permissão de uso das máquinas;
- Finalizar processos;
- Reiniciar o *cluster*.

Estas opções serão descritas abaixo.

A opção "incluir usuário", permite ao gerente incluir usuários tanto no sistema operacional como no sistema paralelo.

A opção "remover usuário", ao contrário da opção anteriormente descrita, exclui usuários, tanto do sistema operacional como do sistema paralelo.

Como já descrito acima, uma das particularidades deste sistema é restringir o uso de nós para cada usuário. Cabe ao gerente do sistema esta atribuição. E para tanto, ele deve utilizar a opção "atribuir permissão de uso de máquinas". Ele pode atribuir, bem como restringir a utilização dos recursos.

Assim como o usuário, o gerente do sistema tem permissão para finalizar um processo, caso seja necessário. O gerente pode finalizar qualquer processo de qualquer usuário. Para executar esta tarefa, deve ser utilizada a opção "finalizar processos".

Caso seja necessário reinicializar todas as máquinas, somente o gerente tem permissão para realizar esta tarefa. E, para tal, deve utilizar a opção "Reiniciar *cluster*".

5.2 Implementação

Para o desenvolvimento da interface com o usuário foi escolhido o uso de HTML com CGI. Isto porque esta ferramenta possibilita maior facilidade de operação remota do sistema, além de permitir acesso por qualquer máquina com navegador Web.

CGI, ou *Common Gateway Interface* [12], é uma interface definida de maneira a possibilitar a execução de programas ("*gateway*") sob um servidor de informações. Até o momento, CGI suporta apenas servidores HTTP.

Neste contexto, os "*gateway*" são programas ou *scripts* (também chamados "cgi-bin") que recebem requisições de informação, retornando um documento com os resultados correspondentes. Esse documento pode existir previamente, ou pode ser gerado pelo *script* especialmente para atender à requisição (usa-se muito a expressão "*on the fly*").

Exemplos de aplicação de CGI incluem:

- Processamento de dados submetidos através de formulários - consulta a banco de dados, cadastramento em listas, livros de visita, pesquisas de opinião;
- Criação de documentos personalizados *on the fly*;
- Gerenciamento de contadores de acesso;
- Processamento de mapas.

Tais scripts podem ser escritos em qualquer linguagem que possa ler variáveis, processar dados e retornar respostas - ou seja, qualquer linguagem de programação. Os programadores de *scripts* costumam utilizar determinadas linguagens, de acordo com a plataforma do servidor: C, Perl ou *shell* de UNIX, em redes de ambientes UNIX e C, Perl ou Visual Basic, em ambiente Windows.

Para o desenvolvimento dos *scripts* utilizamos Perl [26], pois Perl é a linguagem de desenvolvimento ideal para trabalhar em servidores Web, por várias razões. Muitos projetos de programação em servidores Web são em alto-nível, o que significa que eles tendem a não envolver manipulação de bits, chamadas diretas ao sistema operacional, ou interação com o hardware do servidor. Pelo contrário, eles focalizam leituras de arquivos, reformulação de saídas, e escritas no navegador. As tarefas de alto nível são as melhores executadas pelo Perl. Além desta vantagem, também é uma linguagem flexível e eficiente. Perl (Practical Extraction and Report Language) ou Linguagem Prática para Extração e Relatório é uma linguagem de programação de alto nível, criada por Larry Wall. Ela é descendente de C, mas também sofreu grandes influências de várias linguagem e ferramentas, tais como Sed, Awk e Unix shell, tendo uma sintaxe um tanto familiar aos usuários destas linguagens.

As facilidades de Perl para manipulação de processos, arquivos e textos tornam essa linguagem particularmente interessante para tarefas que envolvem desenvolvimento rápido de protótipos, utilitários e gerenciamento do sistema operacional, acesso a bancos de dados, redes, e programação para a World Wide Web.

