Gabriel Madeira Pessoa

Proposta de Middleware Para Meta-Escalonamento em Ambientes Multi-Cluster

Florianópolis

Novembro 2010

Gabriel Madeira Pessoa

Proposta de Middleware Para Meta-Escalonamento em Ambientes Multi-Cluster

Monografia submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciências da Computação.

Orientador: Prof. Dr. Mário Antônio Ribeiro Dantas

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Florianópolis

Novembro 2010

Monografia de graduação sob o título *"Proposta de Middleware Para Meta-Escalonamento em Ambientes Multi-Cluster"*, defendida por Gabriel Madeira Pessoa e aprovada em 05 de novembro de 2010, em Florianópolis, Santa Catarina, pela banca examinadora constituída pelos professores:

Prof. Dr. Mário Antônio Ribeiro Dantas
Universidade Federal de Santa Catarina
Orientador

Prof. Dra. Lúcia Helena Martins Pacheco
Universidade Federal de Santa Catarina
Membro da Banca

Prof. Dra. Luciana de Oliveira Rech
Universidade Federal de Santa Catarina
Membro da Banca

Agradecimentos

Primeiramente agradeço a meus pais, Ilzo e Iza, por sempre me apoiarem na escolhas, sejam as da vida, ou a de fazer Ciências da Computação, mesmo isso signficando ter que ficar tão longe deles. O fato de sempre acreditarem que eu era capaz, me motivou a deixá-los orgulhosos.

Agradeço ao professor Mário Dantas pela oportunidade de trabalhar com ele e de desenvolver este projeto. Além dele, agradeço também ao pessoal do LaPeSD pelo apoio e ajuda nesse ano de convivência.

Agradeço também aos meus amigos, não muitos, mas também fico feliz em dizer que não poucos. Os de minha infância me fazem lembrar quem eu era e de onde vim, e de não perder nada disso pelo caminho, e por nunca me esquecerem. Os de minha adolescência e vida adulta, por estarem do meu lado quando mais eu precisei, por me darem força quando fraquejei, e por ficarem felizes quando comemorei.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

A área da computação de alto desempenho busca continuamente por alternativas que integrem as diversas opções de ferramentas computacionais e arquiteturas físicas que utiliza, e melhor aproveitem a soma de seus recursos, esta é uma tarefa complexa, pois muitas das aplicações nesse ramo atuam sobre sistemas heterogêneos que às vezes são muito diferentes em sua composição individual. Este trabalho tem como objetivo apresentar o primeiro protótipo de uma alternativa gratuita, de código aberto, e amigável ao usuário de ferramenta de meta-escalonamento para ambientes de multi-clusters computacionais. Serão apresentados os conceitos por trás dessa ideia, sua proposta de implementação, seus resultados e um estudo de sua viabilidade.

Palavras-chave: Multi-Cluster Computacional, Computação de Alto Desempenho, Meta-Escalonamento.

Abstract

The area of high performance computing continuously search for alternatives that integrates the various options of computational tools and physical architectures used, and better utilize the sum of its capabilities, this is a complex task because many of the applications operate on heterogeneous systems that are sometimes very different in their individual composition. This paper aims to present the first prototype of a open source, and user-friendly free alternative tool for meta-scheduling for multi-cluster computing environments. The concepts behind this idea will be presented, then its proposal of implementation, its results and a viability study.

Keywords: Computacional Multi-Cluster, High Performance Computing, Meta-Scheduling.

1 Introdução

Sistemas computacionais de alto desempenho tem sido frequentemente utilizados como ferramentas para viabilizar a realização de pesquisas nas mais diversas áreas de conhecimento. Muitas vezes, o avanço das pesquisas nessas áreas é dependente da existência de sistemas computacionais que sejam capazes de resolver determinadas tarefas em tempo hábil. Frequentemente, a capacidade computacional dos computadores convencionais não é suficiente para a realização de tarefas necessárias, seja para a realização de cálculos excessivamente complexos ou para a realização de processamento de grandes quantidades de dados, necessários em áreas de pesquisa como astronomia, meteorologia, simulações científicas, dentre outras. Em muitos casos, supercomputadores são utilizados de modo a suprir tal demanda. Porém, tais sistemas apresentam um custo muito elevado, tornando-se inviáveis para muitas instituições de pesquisa e empresas. Uma dentre as soluções mais utilizadas atualmente são os clusters computacionais, que são conjuntos de computadores convencionais interconectados por redes de alto desempenho de modo a obter capacidades de processamento semelhantes aos supercomputadores. Assim, através da união de componentes baratos é possível oferecer sistemas de alto poder de processamento por um baixo custo.

A utilização de clusters computacionais tem sido uma das alternativas mais utilizadas para a realização de tarefas com complexidade computacional alta. Para o desenvolvimento da camada de software responsável pela realização das tarefas, existem middlewares e APIs que oferecem facilidades para o desenvolvimento de aplicações paralelas para execução em clusters. Algumas das tecnologias mais utilizadas nesses ambientes são o middleware *PVM (Parallel Virtual Machine)*, bibliotecas de funções que seguem o padrão *MPI (Message Passing Interface)* e sistemas rodando *MOSIX*. Através dessas tecnologias, é possível a implementação de programas paralelos para execução em clusters de forma padronizada, utilizando a passagem de mensagens como forma de comunicação entre processos residentes em máquinas distintas.

A grande disponibilidade de computadores pessoais, seja em empresas, universidades e até mesmo nas residências, gerou uma nova oportunidade para compartilhamento de recursos computacionais. Pesquisas têm sido conduzidas de modo a buscar a otimização do uso dos recursos

computacionais disponíveis. O aproveitamento de recursos ociosos cedidos voluntariamente é uma realidade, sendo utilizado para a computação de dados relativos a projetos matemáticos, genéticos, físicos, dentre outros (BOINC, 2010). Outra área de pesquisa que tem apresentado resultados relevantes para a solução de problemas de alta complexidade computacional é a computação oportunista, que consiste na utilização de recursos computacionais ociosos. Com essa solução, o poder de processamento de estações de trabalho comuns em ambientes corporativos pode ser utilizado durante os períodos em que tais máquinas permanecem desocupadas, visando a cooperação em prol da conclusão de uma tarefa que necessite de um grande poder computacional para que seja resolvida. Nesse contexto, o uso de escalonadores de recursos é bastante importante, visto que esses componentes são responsáveis pela delegação de tarefas para as máquinas que compõem o ambiente a ser aproveitado. Quanto melhor a taxa de utilização dos recursos disponíveis, maior será o benefício obtido. Assim, o escalonador de recursos assume um papel crucial no gerenciamento da execução de tarefas em sistemas distribuídos de alto desempenho, realizando escalonamento de tarefas, balanceamento de carga entre as máquinas, dentre outras atividades.

1.1 Justificativa

O uso de ambientes multi-clusters pode ser uma solução para que aplicações específicas de organizações alcancem alto desempenho. Entretanto, para que o sistema obtenha um poder computacional diferencial, é importante que haja um controle eficiente no compartilhamento de recursos de forma que maximize a utilização dos mesmos, aproveitando recursos ociosos dos diferentes domínios. Um melhor aproveitamento dos recursos e balanceamento de carga pode acelerar o tempo de resposta dos processos, representando melhorias no custo das aplicações. Por outro lado, a complexidade no gerenciamento destes recursos deve ser realizada de forma transparente ao usuário, para que não precise se preocupar com questões de manipulação de recursos e possa focar seus esforços na sua aplicação. Desta forma diminuem custos de tempo e trabalho.

Para execução de aplicações que usam processamento paralelo, há ferramentas como PVM e MPI que podem utilizar diversos processadores da rede e suas memórias locais. Apesar dessas ferramentas serem eficientes e convergirem no objetivo principal de processamento paralelo, há consideráveis diferenças de implementação. Desta forma, as características e funcionalidades diversas que cada um apresenta podem ser diferenciais, dependendo do tipo de aplicação, influenciando no desempenho geral. A escolha do ambiente paralelo mais eficiente para se aplicar em uma organização depende de vários fatores, como os tipos de processos, a granularidade da

aplicação, as necessidades de funcionalidades específicas dos usuários. É necessário fazer um estudo detalhado de cada ambiente e configuração para alcançar maior desempenho com menor custo.

No âmbito de escalonamento, compartilhamento e monitoramento de recursos em clusters, entre os principais gerenciadores que existem no mercado estão: *LSF* da *Platform Computing*; *Grid Engine* da *Oracle*; a família do *PBS* (*PBS Pro*, *Torque*, *OpenPBS*); e o *Condor*. Estas ferramentas se mostram eficientes com relação a proposta de gerenciamento de recursos em ambientes de clusters heterogêneos, permitindo execução de múltiplas aplicações em diversas máquinas e processadores. Entretanto, também são diferentes em implementação e não são soluções compatíveis uma com a outra, ou com muitas das arquiteturas utilizadas nas organizações. Geralmente, os escalonadores servem de interface para diferentes tecnologias, em diferentes níveis da organização. A escolha de qual escalonador utilizar implica na avaliação dos sistemas e das necessidades da organização, dependendo dos projetos e tipos de workflows que serão executados. Outra questão é o custo dessas ferramentas, sem uma análise da situação em que é aplicada, pode se tornar caro, por conta de funcionalidades extras, que não são necessárias. Ou então aumentar consideravelmente o custo por manutenção e desenvolvimento de características, regras e políticas específicas a mais, que um outro escalonador pode ter por padrão.

Com o passar do tempo, mudam e crescem as necessidades dos usuários, e estas ferramentas podem se tornar obsoletas. Por isso, também é importante a manutenção e atualização dos sistemas, além do desenvolvimento de funcionalidades específicas que atendam problemas particulares de determinadas aplicações.

Neste trabalho é feito um levantamento dos paradigmas de construção de sistemas distribuídos, e do estado da arte das tecnologias usadas nesses sitemas: ambientes paralelos, gerenciadores de recursos e meta-escalonadores, comparando avaliações feitas sobre suas disponibilidades de recursos e desempenhos. Isso leva à segunda metade deste trabalho, encontrar uma solução adequada para ambientes multi-clusters, visto que as aplicações atuais são desenvolvidas tendo como foco ambientes de cluster ou grid. Por isso, foi proposta e implementada uma ferramenta de meta-escalonamento para esse tipo de ambiente, levando em conta a diversidade de plataformas, facilidade de uso, e os princípios do código-aberto.

1.2 Objetivos

1.2.1 Objetivo Geral

Propor uma ferramenta alternativa de alto desempenho para o meta-escalonamento de processos em ambientes multi-cluster a partir do conceito de sistema de imagem única;

1.2.2 Objetivos Específicos

- Fazer o levantamento do estado da arte das tecnologias de ambientes paralelos e das ferramentas de gerenciamento de recursos;
- Avaliar qualitativamente as tecnologias estudadas para a escolha das mais adequadas para a proposta;
- Elaborar um serviço de sistema de imagem única para utilização em multi-clusters;
- Implementá-lo em um ambiente real e executar testes;
- Estudar a viabilidade do uso da ferramenta em ambientes de produção real e da implementação de novas utilidades necessárias.

1.3 Estrutura do Trabalho

Este trabalho está dividido na seguinte forma. O capítulo 2 tratará da conceituação de sistemas distribuídos e das diversas configurações que podem ser neles encontrados, também serão abordados os chamados ambientes paralelos, que são usados na programação de aplicações paralelas. O capítulo 3 lida com a gerência de recursos em ambientes distribuídos, e as principais ferramentas utilizadas na área, além do conceito de meta-escalonador, aplicado em grids e multi-clusters computacionais.

No capítulo 4 é apresentada a proposta de trabalho, ou seja, o middleware pesquisado, a configuração física, e a aplicação que será executada. O capítulo 5 serve como comparativo dos resultados das aplicações dos testes explicados no capítulo anterior. Por fim, o capítulo 6 trata das conclusões tiradas após observação dos resultados e propõe trabalhos futuros que poderão ser desenvolvidos a partir deste.

2 *Sistemas Distribuídos*

2.1 Computação Distribuída

O paradigma de computação distribuída é aquele no qual os componentes responsáveis por uma aplicação não se encontram necessariamente próximos. Hardware, pacotes de software, e até sensores podem estar em continentes diferentes e os resultados são obtidos em tempo e precisão razoáveis e com um custo relativamente baixo às partes envolvidas, ele pode ser observado no diagrama da figura 2.1.

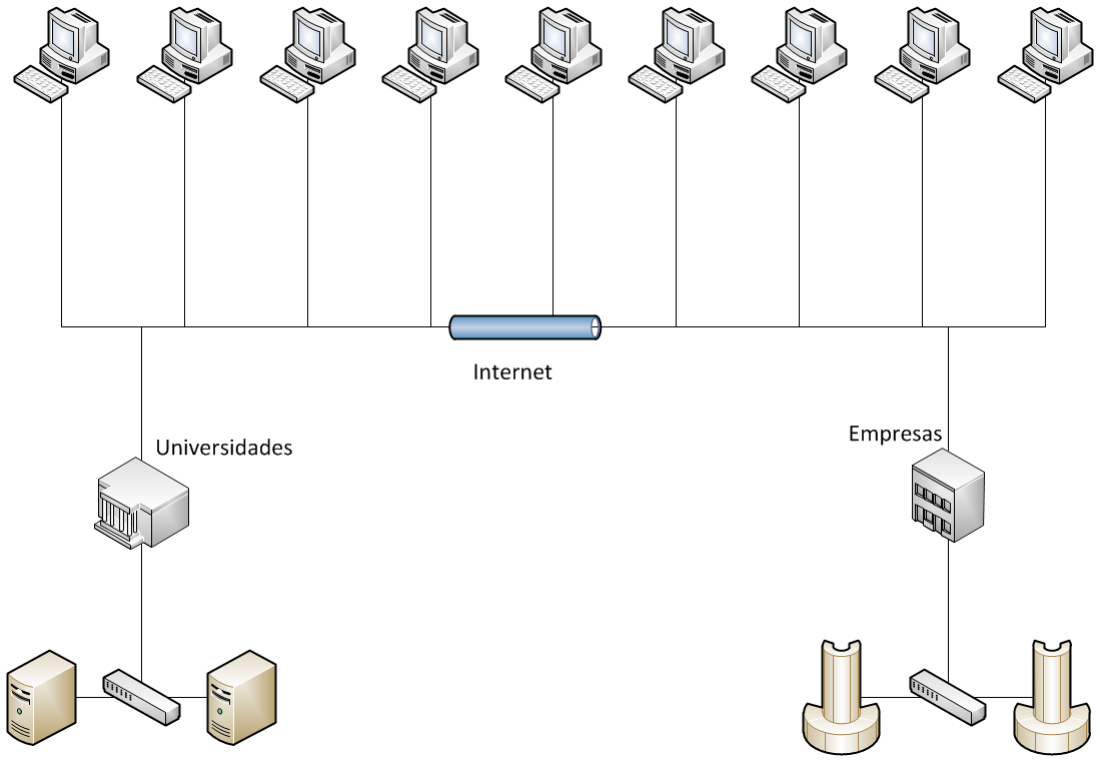


Figura 2.1: Computação Distribuída

A ideia de agregar poder computacional não é nova e já existe desde que os primeiros computadores foram ligados via rede, e com os anos sua utilização sofreu refinamentos, seja nas possibilidades de configuração, seja na forma como seus recursos são utilizados.