JavaScript é uma linguagem de *script* desenvolvida pela Netscape para permitir aos desenvolvedores de páginas criar páginas interativas. Embora ela compartilhe diversas das características e estruturas da linguagem Java, ela foi desenvolvida independentemente. JavaScript pode interagir com códigos HTML, permitindo aos desenvolvedores incrementar suas páginas com conteúdo dinâmico. JavaScript é endossado por um grande número de companhias de software e é uma linguagem aberta que qualquer pessoa pode utilizar sem o pagamento de qualquer tipo de licença. Ela é suportada pelos navegadores mais recentes

IFSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

da Netscape e Microsoft, entretanto o Internet Explorer suporta só um subconjunto, que a Microsoft chama de Jscript.

JavaScript, no nosso caso, foi utilizado em todos os momentos em que se fazia necessária a autenticação do usuário, após este estar conectado ao sistema.Ou seja, no momento em que o usuário se conecta no sistema, este armazena dados, em forma de *Cookies*, para que o JavaScript possa identificar que o mesmo está conectado. Ao desconectar, estes dados são apagados.

Seguindo as opções mencionadas anteriormente, mostraremos como cada opção foi implementada. Começaremos pela tela do usuário.

A opção "Submeter processo" segue a seguinte lógica: identifica as máquinas disponíveis para o usuário (é feita a leitura de um arquivo no diretório do usuário com as máquinas que ele pode acessar) e retorna uma página HTML, em forma de formulário com botões de escolha e os números das máquinas que podem ser utilizadas, para o usuário escolher em quais máquinas ele deseja executar seu processo. Assim que as máquinas forem escolhidas, o usuário clica no botão *Submeter* e o processo é executado, utilizando o comando do MPICH, o MPIRUN. O *script* ativado quando o usuário clica no botão Submeter envia para o sistema os seguintes dados: *processo, número de processos, momento, hora e máquinas*. Essas informações são tratadas e concatenadas em uma linha do comando MPIRUN, com a seguinte sintaxe:

MPIRUN -np número de *processos* –machinefile *máquinas* -nolocal *processo* > *processo*.log

Se o usuário escolher agendar o processo, acrescentamos parâmetros no comando de tal forma que seja possível executar o processo no dia e hora agendado pelo usuário. Para saber se o usuário escolheu agendar o processo, o sistema verifica se a informação *momento* não está vazia. O comando MPIRUN fica com a seguinte sintaxe:

MPIRUN -np número de processos –machinefile máquinas -nolocal processo > processo.log | **at hora**

Todos os processos executados geram um arquivo *log*, para verificação posteriormente.

A opção "Verificar discos" executa o comando DF do UNIX, utilizando RSH[2], para verificar o espaço em disco de cada máquina disponível ao usuário.

A opção "Verificar memória" executa o comando MENINFO do UNIX, também utilizando RSH, para exibir ao usuário as informações sobre a memória, tais como ocupação e quantidade livre no momento.

A opção "Finalizar processos" segue a seguinte lógica: o usuário pode digitar qual processo quer finalizar ou verificar quais os processos em andamento, e então escolher qual quer finalizar. Caso o usuário saiba exatamente o nome do processo que deseja finalizar, ele apenas digita o nome do processo numa caixa de texto disponível para isso e clica no botão *Finalizar*. Caso contrário, ele clica no botão *Finalizar* e o sistema retorna uma página HTML em forma de formulário com os processos do usuário em andamento. O usuário então escolhe o processo e clica no botão *Finalizar*. O comando UNIX utilizado aqui para finalizar o processo é o KILL.

Com a opção "Conteúdo do arquivo *log*", o sistema exibe uma página em formato HTML ao usuário mostrando os arquivos *log* disponíveis no diretório do usuário, identificando se aquele arquivo de *log* é de um processo que está em andamento ou finalizado. Para obter o *status* do processo, o sistema verifica a existência de um arquivo com o formato *nome do processo*.acabou. Este arquivo é criado pelo sistema quando o processo encerra. Caso esteja finalizado, o usuário é informado. Assim que o usuário escolher o arquivo e pressionar o botão "Visualizar

[2] O RSH permite a executar de outro comando em uma máquina remota

arquivo", o comando CAT do UNIX é executado e direciona o conteúdo do arquivo log para uma página HTML.