O conceito por trás da computação distribuída é de transformar agrupamentos de máquinas, normalmente de baixo e médio custo, em potentes centros computacionais, os quais podem resolver problemas que envolvam milhares de aplicações simultâneas, ou o processamento paralelo de tarefas complexas. A essa área específica se dá o nome de computação distribuída de alto desempenho.

Outrora, esse tipo de problema era tratado pelos chamados supercomputadores, máquinas de alto custo mas que possuíam capacidade de processamento acima daquela dos computadores de uso cotidiano. Apesar da eficiência e resultados dos supercomputadores, sempre se buscou reduzir os custos desse tipo de operação, uma alternativa sempre pesquisada foi o uso dos computadores pessoais (também chamados de *de prateleira*) que possuíam essencialmente os mesmos componentes. Os estudos feitos com esse tipo de equipamento acabou provando, com o surgimento de clusters e grids, que quando devidamente arranjados e configurados, grupos desses computadores eram capazes de atingir resultados iguais ou superiores aos dos obtidos com supercomputadores.

Deu-se assim o nascimento dos ambientes distribuídos, dentre os quais os mais conhecidos atualmente são os clusters (agregados) e as grids (malhas) computacionais, que serão descritos melhor à frente.

A alternativa logo adquiriu popularidade por seu custo-benefício, e facilidade de operação, e a possibilidade de seus recursos estarem dispersos geograficamente. Esse sucesso se mantém até hoje, visto que no ranking dos dez computadores mais potentes do mundo, três são do tipo cluster (TOP500, 2010), mas no total da listagem, somam mais de quatrocentos sistemas em uso.

Mas como todo novo paradigma, contratempos foram e continuam sendo enfrentados. Consistência de dados, garantia da execução, disponibilidade do sistema, segurança das informações e retardo de comunicação, entre outros, são questões estudadas e que se encontram com diferentes soluções em software e hardware, mas esses tópicos ainda são assunto de muitas pesquisas de grande utilidade futura.

2.2 Configurações Distribuídas

2.2.1 Meta-Computador

Ao ambiente que transmite essa sensação de único, dá-se o nome de meta-computador. Na definição mais ampla possível para computação distribuída, a união e disponibilização de

recursos de computadores através da Internet é um meta-computador, o mais rápido do mundo, por sinal (PROJECT, 2010).

Seguindo o conceito de uso da Web como um meta-computador, sempre se esbarrou na dificuldade de particionamento do problema, disponibilidade, heterogenia e segurança dos sistemas executores. Por isso, apesar de existirem iniciativas como o *Projeto Charlotte*, o foco inicial da computação distribuída foram os clusters e grids.

Em (STRUMPEN, 1995) temos as primeiras fundamentações no conceito de meta-computador sob o nome de *"Máquina de Rede"* e embora se encontre um tanto quanto datado, sua base ainda pode ser aproveitada na contextualização deste trabalho.

Na tese argumenta-se que os ambientes paralelos acabam equilibrando-se entre programas eficientes, como é o caso cientistas e engenheiros buscando resultados em tempos razoáveis, e o desenvolvimento eficiente de programas, como ferramentas de programação e modelos abstratos, e que costumam ter motivações comerciais. Apenas com iniciativas como o MPI é que questões como a portabilidade de programas paralelos começaram a ser tratadas com uma cobrança menor em relação à eficiência.

Ela também propõe o uso da Internet como um supercomputador distribuído para aplicações de granularidade grossa, ou seja, com muito processamento local, mas baixa taxa de comunicação. No exemplo descrito, em uma rede mundial de 800 estações de trabalho, um problema de análise de sequência biomolecular foi concluído em questão de minutos, enquanto que caso executado em um supercomputador teria levado dias.

Pontos levantados no trabalho envolvem a eficiência de três componentes do processador, que perde em muito desempenho devido à latência de comunicação, eficiência da rede, que deve diminuir o tempo de transferência de mensagens curtas e aumentar a saída de mensagens longas, e da memória, que lida com cópias desnecessárias de mensagens e paginação de dados.

A avaliação sobre comunicação via rede chegou a conclusão que a redução da latência não seria prática pois dependeria de alterar o sistema operacional, o que reduz a portabilidade e inviabiliza o acesso à máquinas remotas. Além disso, deve-se evitar cópias de mensagens para minimizar o gasto adicional para mensagens longas devido a acessos de memórias.

Já a observação das características dos sistemas operacionais mostrou que nos chamados micro-kernels, que apesar do nome ainda tem um tamanho que se torna relativamente grande quando todo o ambiente é levado em conta, threads são a escolha ideal para a estrutura de computações de granularidade média e grossa em nível de usuário, e que o uso de chamadas de sistema encarece o processo pois é necessário atravessar a fronteira entre nível de usuário e

nível de kernel.

Na proposta, a Máquina de Rede é definida como uma máquina virtual acessível tanto pelo sistema runtime quanto pela interface com o usuário, algo não muito diferente do PVM (SUNDERAM, 1990). Essa divisão serve basicamente para a proteção dos processos, que rodam exclusivamente na máquina virtual ou no sistema, e necessitam de uma interface de comunicação inter-processos para trocarem mensagens. Nessa perspectiva, funcionalidades críticas podem ser implementadas em qualquer uma das duas camadas, dependendo da necessidade, seja facilidade de acesso, ou redução da complexidade do sistema no nível do usuário.

A máquina virtual tem como função primária fornecer um único ambiente de programação em todos os componentes da Máquina de Rede, além de oferecer visão, controle e gerenciamento da Máquina de Rede durante a operação. Nesse ponto ela se difere do PVM pois este apenas permite controle mínimo sobre os processos e a obtenção de dados do ambiente, mas não acesso geral consistente. O outro componente básico, o sistema runtime é usado para abrigar características críticas para o desempenho de aplicações paralelas para evitar mudança de contexto com o sistema operacional. Além de também poder fornecer abstrações de programação sobre uma interface de passagem de mensagens.

Apesar de oferecer uma visão geral sobre os principais tópicos que devem ser lidados no âmbito do uso de uma Máquina de Rede, o objetivo exato da tese foi de testar um modelo de ocultamento de latência de comunicação, que se mostrou eficiente dado um número processadores no sistema que opera em condições ótimas graças ao modelo apresentado.

2.2.2 Clusters Computacionais

O cluster computacional é uma das configurações mais utilizadas na computação distribuída de alto desempenho, podendo ser definido como um agrupamento físico, local ou virtual, de inúmeros computadores, chamados aqui de nós ou nodos, com o objetivo de agregar recursos computacionais para disponibilizá-los para a melhoria de aplicações (DANTAS, 2005, p. 147-148).

Fruto direto das primeiras pesquisas na área, até hoje muitos clusters são basicamente computadores pessoais conectados entre si e que compartilham CPU, disco rígido e RAM para fins de processamento. Eventualmente, empresas como *Cray* e *NEC* começaram a manufaturar sistemas especializados que utilizam o conceito de clusterização, mas com o diferencial do renome e do know-hall.

Apesar de não haver uma forma unificada de classificação dos clusters, alguns aspectos de

suas configurações podem ser usados para distingui-los, basicamente em torno do hardware e software utilizados e da função dos componentes individuais do ambiente, conhecidos como nós ou nodos.

Um ambiente composto por computadores de mesma configuração e com os mesmos programas instalados é chamado de homogêneo, e se beneficia da facilidade de adição de novos nós e interoperabilidade entre eles. Já quando máquinas diferentes são agregadas, encaramos um cluster heterogêneo, que pode ser mais prático durante a montagem do ambiente, mas traz a complicação da necessidade de transparência para as aplicações de um sistema único, e não vários componentes diferentes. A figura 2.2 exemplifica um cluster homogêneo dentro de uma instituição.

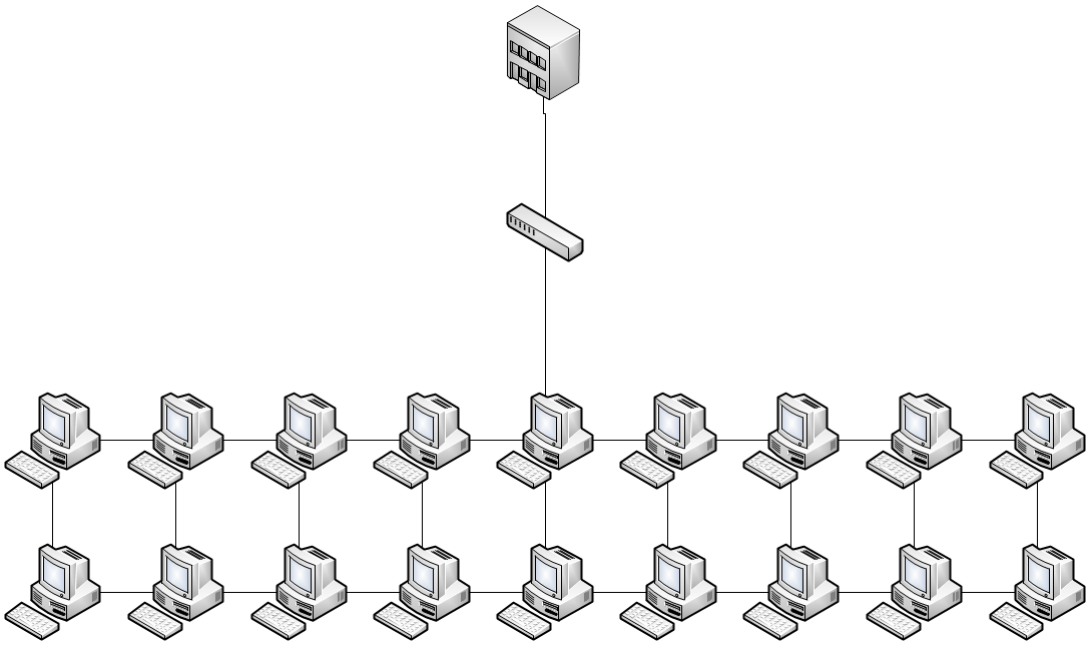


Figura 2.2: Exemplo de Cluster

Quando as máquinas do agregado utilizam apenas o seu tempo ocioso para a execução das aplicações, o ambiente é chamado de não-dedicado e tira proveito de um melhor custo-benefício, pois as estações de trabalho já existem, porém são necessárias políticas de escalonamento, segurança, e não existe a certeza de disponibilidade dos nós. Por isso, para aplicações vitais a configuração dedicada, na qual os nós são utilizados apenas para fins de processamento, é preferível, pois os middlewares aplicados ao cluster encontram menos problemas de gerenciamento, além de facilidade na escalabilidade do ambiente, tornando a computação mais efetiva e precisa.

2.2.3 Grids Computacionais

Uma evolução natural dos clusters foram as grids computacionais. Quando as instituições começaram a pesquisar formas de estender o limite geográfico dos clusters, que até o momento atingiam as fronteiras das próprias instituições, perceberam que da mesma forma que alguns computadores podiam ser conectados para compartilhar recursos, os agregados podiam ser ligados uns aos outros, criando nodos em uma grande malha que poderia ter alcance e escalabilidade muito superiors aos dos agregados.

Uma grid pode ser vista como um ambiente de componentes heterogêneos e geograficamente dispersos e com uma quantidade de recursos e serviços superior a de outras configurações distribuídas. Isso torna a questão da consistência do sistema ainda mais importante, dado que não existe um nó mestre coordenando a submissão, escalonamento e verificação dos processos, uma exemplificação de grid pode ser vista na figura 2.3.

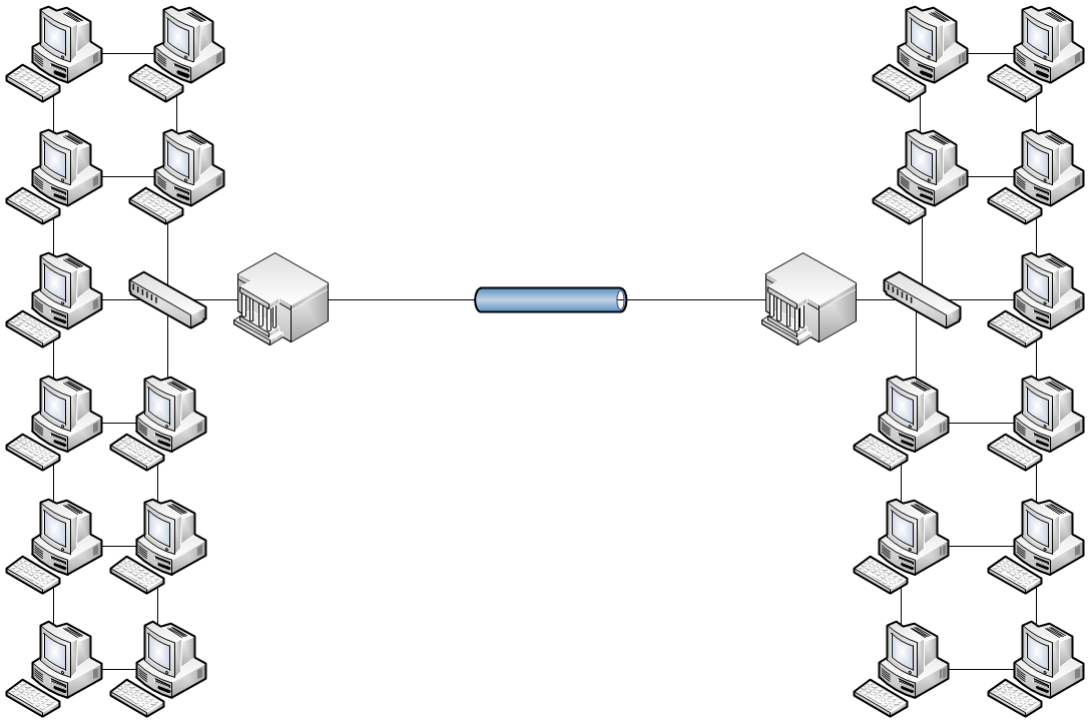


Figura 2.3: Exemplo de Grid

Segundo (FOSTER, 2002), um ambiente de grid deve coordenar os recursos disponíveis que não estão sob um gerenciamento centralizado, o que faz com que muitas das soluções hoje disponíveis na verdade servem para a estruturação de clusters, pois são compostas por gerenciadores locais. Deve usar protocolos e interfaces padronizados, abertos e de propósito geral, já que deve integrar diferentes domínios de máquinas, com suas próprias políticas de disponibilidade de recursos, autenticação e autorização. Por fim, uma grid deve entregar diferentes

qualidades de serviços em diferentes áreas, seja tempo de resposta, throughput, disponibilidade e outros, para que a utilidade da soma dos componentes seja maior do que a deles individualmente.

Alguns pesquisadores consideram a Internet uma grid global com uma vasta gama de recursos disponíveis, sejam eles dedicados, ou apenas o tempo ocioso de máquinas pessoais. Esse último é conhecido também como *configuração oportunista*, que em toda amplitude da malha, usa apenas os recursos ociosos dos computadores que a formam para reduzir o custo da computação, como os projetos *SETI@Home* e *Folding@Home*. Apesar dessa visão otimista, os pontos levantados pelo artigo de Foster mostram que a Internet ainda não pode ser vista como uma grid, pois apesar de usar protocolos abertos e de propósito geral para acessar recursos distribuídos, nãoo permite o uso coordenado destes para obter qualidades de serviço interessantes.

Tal como os clusters, não existe uma classificação clara dos ambientes de grid, mas uma das características observadas é a topologia. Vários clusters dentro de uma mesma instituição, formam uma *intragrid*. Quando mais de uma instituição comunicam suas grids, há uma *extragrid*, e quando se extrapola para muitas instituições, tem-se uma *intergrid*.

Devido à complexidade inerente de se trabalhar com as grids, um esforço tem sido feito no desenvolvimento dos chamados portais de grid, que tem como objetivo servir de ponto de acesso único às informações e recursos disponíveis, além esconder os detalhes complexos do ambiente. Eles podem ter um enfoque mais genérico, sendo então chamados de portais de usuário, ou então serem aplicados em áreas específicos, os portais científicos.

Além destes, a um grande trabalho na área de ferramentas de unificação de grids, como os projetos Globus e Condor, que lidam com a parte de segurança e comunicação entre as diferentes organizações que formam a grid, de forma que os processos sejam encaminhados aos nós mais apropriados para sua execução.

2.2.3.1 Multi-Cluster

Um subconjunto interessante dos ambientes de grid é o chamado multi-cluster, um agrupamento de clusters de diferentes domínios para a execução de processos, uma exemplificação pode ser vista na figura 2.4. A grid convencional se distingue do multi-cluster por geralmente serem altamente heterogêneas, distribuídas entre várias organizações e composta por diversos tipos de sistemas operacionais, serviços, hardware e dispositivos de variados provedores.

Considera-se também que cada cluster componente trabalhará com sua própria fila de submissão, arquivos de registro e unidades funcionais, ou seja, as instruções executadas em um

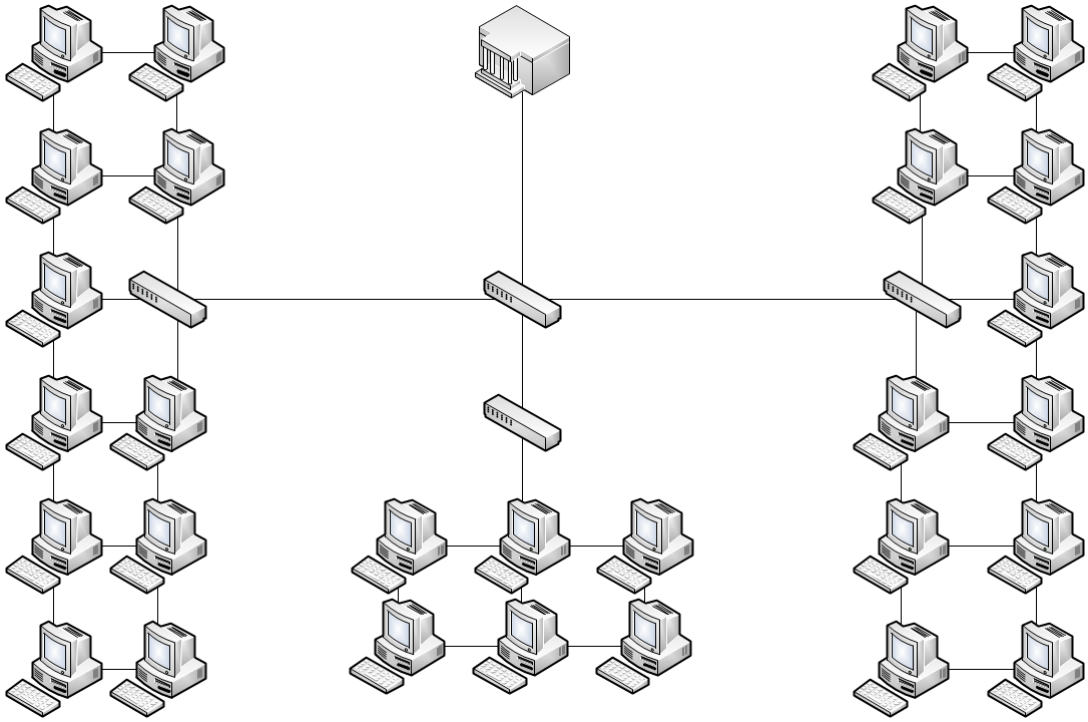


Figura 2.4: Exemplo de Multi-Cluster

cluster foram submetidas pelo nodo mestre dele, e apenas elas podem ler/escrever fisicamente nele. Entretanto, as instruções são obtidas de uma cache única e globalmente acessível.

Há consideráveis benefícios no uso de um multi-cluster. Basicamente, cada cluster realiza menos operações de leitura/escrita por ciclo, por isso ocorrem menos acessos aos arquivos de registro. Com menos operações de leitura/escrita há a necessidade de uma lógica de escalonamento de instruções menos complexa.

Além disso, o multi-cluster trabalha sobre políticas similares de uso de recursos e escalonamento, ou seja, há a necessidade de um gerenciador global de recursos, que por sua vez, deve ser capaz de compreender as diversas formas de descrição do que cada organização ou domínio disponibiliza para o ambiente.

Nesse ponto o trabalho recente de (JANSON et al., 2010) propõe o uso de ontologias e lógica difusa para o pareamento dos recursos necessários pelas requisições com os disponíveis entre as organizações que compõe o ambiente. Essa proposta ainda trabalha com o princípio junção dinâmica de recursos, ou seja, se um cluster isolado não puder responder a uma requisição, combinações de componentes de vários clusters também são levadas em conta na tentativa de atender o pedido. O middleware apresentado também traz diversas características interessantes quando se trata do ambiente estudado, como integrar as diferentes organizações virtuais, atualização dinâmica do ambiente, co-alocação, reserva antecipada e a descrição de

diversos recursos diponíveis

Outro trabalho interessante na área é apresentado em (BARRETO; AVILA; NAVAUX, 2000) onde levanta-se a questão das diferentes redes de interconexão entre os clusters, que eles tratam através de arquivos descritores dos nodos, fazendo com que comunicação intra-nodos seja realizada por memória compartilhada, e inter-nodos por passagem de mensagens. Outro foco do artigo é o *DECK (Distributed Executive Communication Kernel),* uma interface entre o programador e o ambiente, que fornece serviços e abstrações úteis para a criação de aplicações distribuídas e paralelas que rodam de forma *SPMD (Single Program, Multiple Data)* sob os diversos nodos. O escopo desse trabalho, entretanto, restrito a apenas resolver as questões de comunicação entre um cluster *Myrinet* e outro *SCI*, mas os resultados obtidos mostravam que as soluções utilizadas eram comparativamente mais simples do que as usadas em ferramentas já em uso como o Globus.

2.3 Ambientes Paralelos

2.3.1 PVM - Parallel Virtual Machine

Desenvolvido pela Universidade do Tennessee, Laboratório Nacional de Oak Ridge e pela Universidade Emory, o PVM (PVM, 2010) é um conjunto de ferramentas integradas que oferece solução para computação concorrente heterogênea. Através da utilização do PVM, é possível utilizar um conjunto heterogêneo de computadores de modo que estes cooperem entre si em prol da solução de uma determinada tarefa, compartilhando recursos computacionais através de uma rede (SUNDERAM, 1990). Em geral, é utilizado para paralelização de aplicações que demandam tempos de execução muito altos quando se utiliza uma abordagem sequencial. O nome, Parallel Virtual Machine, se deve ao fato de o PVM utilizar uma abordagem na qual uma coleção de máquinas heterogêneas formam uma única máquina paralela virtual, de modo que seus usuários podem interagir com ela como se estivessem lidando com um computador paralelo único.

O PVM, por ser uma tecnologia considerada madura e estável, apresenta um reduzido número de lançamentos nos últimos anos. Intervalos de até cinco anos entre uma versão e sua sucessora são comuns. Por exemplo, a última versão, PVM 3.4.6, lançada em fevereiro de 2009, possui uma diferença de quase cinco anos para a versão anterior a ela, PVM 3.4.5, lançada em setembro de 2004. Além disso, com o passar da última década, percebeu-se uma queda substancial na quantidade de pesquisas científicas relacionadas ao uso/desenvolvimento do PVM, vide as publicações do evento anual de encontro de usuários e desenvolvedores de PVM e MPI,

EuroPVM/MPI (EUROPVM/MPI, 2010), um dos mais importantes da área. Tanto o ciclo de desenvolvimento, quanto as pesquisas realizadas recentemente indicam que novas características e melhorias não serão muito frequentes. Assim, apesar de o PVM ser considerado uma tecnologia bastante estável, ele pode ser descontinuado ou manter um processo de desenvolvimento pouco ativo, perdendo em novas funcionalidades e em portes para novas arquiteturas.

2.3.2 MPI - Message Passing Interface

Diferentemente do PVM, que é um conjunto de ferramentas para computação paralela, construído e mantido por universidades e laboratórios de pesquisa, MPI (MPI, 2010) é a especificação formal de uma API padrão para desenvolvimento de aplicações paralelas através do modelo de passagem de mensagens. MPI é um padrão de facto para o desenvolvimento de aplicações paralelas utilizando o modelo de passagem de mensagens (SQUYRES; LUMSDAINE, 2003) para execução em ambientes como clusters, onde os recursos computacionais são interligados por uma rede, seguindo uma abordagem de memória distribuída. Sendo apenas uma especificação, MPI se torna independente de linguagem, possuindo implementações e bindings para várias linguagens de programação. Devido a independência de linguagem, existem diversas implementações do padrão MPI, desenvolvidas por diferentes fabricantes/pesquisadores, como por exemplo, MPICH, LAM/MPI e a mais recente, OpenMPI, sendo esta uma evolução da implementação LAM/MPI.

A padronização de uma interface para comunicação entre máquinas para computação paralela possui como vantagens a facilidade de utilização e portabilidade. Facilidade de uso, pois existe uma interface bem definida, com funções específicas para a realização de determinadas tarefas. Portabilidade, pois se mantém um padrão para comunicação, abstraindo detalhes de baixo nível das implementações. Um fórum de desenvolvedores e fabricantes (OPENMPI, 2010) é responsável pela definição da interface e funcionalidades que esta deve seguir, tendo evoluído bastante nos últimos anos.

Observando as versões existentes, é perceptível que o padrão está em constante evolução, incorporando novas características de acordo com as necessidades. Esse é um ponto bastante importante, visto que os sistemas computacionais, tanto hardware como software passam por frequentes mudanças. Um exemplo são as melhorias que as arquiteturas vem sofrendo, com um uso massivo de processadores multi-core, onde existem muitas possibilidades para paralelização de acesso a recursos locais de uma máquina componente de um cluster.

Seguindo o padrão definido na especificação, existem algumas implementações que são largamente utilizadas para computação de alto desempenho. A seguir serão descritas duas das

principais: *MPICH2* e *OpenMPI*.

2.3.2.1 MPICH2

MPICH2 é uma implementação do padrão MPI-2.1, utilizado em várias plataformas, como Linux, Mac OS/X, Solaris e Windows. Focando principalmente no alto desempenho, MPICH2 ainda não oferece suporte a execução em clusters com máquinas com diferentes representações de dados, embora a incorporação dessa característica esteja dentro dos planos de desenvolvimento (MPICH2, 2010).

O MPICH2 é uma tecnologia que está em constante desenvolvimento. Diariamente, uma equipe de desenvolvedores realiza a correção de bugs, inserção de melhorias e novas funcionalidades na base de código (como pode ser visto na página de desenvolvimento (MPICH2A, 2010)). Além da versão oficial do MPICH2, existem alguns produtos derivados deste, criados por empresas e universidades, visando a implementação de otimizações para determinadas plataformas, bem como personalizações. Alguns exemplos são: Intel MPI Library (INTEL, 2010), MVAPICH2 (MVAPICH, 2010) (para plataformas com tecnologia de rede InfiniBand, 10GigE/iWARP ou RDMAoE), MPICH2-MX (MPICH2-MX, 2010) (para tecnologias de rede Myricom), dentre outros.

A figura 2.5 apresenta a arquitetura do MPICH2, que, assim como o OpenMPI, também apresenta uma arquitetura com as implementações de suporte a dispositivos específicos separadas em camadas.

2.3.2.2 OpenMPI

OpenMPI (OPENMPI, 2010), como o nome já indica, é uma implementação aberta do padrão MPI-2, desenvolvida e mantida por um conjunto de instituições de pesquisa e de indústria. Foi inicialmente concebido pelos desenvolvedores de outras implementações MPI (FT-MPI, LA-MPI, LAM/MPI, PACX-MPI), de modo a criar uma implementação aberta na qual os esforços e base de conhecimento anteriores pudessem ser combinados para formar uma implementação sem necessidade de divisões, como ocorre com o MPICH2, no qual existem vários forks para outros projetos específicos. A idéia é que seja mantida uma implementação abrangente o suficiente para evitar o surgimento de projetos paralelos ao OpenMPI, de modo que todos os esforços de desenvolvimento sejam concentrados no projeto principal, incorporando ao projeto as novas funcionalidades que, em outro caso, seriam inseridas apenas nos projetos paralelos.