A opção "Status da máquina" utiliza o comando PS para informar o que está sendo executado em cada máquina disponível ao usuário e assim possibilitar ao mesmo a escolha das melhores máquinas para que seu processo seja executado mais rapidamente.

A opção "Conteúdo do diretório" utiliza o comando LS para mostrar ao usuário o que está armazenado em seu diretório.

A opção "Máquinas disponíveis" faz a leitura de um arquivo no diretório do usuário (este arquivo pode ser alterado apenas pelo administrador do sistema) que informa quais as máquinas que estão disponíveis ao usuário. Este arquivo é criado no momento em que o administrador do sistema associa as máquinas ao usuário.

Agora falaremos sobre as opções da página do administrador do sistema. Para identificarmos que este usuário é o administrador do sistema, ele é incluído em uma lista de usuários chamada "Sudoers". Nesta lista são incluídos os usuários que possuem a permissão de executar comandos e tarefas como superusuário. Um usuário incluído nesta lista deve acrescentar o comando SUDO antes do comando a ser executado e recebe a permissão habilitada no arquivo. O sistema então verifica se o usuário está incluído nesta lista e executa o comando desejado.

A opção "Incluir usuários" inclui o usuário em um arquivo de controle do sistema e utiliza o comando USERADD para adicionar o usuário ao sistema. É também neste momento que o administrador atribui máquinas ao usuário, criando um arquivo no diretório do usuário com as máquinas disponíveis ao mesmo. Este arquivo pode ser alterado posteriormente, desde que seja alterado apenas pelo administrador do sistema.

A opção "Reiniciar *cluster*" executa o comando SHUTDOWN em cada máquina do *cluster*.

Para desenvolver este protótipo encontramos vários problemas. Seguem abaixo os problemas e suas soluções:

Problema 1:

No momento em que o usuário escolhe a(s) máquina(s) para verificar *status* ou executar seu processo, como acessar esta(s) máquina(s) remotamente?

Utilizando comando system, um comando UNIX com a seguinte sintaxe: system ("comando","parâmetro1"... "parâmetro n"). Por exemplo, digamos que o usuário deseja saber o status da máquina de número 6 do cluster. O comando teria a seguinte forma: system ("rsh", "10.6.1.6 'ps –aux'"), onde RSH determina que será executado um comando remotamente e 10.6.1.6 'ps –aux' são os parâmetros do comando que será executado.

Problema 2:

A máquina principal não permitia que o usuário *nobody* (usuário utilizado para executar páginas http) se conectasse ao *cluster*. Como solucionar este problema, uma vez que precisaremos executar páginas HTTP na máquina principal por ser uma interface Web.

Criar um usuário na máquina principal que possa executar http. Este usuário é utilizado para executar o Apache. O usuário criado foi o usuário httpd.

Problema 3:

Qual a melhor forma de fazer a autenticação do usuário no sistema para evitar que usuários não cadastrados acessem o *cluster* e garantir que o usuário cadastrado está acessando apenas sua área de trabalho? Isto se faz

necessário uma vez que a configuração padrão do servidor Apache permite a execução de *scripts* CGI localizados em um diretório específico por um usuário específico, o *nobody*.

A utilização do módulo SUExec do Apache. Este módulo permite que os scripts CGI sejam executados nos diretórios dos usuários e não no diretório específico do servidor Apache. Também permite que os scripts CGI sejam executados pelo usuário em questão. Ou seja, a utilização do módulo SUExec permite uma maior facilidade e privacidade quando utilizamos servidor Apache.

Outros problemas relacionados com configurações de servidores e utilização de comandos UNIX da melhor forma possível também foram enfrentados e solucionados, mas não merecem maior destaque.

Dentre as características descritas nos capítulos anteriores, este protótipo possui as seguintes:

- Ambiente *Batch*;
- Apoio a programação paralela;
- Monitoração de processo;
- Interface Web;
- Fonte aberta.

Como sugestão para futuros trabalhos, seria interessante enfatizar as seguintes características:

- Balanceamento de carga;
- Reescalonamento de processo;
- *Checkpointing*.
- Migração de processo.

Capítulo 6

Conclusão

Neste trabalho foram expostos alguns softwares de gerenciamento de *cluster* existentes no mercado, suas características e seus principais recursos. Também foi sugerido um software para o *cluster* do Laboratório de Processamento Paralelo Aplicado do Instituto de Física de São Carlos, software este com algumas ferramentas necessárias para o *cluster*.

Todos os softwares mencionados acima, assim como outros existentes no mercado, se preocupam basicamente com tarefas como gerenciamento e escalonamento de processos. Nosso trabalho tem como principal propriedade a preocupação em simplificar as tarefas mais comuns de administração e uso do *cluster*. Este software proposto possui ambiente *batch*, necessário para submeter, monitorar e manter os processos. Também possui apoio à computação paralela, indispensável neste caso, uma vez que o *cluster* em questão é utilizado para computação paralela. Outra característica é a interface Web, o que torna possível ao usuário submeter, monitorar e manter o processo a partir de qualquer navegador Web, desde que este usuário esteja autorizado. O software proposto é de fonte aberta, permitindo a alteração de suas funcionalidades, por qualquer programador que conheça a linguagem Perl utilizada, sempre que a necessidade de alteração seja detectada.

Como proposta para futuros estudos, pensamos em verificar uma possível integração entre a interface por nós desenvolvida e alguns dos softwares desenvolvidos por outras entidades, além de tratar as características citadas no capítulo anterior. Dentre as características a serem tratadas, destacam-se o balanceamento de carga, *checkpointing* e migração de processos.

Uma característica que este software possui, mas precisa ser melhorada é a monitoração de processo. O software proposto permite ao usuário verificar o andamento do processo. Mas não possui a ferramenta que permite cancelá-lo e reescaloná-lo.

Referências Bibliográficas

[1] Amoeba
http://www.cs.vu.nl/pub/amoeba/ (visitada em 20 de agosto de 2002).

[2] ANDERSON, T. E.; CULLER, D. E.; PATTERSON, D.A. ***A Case for NOW (Network of Workstations)***. IEEE Micro, 1995.

[3] ARABE, J. N. C.; BEGUELIN, A.; LOWEKAMP, B.; SELIGMAN, E.; STARKEY, M.; STEPHAN, P. ***Dome: Parallel Programming in a Heterogeneous multi-user environment***. Carnegie Mellon University, 1995.

[4] Apache Web Server, The Apache Foundation
http://www.apache.org (visitada em 20 de agosto de 2002).

[5] BAKER, Mark. ***Cluster Computing White Paper, Versão Final***. University of Portsmouth, UK, 2000.

[6] BAKER, M.; BUYYA, R.; LAFORENZA, D. ***The Grid: International Efforts in Global Computing.***

[7] BAKER, M.; FOX, T. C.; YAU, H. W. ***Cluster Computing Review***. Syracuse University, 1995.
http://www.npac.syr.edu/techreports/hypertext/sccs-748/cluster-review.html (visitada em 07 de agosto de 2002).

[8] BASNEY, J.; LIVNY, M. ***Deploying a high-throughtput computing cluster. High Performance Cluster Computing***. R. Buyya (Ed) vol. 1, Prentice-Hall, 1999.

[9] Beowulf, Scyld Computing Corporation
http://www.beowulf.org/ (visitada em 20 de agosto de 2002).

[10] BSP, University of Oxford Parallel Applications Centre
http://oldwww.comlab.ox.ac.uk/oucl/oxpara/bsp/bspmodel.htm (visitada em 20 de agosto de 2002).

[11] BURNS, G.; DAOUD, R.; VAIGL, J. ***LAM: An open cluster environment for MPC***. Ohio Supercomputer Center, 1994.

[12] CGI
http://hoohoo.ncsa.uiuc.edu/cgi/overview.html (visitada em 20 de agosto de 2002).

[13] CGI: Common Gateway Interface
http://www.w3.org/CGI/ (visitada em 21 de agosto de 2002).