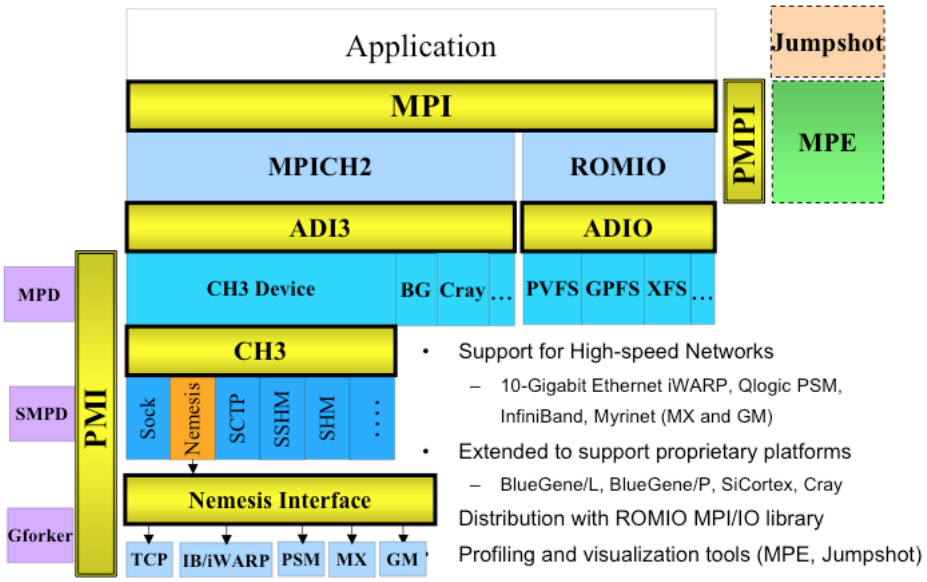


Figura 2.5: Arquitetura do MPICH2, de (LABORATORY, 2007)

Seguindo a idéia já bastante difundida na indústria de desenvolvimento de software baseado em componentes, o OpenMPI foi desenvolvido utilizando a *MPI Component Architecture (MCA)*. A arquitetura do OpenMPI é dividida em três áreas funcionais:

- *MCA:* Responsável pela arquitetura de componentes, gerencia e provê serviços aos componentes.
- *Componentes:* Cada um é dedicado a uma tarefa única, carregam e utilizam os módulos.
- *Módulos:* São programas que implementam determinadas funcionalidades para serem utilizadas pelos componentes.

Assim, através dessa arquitetura modularizada, é possível a implementação de suporte a novos dispositivos sob a forma de módulos, facilitando o desenvolvimento e lançamento destes, evitando a criação de forks do projeto para implementação de suporte ou otimizações para uma determinada arquitetura.

Assim como o padrão MPI, o OpenMPI vem sofrendo constantes melhorias, com um ciclo de desenvolvimento bastante ativo. Ao contrário do PVM, o OpenMPI ainda não se encontra

em um estado completamente maduro e estável, sendo constante a liberação de novas versões (OPENMPIA, 2010), com correção de bugs e inserção de novas funcionalidades. Apesar disso, o fato de o OpenMPI ser software aberto, com um alto grau de participação da comunidade e de estar em constante desenvolvimento, faz com que a correção de bugs seja mais dinâmica (OPENMPIB, 2010), bem como a inserção de novas características. O suporte aos usuários e desenvolvedores também é um ponto forte, visto que o OpenMPI apresenta uma lista de discussões bastante responsiva e ativa (OPENMPIC, 2010). Além do projeto principal, o OpenMPI possui diversos sub-projetos, que buscam acrescentar funcionalidades ao projeto principal, como por exemplo, suporte a afinidade de processador, suíte de testes, dentre outras.

2.3.3 MOSIX

MOSIX é uma melhoria do BSD/OS que faz uso de algoritmos para compartilhamento de recursos de forma adaptativa, sendo implementado em camada de software e que permite que aplicações rodem em nodos remotos como se estivessem executando localmente (BARAK; LA'ADAN, 1998). A versão mais recente do MOSIX permite a estruturação de clusters, multi-clusters e ainda traz a interessante característica da possibilidade de ser montado um cluster de GPUs, tirando proveito da potência computacional matemática desses processadores. A figura 2.6 demonstra o funcionamento da funcionalidade de migração de processos.

A transparência do sistema executando o MOSIX se dá principalmente porque ele realiza busca dinâmica de recursos e opera em nível de processo, ou seja, o sistema adapta a carga quando o número de processos, e/ou sua demanda, são alterados. Há a garantia de segurança da execução, pois os processos migrados rodam em ambiente de sandbox. A questão da disponibilidade é resolvida pois processos rodando em nodos que são desconectados, são antes migrados para outros nodos.

Há a possibilidade de operação em ambientes heterogêneos, visto que ele pode ser executado tanto nativamente em sistemas Linux, obtendo ganho de desempenho, mas sendo necessário alterações no kernel, quanto via máquina virtual em Windows, Linux e OS X, além de executar em ambientes 32 e 64-bits. Essa característica também permite a formação de multi-clusters que desfrutam dos mesmo benefícios dos clusters simples, incluindo a adição e remoção dinâmica de novos clusters, e níveis de prioridade para processos migrados de outros pontos.

O MOSIX deu origem ao openMOSIX quando se tornou proprietário em 2002, apesar de sua utilidade e disponibilidade gratuita, principalmente em aplicações paralelas e de entrada/saída intensas, acabou sendo descontinuado após a popularização dos computadores de múltiplos núcleos, atualmente seu código-fonte ainda pode ser acessado, mas não aceita mais

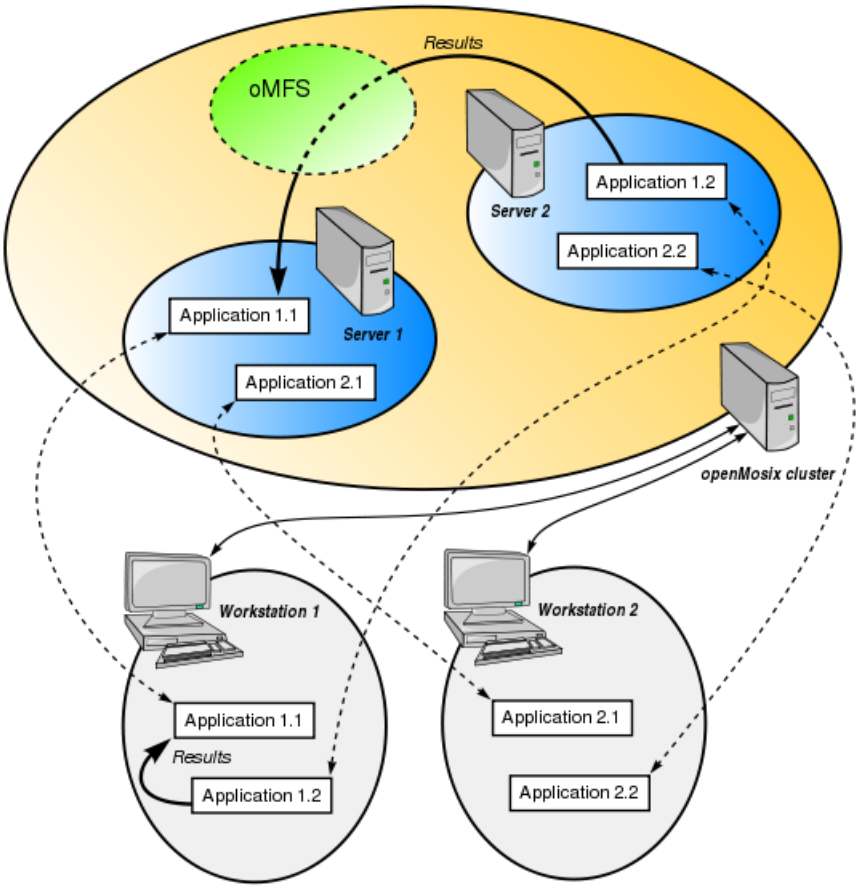


Figura 2.6: Transferência em um cluster openMosix, de (OPENMOSIX, 2010)

contribuições. Do openMOSIX por sua vez, nasceu o LinuxPMI que também tem como ideia central a de migrar processos sem que o usuário perceba que isso aconteceu.

2.3.4 Comparação

O PVM oferece suporte oficial a um conjunto maior de arquiteturas e sistemas operacionais do que as outras alternativas, sendo capaz de executar uma máquina virtual composta de computadores variados, suportando, por exemplo, AIX, HPUX, NetBSD, dentre outros não suportados oficialmente pelos outros projetos, porém o MOSIX pode rodar em virtualmente qualquer sistema operacional executado em processadores Intel x86. Enquanto as abordagens MPI focam principalmente em um ganho de performance, o PVM se concentra em oferecer suporte a heterogeneidade e interoperabilidade, exigindo camadas adicionais para transporte dos dados entre plataformas heterogêneas, abrindo mão de performance para a execução dessas operações (GEIST; KOHL; PAPADOPOULOS, 1996).

Uma questão bastante relevante quanto a escolha da tecnologia a ser utilizada para a paralelização de programas é o escalonador de processos. A tarefa do escalonador é verificar recursos ociosos ou menos sobrecarregados e encaminhar as tarefas para estes, de modo a obter a melhor taxa de utilização possível dos recursos computacionais disponíveis. Quanto a isso, o OpenMPI se destaca, sendo suportado por uma vasta gama de escalonadores, incluindo os principais e mais utilizados no mercado, como LSF, Sun Grid Engine e PBS.

Nesse ponto o MOSIX fica em seu próprio patamar, visto que ele próprio faz o balanceamento e escalonamento de processos, não necessitando de um middleware que cumpra essa função. Entretanto, ele próprio pode ser usado em conjunto com MPI e PVM para computação de alto desempenho.

As APIs de desenvolvimento de aplicações são, nas três tecnologias, oficialmente oferecidas para as linguagens C, C++ e Fortran, porém, existe um grande número de bindings dessas APIs para outras linguagens.

Uma característica importante para a escolha de uma plataforma e/ou tecnologia para o desenvolvimento de um produto é o quanto se espera que esse produto irá evoluir e oferecer melhorias aos usuários/desenvolvedores. No quesito pesquisa científica, que acabam sendo revertidas em melhorias nas tecnologias, as implementações MPI têm muita vantagem, visto que o padrão MPI vem sendo constantemente melhorado e que as tecnologias também vem apresentando novas funcionalidades. O evento EuroPVM/MPI é um exemplo real desse cenário, pois a grande maioria das pesquisas apresentadas nas últimas edições do evento são relacionadas ao padrão MPI.

2.3.5 Desempenho

Embora propostos e implementados com objetivos e arquiteturas diferentes, é possível a resolução de um mesmo problema através da utilização das três tecnologias, seja PVM, MOSIX ou alguma implementação MPI. Desse modo, apesar de diferirem em detalhes importantes, como arquitetura e API, é possível a realização de comparações quantitativas entre aplicações paralelas implementadas com base no PVM, MOSIX ou em alguma implementação MPI. Alguns trabalhos realizaram comparações entre programas paralelas utilizando os três ambientes.

Em (KITOWSKI; BORYCZKO; MOŚCIŃSKI, 1996) e (BORYCZKO et al.,), foram realizadas comparações de desempenho obtidas com implementações PVM e MPI para uma aplicação de simulação de dinâmica de moléculas. Foram utilizadas implementações proprietárias do PVM e do padrão MPI, ConvexPVM e ConvexMPI, para desenvolvimento do algoritmo par-

alelo para realização dos testes. Nesses testes, as simulações realizadas mostram uma vantagem considerável para a implementação MPI, apresentando tempos de execução mais baixos do que a implementação PVM para a mesma tarefa. Além disso, a implementação MPI se mostrou mais escalável a medida que o número total de nós foi aumentado.

Em (WERSTEIN; PETHICK; HUANG, 2003), outro trabalho que realizou comparações quantitativas entre PVM e MPICH, foram implementados três soluções paralelas em ambas as tecnologias para três diferentes classes de problemas comuns para execução em ambientes distribuídos. O primeiro experimento realizado é uma implementação do algoritmo de ordenação MergeSort, da classe dos problemas fracamente síncronos. A abordagem MPI obteve os melhores resultados nos testes de speedup obtido com a paralelização do problema. Em outro teste realizado, geração de fractais de Mandelbrot, da classe dos problemas massivamente paralelos, a implementação MPI obteve leve vantagem sobre a implementação PVM em termos de speedup. Porém, os resultados são muito próximos. Em experimentos envolvendo redes neurais, com problemas da classe síncrona, a implementação MPI obteve resultados de speedup significativamente melhores que a implementação PVM. Além disso, foram comparados os tempos de execução de operações de broadcast em ambas as implementações. Nesse ponto, a abordagem MPI se mostrou muito mais eficiente que o PVM, obtendo tempos de execução até 20 vezes menores.

Em (BARAK et al., 2002) foram feitas diversos testes de carga comparando MOSIX com migração de processos e PVM. A primeira avaliação consistiu na avaliação no algoritmo de balanceamento de carga do MOSIX, o que mostrou que ele pode ser até 30% mais rápido que o PVM, e o atraso se comparado com o tempo de execução ótimo é quase irrelevante. Nesta situação também foi testado o PVM executando junto com o MOSIX migrando os processos, que gerou resultados ligeiramente melhores que o MOSIX isolado. O segundo teste envolveu processos com tempos de execução variados, onde o atraso do PVM foi de até 75%. O terceiro teste foi similar ao anterior, mas os processadores agora contavam com processos de fundo rodando, uma característica comum em ambientes de múltiplos usuários, e dessa vez o PVM chegou a mais de 60% de retardo. O A última avaliação envolveu o tempo de comunicação entre os processos executando, de novo o MOSIX obteve resultados superiores, às vezes chegando a ser mais de cinco vezes mais rápido.

Em (KOFAHI; ZAHRANI; HUSSAIN, 2006) ocorre um experimento similar ao anterior, mas desta vez comparando MOSIX e MPI. A execução de processos em sistemas ociosos mostrou que o MOSIX chega a ser na média de 70% mais rápido que o MPI quando executa acima de 6 processos, e na marca de 32, pode ser mais de 140% superior. O teste seguinte

avaliou a execução em ambientes com processos de fundo, resultando no MPI ser cerca de 50% mais lento que o MOSIX, atingindo até 165% quando executando 6 processos. O teste final envolveu um problema mais próximo da área de computação de alto desempenho, a multiplicação de matrizes. Enquanto em matrizes quadradas 500x500 o MOSIX executava entre 50% e 75% do tempo do MPI, usando matrizes 5000x200 o ganho não foi tão significativo, indo no máximo a 80% do tempo de execução do MPI.

Comparações entre diferentes implementações do padrão MPI foram apresentadas em (GRAHAM; WOODALL; SQUYRES, 2006). Foram comparadas as tecnologias OpenMPI, MPICH-GM, MPICH2 e MVAPICH. No experimento que verifica a latência para a realização de uma operação de ping-pong com mensagens de tamanho zero, os resultados foram variados, mas o OpenMPI levou vantagem em boa parte deles. Nos experimentos de verificação de largura de banda, a implementação OpenMPI obteve resultados consideravelmente melhores e mais estáveis.

Outro trabalho que apresentou resultados comparativos entre as implementações MPI foi (WOODALL et al., 2004). Nesse trabalho a implementação OpenMPI se destacou sobre as concorrentes, que no caso foram LAM7, MPICH2, LA-MPI e FT-MPI. Nos resultados de latência, os resultados de todas as implementações foram muito próximos, porém no experimento de largura de banda, o OpenMPI obteve resultados com melhorias de 30% com relação às outras abordagens.

3 Gerenciamento de Recursos

Nesta sessão serão apresentados 4 gerenciadores de recursos para clusters: *Oracle Grid Engine*, *LSF (Load Sharing Facility)*, *Torque*, *Condor*. São gerenciadores competitivos no mercado, eficientes com relação às principais funções de um gerenciador de recursos (escalonamento e políticas de alocação em sistemas distribuídos heterogêneos). Entretanto, cada um possui algumas peculiaridades, o que os fazem mais adequados ou não, dependendo do contexto a ser aplicado. A escolha de um gerenciador de recursos depende do tipo de projeto, do tipo de workflow e das necessidades do usuário.

3.1 Oracle Grid Engine

Até pouco tempo atrás conhecido como *Sun Grid Engine (SGE)*, o Oracle Grid Engine é um gerenciador de recursos distribuídos desenvolvido para distribuir as workloads dos usuários entre os recursos computacionais disponíveis (ORACLE, 2010c).

Os componentes principais do Oracle Grid Engine são os diferentes tipos de hosts, mas algumas outras entidades do sistema também se destacam, como pode ser visto na figura 3.1:

- *Qmaster:* componente central, o qual possui as principais funções administrativas, dentre as quais, aceitar jobs; associar os jobs aos recursos; escalonamento; monitorar o cluster. É executado em uma única máquina.
- *Shadow Master:* uma ou mais máquinas que podem substituir o a máquina que roda o Qmaster, caso haja alguma falha.
- *Execution Daemon:* componentes que recebem jobs do Qmaster e os executam utilizando recursos locais. Quando termina a execução de um job, o Qmaster é avisado, para que possa enviar mais um job. Também são enviadas notificações do estado do nó em um intervalo fixo de tempo. Caso o Qmaster não rebeba notificações de um determinado nó por algumas vezes consecutivas, este é retirado da lista de máquinas disponíveis. Não há

limites na quantidade de jobs que um deamon pode rodar. Mas geralmente esse número é limitado a quantidade de processadores da máquina em execução.

- *DRMAA:* interface de programação que permite às aplicações submeter, monitorar e controlar os jobs no cluster. Permite programação em C e Java.
- *Qmon:* interface gráfica para submissão, monitoramento e controle dos jobs e dos clusters. Jobs também podem ser enviados por linha de comando através do qsub e outros serviços (qsh, qlogin, etc.)
- *ARCo:* ferramenta web para acessar histórico de informações armazenadas do sistema, por padrão em SQL.

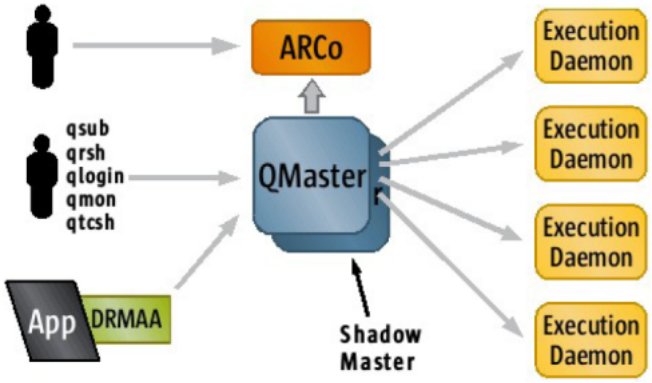


Figura 3.1: Componentes do GridEngine, de (ORACLE, 2010b)

Os usuários podem enviar seus jobs manualmente por interface gráfica ou linha de comando; ou por meio de um programa, utilizando a API DRMAA (Distributed Resource Management Application API). Cada job possui uma descrição do que deve executar, além de um conjunto de propriedades e definições que descrevem como deve ser executado. O SGE possui 4 tipos de job:

- *Batch:* possui uma sequência única de execução.
- *Paramétrico:* diferentes tarefas independentes entre si e que podem ser executados em paralelo. As tarefas são idênticas, o que muda é o conjunto de dados em que operam.
- *Paralelo:* conjunto de tarefas que podem ser dependentes e cooperar entre si. Fazem uso de ambientes paralelos como MPI, PVM.
- *Interativos:* permite que o usuário interaja diretamente com algum recurso disponível no cluster.

Os jobs enviados pelos usuários ficam em uma lista de pendentes no módulo Qmaster. Quando o escalonador é acionado, associa os jobs de maior prioridade aos recursos apropriados. O escalonador pode ser acionado de 3 formas:

- Periodicamente (o padrão de instalação é a cada 15 segundos);
- A cada job ou conjunto de jobs submetido, ou notificação de que um job ou um conjunto de jobs terminaram execução (liberaram recursos);
- Por linha de comando.

A prioridade do job é definida por uma série de fatores: usuário que submeteu; quais recursos necessita; deadline; quanto tempo está aguardando para executar; e outros. Para escalonar também são considerados os tipos de fila e a carga atual dos recursos. O esquema de filas do SGE é composto por uma série de instâncias de filas, uma por máquina, como pode ser visto na figura 3.2. Diferentes filas podem ser criadas para diferentes configurações dentro do cluster. A forma de escalonamento pode ser configurada. Há suporte para reserva antecipada de recursos, para acesso exclusivo a hosts, e controle de submissões.

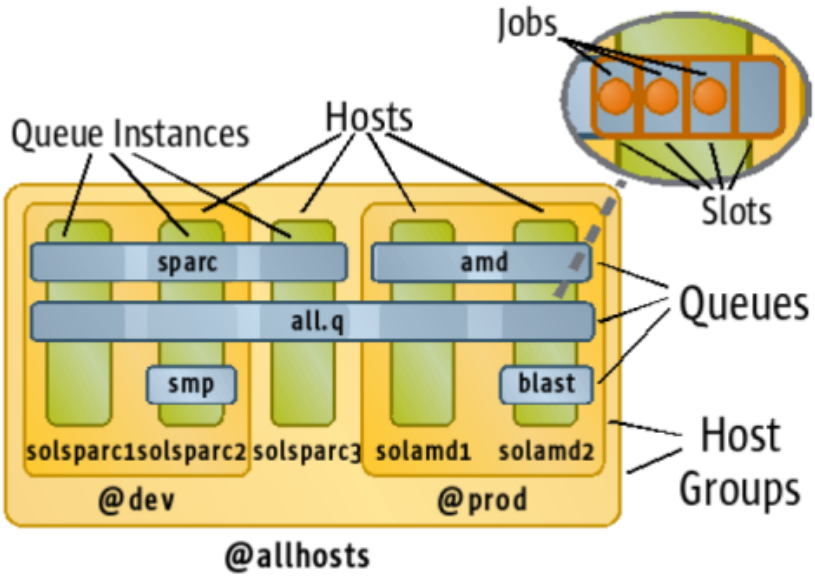


Figura 3.2: Exemplo de Filas e Hosts, de (ORACLE, 2010a)

Um fato recente importante, é que após o lançamento da versão mais recente do Oracle Grid Engine, houve um fork do projeto para o Open Grid Scheduler (SCHEDULER, 2010), pois as novidades da iteração não foram distribuídas à versão gratuita e de código aberto do sistema, chamado apenas de Grid Engine (ENGINE, 2010). Apesar de contar com a colaboração

da comunidade interessada, o Open Grid Scheduler ainda não conta como uma opção a ser analisada em aplicações práticas.

3.2 LSF - Load Sharing Facility

O LSF é um gerenciador de recursos que cria um sistema de imagem único em uma rede de computadores heterogêneos, para que todos os recursos possam ser manipulados e utilizados (PLATAFORM, 2010a)

O sistema base do LSF oferece os principais serviços para compartilhamento de carga em uma rede com diferentes sistemas computacionais, que podem ser vistos na figura 3.3:

- *LSLIB:* API do LSF para interface com o usuário.
- *LIM (Load Information Manager):* é executado em cada servidor LSF, monitorando a carga de cada máquina. Também envia essas informações ao LIM mestre.
- *LIM Mestre:* armazena informações de carga coletados dos LIMs que executam nas outras máquinas do cluster. Oferece essas informações aos usuários. Se um LIM mestre falha, é substituído pela próxima máquina com as configurações mais adequadas.
- *RES (Remote Execution Server):* é executado em cada servidor LSF, com a função de aceitar requisições remotas; oferecer transparência e segurança na execução de tarefas interativas; executa os jobs em background.
- *MBD (Master Batch Daemon):* recebe os jobs dos usuários e aplica políticas de escalonamento para associar estes jobs a recursos do cluster. Registra todas as transações efetuadas, além do ciclo de vida de cada job. Cada cluster do sistema possui um MBD, o qual é executado no nó mestre.
- *SBD (Slave Batch Daemon):* é executado em cada servidor, recebendo jobs escalonados pelo MBD. Executa localmente esses jobs, por meio do RES, controlando o estado do sistema local.
- *PIM (Process Information Manager):* é executado em cada servidor para monitorar cada job e processo criado localmente.

No LSF, os jobs enviados permanecem em fila até serem escalonados para execução em um host. Vários fatores influenciam no escalonamento de um job, entre estes:

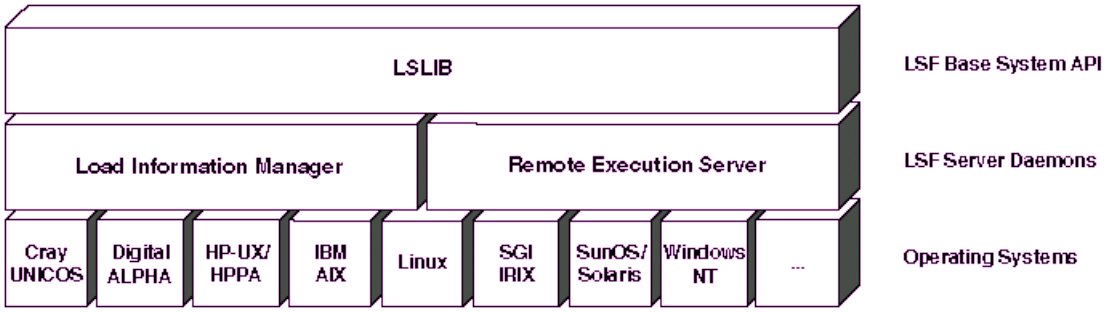


Figura 3.3: Arquitetura do LSF, de (PLATAFORM, 2010b)

- requisitos de recursos do job;
- disponibilidades destes recursos no cluster;
- limitações dos recursos para execução do job;
- dependências do job;
- restrições de compartilhamento;
- carga atual do recurso.

Os tipos de filas são controladas e configuradas pelo administrador, assim como os tipos e prioridades dos jobs. Se a fila não é especificada, o LSF escolhe uma fila mais adequada de acordo com configurações padrões que devem ser especificadas. Ou então de acordo com o cluster com mais recursos disponíveis.

Na questão de interface, usuários possuem acesso submissão, monitoração e controle dos jobs. Já administradores têm controle sobre os hosts e filas. Ainda possibilita descrição de recursos necessários para execução. Neste caso, os hosts adequados são escolhidos no cluster.

No caso de uma formação de multi-clusters, o LSF permite a distribuição dos workloads para execução em servidores ociosos localizados em qualquer lugar da organização, além de trabalhar com prioridades de recursos e tarefas.

3.3 Condor High-Throughpout Computing System

O Condor é uma ferramenta que que visa desenvolver, implementar, distribuir, e avaliar mecanismos e políticas que suportam a computação de alto desempenho em grandes coleções de recursos computacionais de propriedade compartilhada.

Quando o usuário submete seu job serial, ou paralelo ao Condor, este o coloca em uma fila, e posteriormente decide onde e como executá-lo, baseado em políticas de gerenciamento configuráveis. Há monitoramento minucioso de processos, o que permite o eficiente checkpointing.

Dentre as suas principais características estão:

1. Tolerância a falhas: faz checkpoints periódicos, permitindo que jobs migrem e evitando que se percam computações acumuladas caso haja falhas no sistema.
2. Fácil de usar: não é necessário programar para usar o condor, podendo executar jobs não interativos. O gerenciamento dos processos é automático e transparente aos usuários
3. Permite que jobs enviados em um primeiro workflow sejam enviado em um segundo workflow, sob diferentes políticas e condições.
4. Os jobs são ordenados de acordo com suas dependências.

O Condor se diferencia dos outros gerenciadores por ter um sistema de checkpoint muito eficiente. Além disso, é o gerenciador de sistemas mais antigo, e permite a formação de grids. Entretanto, possui algumas restrições, oferece um suporte limitado a programação paralela, e tem suporte técnico não tão eficaz como os outros gerenciadores, com pouca documentação.

Uma forma interessante de utilizar esta ferramenta é integrar com outros gerenciadores, pois permite *cycle-scavenge*. Ou seja, quando um segundo gerenciador escalona seus jobs em um processador que está sendo utilizado pelo Condor, ele faz checkpoint do seu job e o move para outra máquina. Desta forma, máquinas que estão reservadas por outros gerenciadores podem ser utilizadas enquanto ociosas.