[14] CGI Environment Variables
http://hoohoo.ncsa.uiuc.edu/cgi/env.html (visitada em 20 de agosto de 2002).

[15] CODINE (Computing in Distributed Network Environment), Humboldt-Universität zu Berlin
http://www.hu-berlin.de/rz/compsv/codine.html (visitada em 07 de agosto de 2002).

[16] CONDOR, Computer Sciences Department, University of Wisconsin.
http://www.cs.wisc.edu/condor (visitada em 07 de agosto de 2002).

[17] CULLER, D. E.; SINGH, J. P.; GUPTA, A. Parallel Computer ***Architecture: a Hardware/Software Aproach***. Morgan Kaufmann, 1998.

[18] Distributed Computing Environment Overview (DCE)
http://www.opengroup.org/dce/info/papers/tog-dce-pd-1296.htm (visitada em 20 de agosto de 2002).

[19] Distributed Queuing System (DQS)
http://www.scri.fsu.edu/~pasko/dqs.html (visitada em 20 de agosto de 2002).

[20] Dome
http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/nectar-adamb/web/Dome.html (visitada em 20 de agosto de 2002).

[21] DWIGHT, J.; ERWIN, M.; NILES,R. ***Using CGI, Second Edition***. QUE Corporation, 1997.

[22] FOSTER, I. ***Designing and building parallel programs***. Addison-Wesley, 1995.

[23] Generic Network Queuing System
http://www.gnqs.org/ (visitada em 20 de agosto de 2002),

[24] GLU
http://www.csl.sri.com/GLU.html

[25] GREEN, T.P.; SNYDER, J. ***DQS, a Distributed Queueing System***. Florida State University, 1993.

[26] HARLAN, D.; POWERS, S.; DOYLE, P.; FÓGHLÚ, M. Ó. ***Using Perl 5 for Web Programming.*** QUE Corporation, 1996.

[27] HAZLEWOOD, Victor. ***Cluster Computing: A Survey and Tutorial***. Miller Freeman, Inc; 1997.

[28] HEBERT, J. ***Generic NQS – Free Batch Processing for UNIX***. University of Sheffield.

[29] HENDERSON, R.; TWETEN, D. ***Portable Batch System: External Reference Specification***. NASA Ames Research Center, 1996.

[30] JAGANNATHAN, R.; DODD, C.; AGI, I. ***GLU: A high-level system for granuler data-parallel programming.*** Concurrency: Practice and Experience, 9(1), 1997.

[31] KAPLAN, J. A.; NELSON, M. L. ***A comparison of Queueing, Cluster and Distributed Computing Systems***. NASA Langley Research Center, 1994.

[32] LAM
http://www.lam-mpi.org/ (visitada em 21 de agosto de 2002).

[33] MULLENDER, S.J.; VAN ROSSUM, G.; TANEMBAUM, A.S.; VAN RENESSE, R.; VAN STAVEREN, H. ***Amoeba: A Distributed Operating System for the 1990s***. IEEE Computer, 23(5), 1990.

[34] Network Queueing System
http://umbc7.umbc.edu/nqs/nqsmain.html (visitada em 20 de agosto de 2002).

[35] Portable Batch System (PBS)
http://pbs.mrj.com (visitada em 07 de agosto de 2002).

[36] STERLING, T.; SAVARESE, D.; BECKER, D.J., DORBAND, J.E., RANAWAKI, U.A.; PACKER, C.V. ***Beowulf: A Parallel Workstation for Scientifc Computation***. Proc of the 24th Intl. Conf. on Parallel Processing, 1995.

[37] The Berkeley NOW Project
http://now.cs.berkeley.edu

[38] TURCOTTE, L. ***Cluster Computing. Parallel and Distributed Computing Handbook, Chapter 26***. McGraw-Hill, 1996.

[39] Tutorial CGI/Perl
http://www.aadsoft.com/cgi-perl/cgi-perl-tutorial.htm (visitada em 20 de agosto de 2002).

[40] VALIANT, L. G. ***A Bridging Model for Parallel Computation***. Comm.. ACM, 33(8), 1990.