3.4 Torque Resource Manager

TORQUE é um gerenciado de recursos open source que fornece controle sobre jobs em lote e nós de computadores distribuídos.

A arquitetura de um cluster TORQUE consiste em um nó mestre que roda o daemon pbs_server, que é o gerenciador de sistema de lotes do gerenciador, e diversos nós computacionais que rodam o daemon pbs_mon, que gerencia a execução dos jobs escalonados, estabelece limites para os recursos, e avisa o servidor que os jobs terminaram.

O nó mestre também roda um demon de escalonador, que interage com o pbs_server quanto a políticas locais para alocação de recursos e de nós para os jobs. A instalação básica do

TORQUE usa um escalonador FIFO e possuí ferramentas para que escalonadores mais complexos sejam implementados.

Os jobs são enviados ao pbs_server através do comano qsub. Quando o servidor os recebe, o escalonador é avisado, e quando encontra nós livres, ele envia uma lista com eles para o pbs_server para que os jobs possam ser executados.

Dentre as características do TORQUE, podem ser ressaltadas a preempção de jobs, permitindo que os usuários suspendam e reiniciem os jobs. Essa funcionalidade trabalha principalmente atrelada a características específicas da máquina, mas existe um pacote chamado BLCR, que quando instalado nos nós, torna o serviço independente de arquitetura. Com isso, podem ser criados checkpoints, que são imagens do estado atual da execução do job, de tempos em tempos, o que permite o reinício do mesmo se necessário.

O TORQUE possuí um tipo especial de job chamado service job, que ao invés de apenas executar ums script contendo as instruções do processo, responde a certos argumentos em linha de comando. Essa variante foi criada pensando em serviços de longa execução, e usa os mesmos argumentos que os scripts de serviços usuais de UNIX (start, stop, status).

Nas versões mais recentes, opções para rodar o TORQUE em modo de redundância ou de alta disponibilidade foram implementadas, ou seja, ele pode continuar rodando mesmo se o pbs_server cair, graças a existência de múltiplas cópias do servidor.

Quando ao monitoramento de recursos, o TORQUE foi projetado para analisar servidores em ambiente de lote. Ele não foi feito para gerenciar licença de software, redes, sistemas de arquivos, e assim por diante. Ele divide os atributos a serem monitorados em três categorias: configuração, lidando com configuração de hardware e atributos de lote especificados; utilização, ou seja, informação sobre a disponibilidade de recursos e de quem ou o que está os consumindo; e estado, que inclui status administrativo, informação geral sobre a "saúde"do nó, e status geral de uso.

Vale ressaltar que o TORQUE por si só não trabalha com migração de processos, essa função fica a cargo de outros programas como o *Moab Workload Manager* (RESOURCES, 2010b) ou o *Maui Cluster Scheduler* (RESOURCES, 2010a).

3.5 Meta-Escalonamento

3.5.1 CSF - Community Scheduler Framework

Os protocolos utilizados nos gerenciadores acima avaliados são muito específicos, e não são padrões, o que não permitiria uma maior heterogeneidade entre clusters. A maioria não integra recursos distribuídos sem um controle central. Ou seja, os clusters devem estar sob controle de um gerenciador específico suportado pelo gerenciador (FOSTER, 2002).

Um meta escalonador provê interfaces para acesso virtual aos recursos. Adota políticas globais para os provedores de recursos e consumidores. Desta forma, são definidas regras importantes em um ambiente de grid.

O CSF4 é um meta-escalonador *WSRF (Web Services Resource Framework)* que permite trabalhar com diferentes escalonadores locais. Desta forma, integra gerenciadores de clusters, entre os quais LSF, SGE, PBS e Condor. Ele mantém independência dos escalonadores locais, mas tem uma visão global dos recursos, permitindo uma melhor utilização dos mesmos. Além disso, está incluído no *Globus Toolkit GT4* e *GT2*. Seu funcionamento pode ser visto na figura 3.4.

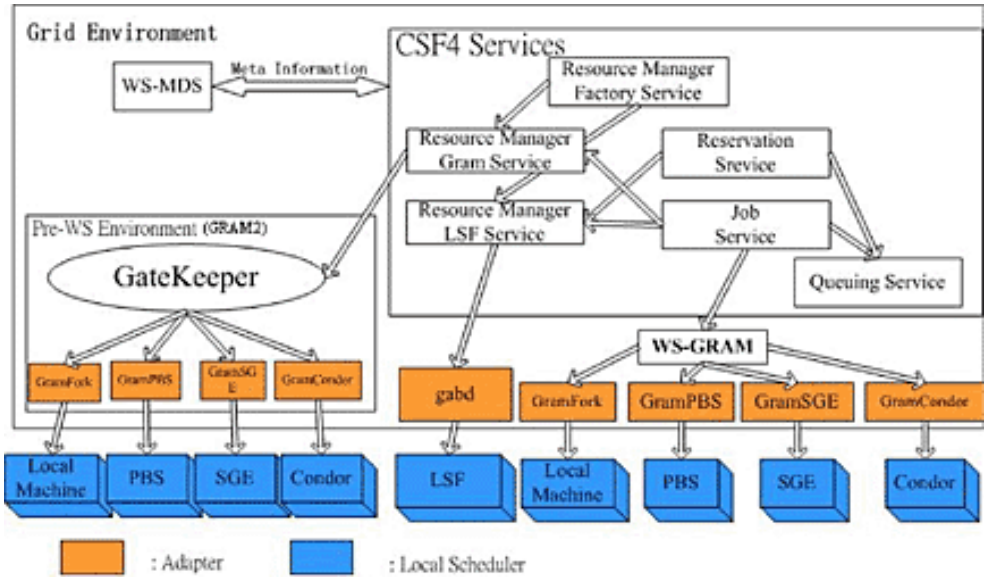


Figura 3.4: Arquitetura do CSF, de (ALLIANCE, 2010)

Utilizando um escalonador global como o CSF4 é possível submeter jobs tanto para clusters individuais como para múltiplos clusters. Ao mesmo tempo controlar a execução dos jobs nos diversos ambientes heterogêneos. Ou seja, em uma configuração onde haja clusters com diferentes gerenciadores, como o SGE, LSF, e até máquinas individuais, o CSF pode enviar

um job para um único gerenciador, acrescentando no comando para qual gerenciador está enviando. Ao mesmo tempo, também pode enviar integrar múltiplos clusters heterogêneos para processamento, utilizando o MPI, por exemplo.

4 Proposta

4.1 Descrição do Ambiente Proposto

Ambientes multi-clusters não são naturalmente utilizados, pois não existem ferramentas gerenciadoras de recursos e de escalonamento projetadas especificamente para eles, o que obriga os interessados em usar as disponíveis para clusters ou grids, com desempenho abaixo do esperado. Pesquisas nessa área não parecem evoluir devido à consolidação no uso de aplicações clássicas, como o Globus Toolkit, Condor, Plataform LSF ou Oracle GridEngine.

A proposta deste trabalho não inviabiliza o uso dessas soluções, pelo contrário, faz uso delas para gerenciar localmente cada cluster funcional do ambiente. Ela se dispõe a servir de interface entre o usuário e o sistema, abstraindo as complicações de uso de operações em linha de comando, ou necessidade de conhecimento e/ou treinamento específico com as ferramentas já existentes. Nossa solução também é simplificada pois é executada como uma aplicação Web, evitando problemas de compatibilidade durante a instalação e abstraindo a necessidade de se conhecer as configurações dos clusters que formarão o multi-cluster. Pode-se ter uma ideia do conceito observando a figura 4.1.

Quando devidamente instalada nossa ferramenta permitirá a escolha de um programa compilado localmente ou o uso de um pré-existente no cluster remoto, a passagem de um arquivo de entrada e/ou argumentos para a execução do programa, a escolha do sistema operacional, ambiente paralelo e o número processos usados por ele. Haverá também duas áreas de visualização, uma com o estado atual dos jobs submetidos pelo usuário, se já foram submetidos ao cluster, seu início, fim e disponibilização dos resultados. É a pretensão deste trabalho que as principais soluções de escalonamento e ambientes paralelos sejam suportados, porém, para fins de apresentação, foram feitas decisões do que seria implementado neste primeiro protótipo, e elas serão apresentadas na próxima seção.

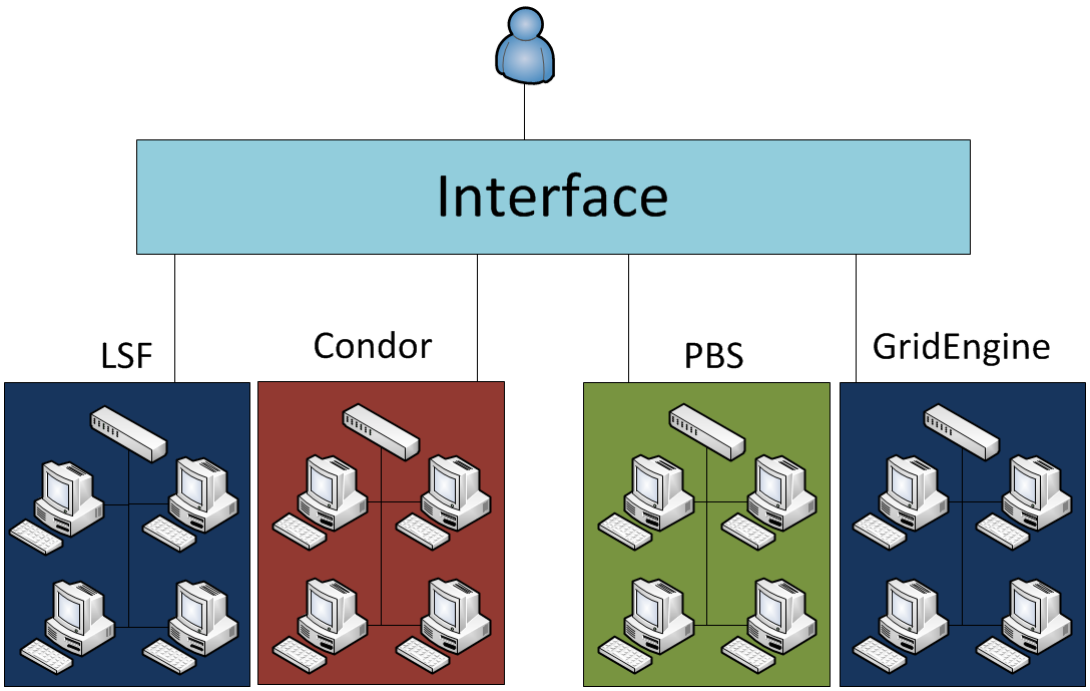


Figura 4.1: Proposta da Interface

4.2 Componentes

A configuraçao fisica do ambiente pode ser vista na tabela 4.1. Os dez computadores que rodam CentOS formam o *cluster Linux* e se conectam na rede através de um switch FastEthernet, já os dois nodos Windows formam o *cluster Windows* e se conectam diretamente à rede por suas placas de fábrica, por fim, um último computador será o hospedeiro da ferramenta e será chamado de *Portal*, sua configuração não importa, pois ele não realiza nenhuma computação de fato. Todo o sistema se encontra conectado à rede da Universidade Federal de Santa Catarina, e se localiza no Laboratório de Processamento em Sistemas Distribuídos no Departamento de Informática e Estatística.

	Linux	**Windows 1**	**Windows 2**
Processador	Intel Pentium III 733 MHz	Core2 Duo	Sempreon 2600+
Disco Rígido	15 GB	230 GB	40 GB
Mem. RAM	256 MB	2 GB	512 MB
S.O.	CentoOS 5.5	Windows XP SP2	Windows XP SP2

Tabela 4.1: Configuração do Ambiente

Como linguagem para a programação da ferramenta, foi escolhida o *PHP*, devido a ser uma linguagem voltada à Web, e a possibilidade de submissão de programas remotamente é um dos focos dela. O PHP também foi selecionado pois é independente do sistema operacional

onde executa, permitindo maior flexibilidade na instalação da ferramenta, característica também compartilhada pelo *Apache*, que foi escolhido como servidor web.

Na questão do ambiente paralelo, foi escolhido o MPI principalmente porque o PVM não recebe mais atualizações constantes, e por isso se encontra estagnado (mas ainda assim eficiente em algumas aplicações) enquanto o MPI está em constante atualização e aperfeiçoamento, dentre todas as implementações da biblioteca foi escolhido o OpenMPI por seu desempenho superior aos demais nas pesquisas feitas sobre o assunto. O MOSIX não foi selecionado devido a inexistência de uma versão completa para uso gratuito, apesar de licenças de teste e universitárias limitadas poderem ser adquiridas.

Dentre os gerenciadores de recursos foram escolhidos dois: o Oracle Grid Engine e o Condor. O Grid Engine foi escolhido por ter se mostrado uma opção interessante e em franca ascenção em um primeiro momento, entretanto, recentes decisões da Oracle talvez inviabilizem a continuidade no uso da aplicação, mesmo assim ele foi instalado no cluster Linux e não foi utlizado no outro, pois seu suporte a Windows é muito básico. O Condor está entre as principais aplicações para o compartilhamento de recursos, além de ser fácil instalação e configuração, por isso foi escolhido para o cluster Windows.

O LSF não foi seleconado como gerenciador por não existir qualquer forma de acesso livre a ele, o que vai diretamente contra a proposta. Já o Torque exigiria a adição das outras ferramentas que foram citadas que se fazem necessárias para o seu funcionamento como o esperado, tal acúmulo de softwares não era desejado.

4.3 Aspectos Relativos à Implementação

A ferramenta é composta de uma série de arquivos PHP cada qual responsável por uma função específica, uma figura apresentando a interface visual da aplicação pode ser vista no próximo capítulo:

- *portal:* É a raiz da ferramenta, validando uploads, parâmentros e seleções, além de permitir a visualização do estado atual da execução de cada job e do cluster que o executa;
- *portalglobalvars:* Reúne as variáveis de sistema como os diretórios raiz da ferramenta, Condor, GridEngine, etc.;
- *portalio:* Lida com a entrada/saída do sistema, incluindo a correta exibição do resultado e a opção de copiá-lo para o computador local;

- *portallogin e portalprotect:* Realizam a verificação de autenticidade do usuário, neste primeiro momento, os usuários são cadastrados direto no arquivo, caso algum dos escalonadores exija algum adicional de segurança, aqui que ele será implementado;

- *portalscriptgeneration:* É o responsável por gerar os scripts que serão submetidos gerenciadores de recurso, por isso deve ser editado com atenção para cada cluster;

Além destes arquivos, o sistema conta com arquivos de layout escritos em CSS, uma série de imagens para a visualização da execução e do resultado dos programas submetidos. Nos testes realizados, os arquivos que sofriam upload, e os arquivos de saída, eram todos carregados no Portal, mas essa configuração pode ser alterada se se quiser que ele sirva apenas para a submissão de processos.

Durante o desenvolvimento notou-se alguma dificuldade na questão de usuários e permissões, visto que em um primeiro momento a aplicação foi desenvolvida em Ubuntu, e o usuário executor de chamadas de shell no CentOs ser diferente. Mas o problema foi contornado alterando este usuário para o criado na instalação do sistema operacional. Outro contra-tempo que não era esperado, é que dentro do ambiente do PHP, muitas funções nativas do Linux como *ls*, por exemplo, não pode ser chamadas diretamente por seus aliases, sendo necessário seu caminho completo que foi embutido no arquivo de variáveis globais. Fora isso, o desenvolvimento da proposta foi bastante direto, e os principais problemas encontrados foram de HTML envolvendo a visualização, mas isso se encontra fora do escopo deste trabalho.

5 Resultados

Para a avaliação da ferramenta proposta, foi utilizado um programa simples de multiplicação de matrizes, sem a necessidade de arquivos de entrada ou passagem de parâmetros, mas matrizes do produto são geradas na execução. Para este trabalho, foram selecionados três estudos de caso: o primeiro em submissão para o cluster Windows, em seguida para o cluster Linux, e por fim, simultaneamente para ambos.

A figura 5.1 mostra a aplicação após as três execuções. Há a hora da submissão do programa do portal para o cluster remoto, o momento em que ele começa a ser executado, e a hora em que ele termina. A coluna *Estado* assume os valores *"Submetido"*, *"Executando"* e *"Concluído"*, mas também pode servir para indicar a ocorrência de algum erro e qual o tipo, dado que o cluster se comunique de forma adequada com o portal.

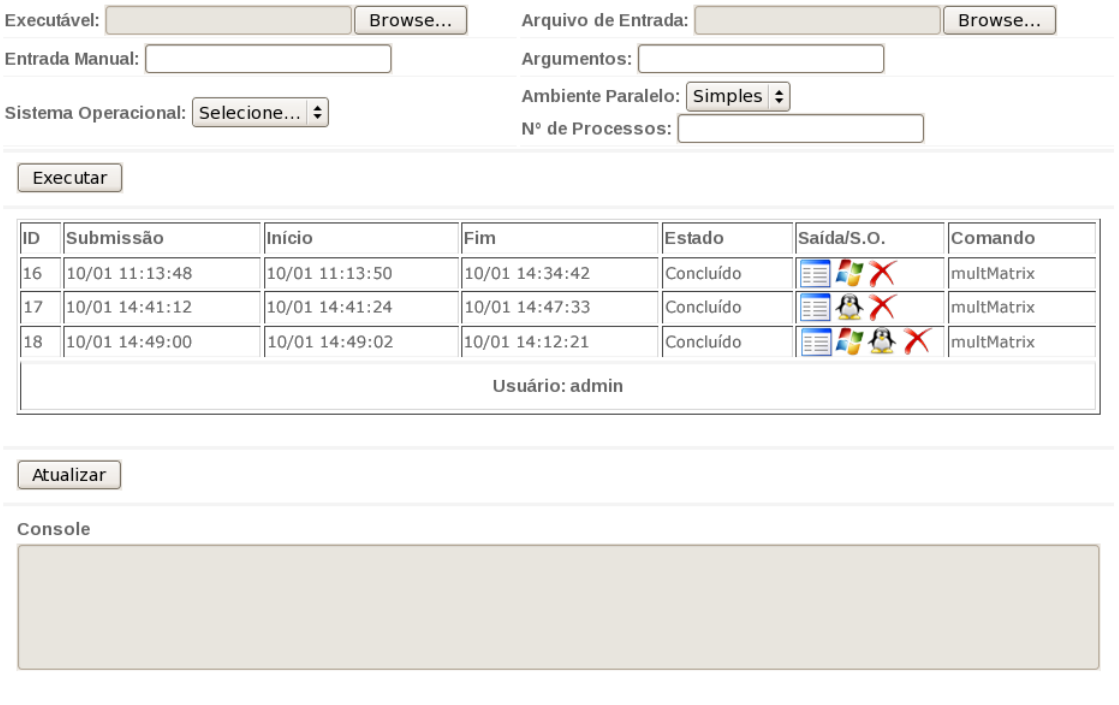


Figura 5.1: Resultado das Três Submissões

Os ícones adjacentes indicam um acesso ao arquivo de saída, o sistema operacional sob o qual o programa foi executado, e a opção de se apagar os arquivos relativos a uma execução (os scripts de submissão, arquivos transmitidos e a saída). Por fim, é indicado qual programa foi executado. A janela abaixo da tabela serve pra mostrar o estado do cluster durante a execução de um programa.

Não se avaliaram os tempos de execução *per se* do programa em cada cluster, pois este não é o escopo do trabalho, e o desempenho dos escalonadores individualmente já são conhecidos pela comunidade científica. O outro único tempo que pode ser medido é o de latência da rede, mas também não se enquadra como informação válida para este estudo, visto depende de cada caso, e pode ser atenuado por uma conexão dedicada.

O único resultado que de alguma forma é realmente válido para este trabalho, porém, não pode ser claramente visto. Com a visualização da interface gráfica, pode-se apenas perceber o estado da execução dos programas, e seu resultado final, o que pode levar a crer que não passa de um visualizador de sistema e arquivos. Porém, arquivos de configuração e de controle desses programas foram criados e replicados pelos clusters de forma consistente, e graças a eles que as execuções ocorreram dentro do esperado. Além disso, dentro do escopo de dois usuários, os arquivos pertinentes a cada um deles permaneceram isolados e inacessíveis, tanto quanto os usuários distintos, quanto pelos programas de cada um deles, uma garantia da integridade e segurança dos dados, mesmo que simples neste primeiro protótipo.

6 *Conclusões e Trabalhos Futuros*

Este trabalho se propôs a revisar o estado da arte das tecnologias de sistemas distribuídos, suas ferramentas de programação e aplicações de gerência de recursos. A partir desse conhecimento, também foi proposta e implementada uma ferramenta de meta-escalonamento de processos em ambientes multi-clusters, algo que até onde pode-se pesquisar, não existe.

A ferramenta pode ser considerada como uma alternativa interessante por possuir apenas cerca de 380 KB, sem contar as instalações de PHP e Apache que costumam ser padrão em quase todas as distribuições Linux. Além disso, conta com uma interface clara, e independente de plataforma, o sistema por si só também não necessita ser instalado em um computador muito potente, visto servirá basicamente de mensageiro entre os usuários e os clusters.

Os testes, apesar de simples, foram realizados com sucesso, retornando as informações úteis que se esperava, e a ferramenta exibe um potencial para continuar sofrendo melhorias para se tornar mais útil em seu escopo de funcionalidade. Com isso, chegamos a conclusão que os objetivos propostos foram atingidos satisfatoriamente.

Pode-se também observar que os objetivos propostos foram atingidos com sucesso. Graças ao estudo das tecnologias disponíveis, foi possível selecionar as mais adequadas ao nosso intento: o OpemMPI como ambiente paralelo simples, atual e eficiente; o Condor e o Oracle GridEngine como gerenciadores de recursos robustos e possuidores de tecnologias diferentes, o que permitiu comprovar nos testes a comunicação de dois clusters distintos.

É importante frisar que esta proposta se distingue do seu mais próximo "competidor"no mesmo nicho, o CSF, no ponto em que é bem mais enxuta e simples, contando com menos módulos necessários para seu funcionamento, porém, mantendo as funções básicas, o que também leva a uma instalação mais rápida. A simplicidade do protótipo pode porém enganar, ele não apenas envia arquivos e coleta informações como pode parecer a primeira vista, ele trata da comunicação entre sistemas diferentes que trabalham com tecnologias próprias para realizar suas funções, no caso deste trabalho ele começa de forma modesta, mas sua possibilidade de expansão é clara e ampla, dado que alguns dos itens levantados no tópico de trabalhos futuros,

logo abaixo, sejam cumpridos.

6.1 Trabalhos Futuros

Existem diversas características inerentes ao escalonamento de processos em ambientes distribuídos que não foram cobertos neste trabalho por se tratarem de tópicos mais avançados, mas que podem ser assumidos em trabalhos futuros relacionados:

- *Reserva Antecipada:* Algumas aplicações necessitam de garantia de tempo de execução e disponibilidade de recursos para fornecer uma certa qualidade de serviço;
- *Outros gerenciados de recursos:* LSF, Globus e PBS podem ser enquadrados como ferramentas a serem acopladas à nossa proposta;
- *Segurança:* Módulos de autenticação e login mais confiáveis e seguros.

Referências Bibliográficas

ALLIANCE, G. *CSF*. 2010. Acessado em: 03 set. 2010. Disponível em: <http://www.globus-.org/grid_software/computation/csf.php>.

BARAK, A. et al. Performance of PVM with the MOSIX preemptive process migration scheme. In: IEEE. *Computer Systems and Software Engineering, 1996., Proceedings of the Seventh Israeli Conference on*. [S.l.], 2002. p. 38–45.

BARAK, A.; LA'ADAN, O. The MOSIX multicomputer operating system for high performance cluster computing. *Future Generation Computer Systems*, Elsevier, v. 13, n. 4-5, p. 361–372, 1998.

BARRETO, M.; AVILA, R.; NAVAUX, P. The MultiCluster model to the integrated use of multiple workstation clusters. *Parallel and Distributed Processing*, Springer, p. 71–80, 2000.

BOINC. *BOINC*. 2010. Acessado em: 03 ago. 2010. Disponível em: <http://boinc.berkeley-.edu/>.

BORYCZKO, K. et al. Comparison of PVM and MPI performance in short-range molecular dynamics simulation. Citeseer.

DANTAS, M. *Computação Distribuída de Alto Desempenho: Redes, Clusters e Grids Computacionais*. [S.l.: s.n.], 2005.

ENGINE, G. *Grid Engine*. 2010. Acessado em: 15 set. 2010. Disponível em: <http:/-/gridengine.sunsource.net/>.

EUROPVM/MPI. *EUROPVM/MPI 2009 - Program*. 2010. Acessado em: 03 ago. 2010. Disponível em: <http://www.csc.fi/english/pages/pvmmpi09/program>.

FOSTER, I. What is the grid? a three point checklist. *GRID today*, v. 1, n. 6, p. 32–36, 2002.

GEIST, G.; KOHL, J.; PAPADOPOULOS, P. PVM and MPI: a Comparison of Features. *Calculateurs Paralleles*, Citeseer, v. 8, n. 2, p. 137–150, 1996.

GRAHAM, R.; WOODALL, T.; SQUYRES, J. Open MPI: A flexible high performance MPI. *Parallel Processing and Applied Mathematics*, Springer, p. 228–239, 2006.

INTEL. *Intel® MPI Library - Intel® Software Network*. 2010. Acessado em 04 de ago. 2010. Disponível em: <http://software.intel.com/en-us/intel-mpi-library/>.

JANSON, D. et al. Dynamic Resource Matching for Multi-clusters Based on an Ontology-Fuzzy Approach. In: SPRINGER. *High Performance Computing Systems and Applications*. [S.l.], 2010. p. 241–250.

KITOWSKI, J.; BORYCZKO, K.; MOŚCIŃSKI, J. Comparison of two short-range molecular dynamics algorithms for parallel computing. *Applied Parallel Computing Industrial Computation and Optimization*, Springer, p. 443–449, 1996.

KOFAHI, N.; ZAHRANI, S. A.; HUSSAIN, S. MOSIX Evaluation on a Linux Cluster. *Int. Arab J. Inf. Technol*, Citeseer, v. 3, n. 1, p. 62–68, 2006.

LABORATORY, A. N. *MPICH2 - Performance and Portability*. [S.l.], 2007.

MPI. *Message Passing Interface*. 2010. Acessado em: 04 ago. 2010. Disponível em: <http://www.mcs.anl.gov/research/projects/mpi/>.

MPICH2. *MPICH2 : about MPICH2*. 2010. Acessado em 04 de ago. 2010. Disponível em: <http://www.mcs.anl.gov/research/projects/mpich2/about/index.php?s=about>.

MPICH2-MX. *MPICH-MX and MPICH2-MX Software*. 2010. Acessado em 04 de ago. 2010. Disponível em: <http://www.myri.com/scs/download-mpichmx.html>.

MPICH2A. *Timeline - mpich2*. 2010. Acessado em 04 de ago. 2010. Disponível em: <http://trac.mcs.anl.gov/projects/mpich2% -/timeline>.

MVAPICH. *MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE*. 2010. Acessado em 04 de ago. 2010. Disponível em: <http://mvapich.cse.ohio-state.edu/>.

OPENMOSIX. *Transferência em um cluster openMosix*. 2010. Acessado em: 06 jun. 2010. Disponível em: <http://en.wikipedia.org/wiki/Openmosix>.

OPENMPI. *Open MPI: Open Source High Performance Computing*. 2010. Acessado em 10 de ago. 2010. Disponível em: <http://www.open-mpi.org/>.

OPENMPIA. *OPEN MPI: Version Timeline*. 2010. Acessado em 10 de ago. 2010. Disponível em: <http://www.open-mpi.org/software/ompi/versions/timeline.php>.

OPENMPIB. *Timeline - Open MPI - trac*. 2010. Acessado em 10 de ago. 2010. Disponível em: <https://svn.open-mpi.org/trac/ompi/timeline>.

OPENMPIC. *Open MPI Mailing Lists*. 2010. Acessado em 10 de ago. 2010. Disponível em: <http://www.open-mpi.org/community/lists/ompi.php>.

ORACLE. *Configuring Queues*. 2010. Acessado em: 15 set. 2010. Disponível em: <http://wikis.sun.com/display/gridengine62u6/How+the+System+Operates>.

ORACLE. *How the System Operates*. 2010. Acessado em: 15 set. 2010. Disponível em: <http://wikis.sun.com/display/gridengine62u6/How+the+System+Operates>.

ORACLE. *Oracle Grid Engine*. 2010. Acessado em: 15 set. 2010. Disponível em: <http://www.oracle.com/us/products/tools% -/oracle-grid-engine-075549.html>.

PLATAFORM. *Load Sharing Facility*. 2010. Acessado em: 03 set. 2010. Disponível em: <http://www.platform.com/workload-management% -/high-performance-computing>.

PLATAFORM. *Platform LSF® Programmer's Guide*. 2010. Acessado em: 03 set. 2010. Disponível em: <http://ams.cern.ch/AMS/7/programmer/index.html>.

PROJECT, C. *Charlotte Project*. 2010. Acessado em: 20 set. 2010. Disponível em: <http://cs.nyu.edu/milan/charlotte/frmain.html>.

PVM. *PVM: Parallel Virtual Machine*. 2010. Acessado em: 03 ago. 2010. Disponível em: <http://www.csm.ornl.gov/pvm/>.

RESOURCES, C. *Maui Cluster Scheduler*. 2010. Acessado em: 15 set. 2010. Disponível em: <http://www.clusterresources.com/products/maui-cluster-scheduler.php>.

RESOURCES, C. *Moab Workload Manager*. 2010. Acessado em: 15 set. 2010. Disponível em: <http://www.clusterresources.com/products/moab-cluster-suite/workload-manager.php>.

SCHEDULER, O. G. *Open Grid Scheduler*. 2010. Acessado em: 01 out. 2010. Disponível em: <http://gridscheduler.sourceforge.net/>.

SQUYRES, J.; LUMSDAINE, A. A component architecture for LAM/MPI. *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Springer, p. 379–387, 2003.

STRUMPEN, V. *The Network Machine*. Tese (Doutorado) — SWISS FEDERAL INSTITUTE OF TECHNOLOGY, 1995.

SUNDERAM, V. PVM: A framework for parallel distributed computing. *Concurrency: practice and experience*, Wiley Online Library, v. 2, n. 4, p. 315–339, 1990.

TOP500. *June 2010 | TOP500 Supercomputing Sites*. 2010. Acessado em: 28 ago. 2010. Disponível em: <http://www.top500.org/lists/2010/06>.

WERSTEIN, P.; PETHICK, M.; HUANG, Z. A performance comparison of DSM, PVM, and MPI. In: IEEE. *Parallel and Distributed Computing, Applications and Technologies, 2003. PDCAT'2003. Proceedings of the Fourth International Conference on*. [S.l.], 2003. p. 476–482.

WOODALL, T. et al. TEG: A high-performance, scalable, multi-network point-to-point communications methodology. *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Springer, p. 307–352, 2004.

Artigo

Proposta de Middleware Para Meta-Escalonamento em Ambientes Multi-Cluster

Gabriel M. Pessoa[1], M. A. R. Dantas[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88.040-900 – Florianópolis – SC – Brasil
{gabrielm, mario}@inf.ufsc.br

Abstract. *The area of high performance computing continuously search for alternatives that integrates the various options of computational tools and physical architectures used, and better utilize the sum of its capabilities, this is a complex task because many of the applications operate on heterogeneous systems that are sometimes very different in their individual composition. This paper aims to present the first prototype of a open source, and user-friendly free alternative tool for meta-scheduling for multi-cluster computing environments. The concepts behind this idea will be presented, then its proposal of implementation, its results and a viability study.*

Resumo. *A área da computação de alto desempenho busca continuamente por alternativas que integrem as diversas opções de ferramentas computacionais e arquiteturas físicas que utiliza, e melhor aproveitem a soma de seus recursos, esta é uma tarefa complexa, pois muitas das aplicações nesse ramo atuam sobre sistemas heterogêneos que às vezes são muito diferentes em sua composição individual. Este trabalho tem como objetivo apresentar o primeiro protótipo de uma alternativa gratuita, de código aberto, e amigável ao usuário de ferramenta de meta-escalonamento para ambientes de multi-clusters computacionais. Serão apresentados os conceitos por trás dessa ideia, sua proposta de implementação, seus resultados e um estudo de sua viabilidade.*

1.Introdução

Sistemas computacionais de alto desempenho tem sido frequentemente utilizados como ferramentas para viabilizar a realização de pesquisas nas mais diversas áreas de conhecimento. Muitas vezes, o avanço das pesquisas nessas áreas é dependente da existência de sistemas computacionais que sejam capazes de resolver determinadas tarefas em tempo hábil. Frequentemente, a capacidade computacional dos computadores convencionais não é suficiente para a realização de tarefas necessárias, seja para a realização de cálculos excessivamente complexos ou para a realização de processamento de grandes quantidades de dados, necessários em áreas de pesquisa

como astronomia, meteorologia, simulações científicas, dentre outras. Em muitos casos, supercomputadores são utilizados de modo a suprir tal demanda. Porém, tais sistemas apresentam um custo muito elevado, tornando-se inviáveis para muitas instituições de pesquisa e empresas. Uma dentre as soluções mais utilizadas atualmente são os clusters computacionais, que são conjuntos de computadores convencionais interconectados por redes de alto desempenho de modo a obter capacidades de processamento semelhantes aos supercomputadores. Assim, através da união de componentes baratos é possível oferecer sistemas de alto poder de processamento por um baixo custo.

2.Sistemas Distribuídos

2.1.Computação Distribuída

O paradigma de computação distribuída é aquele no qual os componentes responsáveis por uma aplicação não se encontram necessariamente próximos. Hardware, pacotes de software, e até sensores podem estar em continentes diferentes e os resultados são obtidos em tempo e precisão razoáveis e com um custo relativamente baixo às partes envolvidas.

A ideia de agregar poder computacional não é nova e já existe desde que os primeiros computadores foram ligados via rede, e com os anos sua utilização sofreu refinamentos, seja nas possibilidades de configuração, seja na forma como seus recursos são utilizados. O conceito por trás da computação distribuída é de transformar agrupamentos de máquinas, normalmente de baixo e médio custo, em potentes centros computacionais, os quais podem resolver problemas que envolvam milhares de aplicações simultâneas, ou o processamento paralelo de tarefas complexas. A essa área específica se dá o nome de computação distribuída de alto desempenho.

2.2.Meta-Computador

Ao ambiente que transmite essa sensação de único, dá-se o nome de meta-computador. Na definição mais ampla possível para computação distribuída, a união e disponibilização de recursos de computadores através da Internet é um meta-computador, o mais rápido do mundo, por sinal [PROJECT 2010]. Seguindo o conceito de uso da Web como um meta-computador, sempre se esbarrou na dificuldade de particionamento do problema, disponibilidade, heterogenia e segurança dos sistemas executores. Por isso, apesar de existirem iniciativas como o Projeto Charlotte, o foco inicial da computação distribuída foram os clusters e grids.

2.3.Cluster Computacional

O cluster computacional é uma das configurações mais utilizadas na computação distribuída de alto desempenho, podendo ser definido como um agrupamento físico, local ou virtual, de inúmeros computadores, chamados aqui de nós ou nodos, com o objetivo de agregar recursos computacionais para disponibilizá-los para a melhoria de aplicações [DANTAS 2005]. Fruto direto das primeiras pesquisas na área, até hoje muitos clusters são basicamente computadores pessoais conectados entre si e que compartilham CPU, disco rígido e RAM para fins de processamento. Eventualmente, empresas como Cray e NEC começaram a manufaturar sistemas especializados que

utilizam o conceito de clusterização, mas com o diferencial do renome e do know-hall.

2.4.Grid Computacional

Uma evolução natural dos clusters foram as grids computacionais. Quando as instituições começaram a pesquisar formas de estender o limite geográfico dos clusters, que até o momento atingiam as fronteiras das próprias instituições, perceberam que da mesma forma que alguns computadores podiam ser conectados para compartilhar recursos, os agregados podiam ser ligados uns aos outros, criando nodos em uma grande malha que poderia ter alcance e escalabilidade muito superiors aos dos agregados. Uma grid pode ser vista como um ambiente de componentes heterogêneos e geograficamente dispersos e com uma quantidade de recursos e serviços superior a de outras configurações distribuídas. Isso torna a questão da consistência do sistema ainda mais importante, dado que não existe um nó mestre coordenando a submissão, escalonamento e verificação dos processos.

2.5.Multi-Cluster

Um subconjunto interessante dos ambientes de grid é o chamado multi-cluster, um agrupamento de clusters de diferentes domínios para a execução de processos. A grid convencional se distingue do multi-cluster por geralmente serem altamente heterogêneas, distribuídas entre várias organizações e composta por diversos tipos de sistemas operacionais, serviços, hardware e dispositivos de variados provedores. Considera-se também que cada cluster componente trabalhará com sua própria fila de submissão, arquivos de registro e unidades funcionais, ou seja, as instruções executadas em um cluster foram submetidas pelo nodo mestre dele, e apenas elas podem ler/escrever fisicamente nele. Entretanto, as instruções são obtidas de uma cache única e globalmente acessível.

2.6.Parallel Virtual Machine

Desenvolvido pela Universidade do Tennessee, Laboratório Nacional de Oak Ridge e pela Universidade Emory, o PVM [PVM 2010] é um conjunto de ferramentas integradas que oferece solução para computação concorrente heterogênea. Através da utilização do PVM, é possível utilizar um conjunto heterogêneo de computadores de modo que estes cooperem entre si em prol da solução de uma determinada tarefa, compartilhando recursos computacionais através de uma rede [SUNDERAM 1990]. Em geral, é utilizado para paralelização de aplicações que demandam tempos de execução muito altos quando se utiliza uma abordagem sequencial. O nome, Parallel Virtual Machine, se deve ao fato de o PVM utilizar uma abordagem na qual uma coleção de máquinas heterogêneas formam uma única máquina paralela virtual, de modo que seus usuários podem interagir com ela como se estivessem lidando com um computador paralelo único.

2.7.Message Passing Interface

Diferentemente do PVM, que é um conjunto de ferramentas para computação paralela, construído e mantido por universidades e laboratórios de pesquisa, MPI [MPI 2010] é a especificação formal de uma API padrão para desenvolvimento de aplicações paralelas

através do modelo de passagem de mensagens. MPI é um padrão de facto para o desenvolvimento de aplicações paralelas utilizando o modelo de passagem de mensagens [SQUYRES; LUMSDAINE 2003] para execução em ambientes como clusters, onde os recursos computacionais são interligados por uma rede, seguindo uma abordagem de memória distribuída. Sendo apenas uma especificação, MPI se torna independente de linguagem, possuindo implementações e bindings para várias linguagens de programação. Devido a independência de linguagem, existem diversas implementações do padrão MPI, desenvolvidas por diferentes fabricantes/pesquisadores, como por exemplo, MPICH, LAM/MPI e a mais recente, OpenMPI, sendo esta uma evolução da implementação LAM/MPI.

2.1.MOSIX

MOSIX é uma melhoria do BSD/OS que faz uso de algoritmos para compartilhamento de recursos de forma adaptativa, sendo implementado em camada de software e que permite que aplicações rodem em nodos remotos como se estivessem executando localmente [BARAK; LA'ADAN, 1998]. A versão mais recente do MOSIX permite a estruturação de clusters, multi-clusters e ainda traz a interessante característica da possibilidade de ser montado um cluster de GPUs, tirando proveito da potência computacional matemática desses processadores.

3.Gerenciamento de Recursos

2.1.Oracle Grid Engine

Até pouco tempo atrás conhecido como Sun Grid Engine (SGE), o Oracle Grid Engine é um gerenciador de recursos distribuídos desenvolvido para distribuir as workloads dos usuários entre os recursos computacionais disponíveis [ORACLE 2010]. Os componentes principais do Oracle Grid Engine são os diferentes tipos de hosts, mas

algumas outras entidades do sistema também se destacam.

- Qmaster: componente central, o qual possui as principais funções administrativas, dentre as quais, aceitar jobs; associar os jobs aos recursos; escalonamento; monitorar o cluster. É executado em uma única máquina.
- Shadow Master: uma ou mais máquinas que podem substituir o a máquina que roda o Qmaster, caso haja alguma falha.
- Execution Daemon: componentes que recebem jobs do Qmaster e os executam utilizando recursos locais. Quando termina a execução de um job, o Qmaster é avisado, para que possa enviar mais um job. Também são enviadas notificações do estado do nó em um intervalo fixo de tempo. Caso o Qmaster não rebeba notificações de um determinado nó por algumas vezes consecutivas, este é retirado da lista de máquinas disponíveis. Não há limites na quantidade de jobs que um deamon pode rodar. Mas geralmente esse número é limitado a quantidade de processadores da máquina em execução.
- DRMAA: interface de programação que permite às aplicações submeter, monitorar e controlar os jobs no cluster. Permite programação em C e Java.

- Qmon: interface gráfica para submissão, monitoramento e controle dos jobs e dos clusters. Jobs também podem ser enviados por linha de comando através do qsub e outros serviços (qsh, qlogin, etc.)
- ARCo: ferramenta web para acessar histórico de informações armazenadas do sistema, por padrão em SQL.

2.1.Load Sharing Facility

O LSF é um gerenciador de recursos que cria um sistema de imagem único em uma rede de computadores heterogêneos, para que todos os recursos possam ser manipulados e utilizados [PLATAFORM 2010]. O sistema base do LSF oferece os principais serviços para compartilhamento de carga em uma rede com diferentes sistemas computacionais. No LSF, os jobs enviados permanecem em fila até serem escalonados para execução em um host. Os tipos de filas são controladas e configuradas pelo administrador, assim como os tipos e prioridades dos jobs.

2.1.Condor

O Condor é uma ferramenta que que visa desenvolver, implementar, distribuir, e avaliar mecanismos e políticas que suportam a computação de alto desempenho em grandes coleções de recursos computacionais de propriedade compartilhada. Quando o usuário submete seu job serial, ou paralelo ao Condor, este o coloca em uma fila, e posteriormente decide onde e como executá-lo, baseado em políticas de gerenciamento configuráveis. Há monitoramento minucioso de processos, o que permite o eficiente checkpointing.

O Condor se diferencia dos outros gerenciadores por ter um sistema de checkpoint muito eficiente. Além disso, é o gerenciador de sistemas mais antigo, e permite a formação de grids. Entretanto, possui algumas restrições, oferece um suporte limitado a programação paralela, e tem suporte técnico não tão eficaz como os outros gerenciadores, com pouca documentação.

2.1.Torque Resource Manager

TORQUE é um gerenciado de recursos open source que fornece controle sobre jobs em lote e nós de computadores distribuídos. A arquitetura de um cluster TORQUE consiste em um nó mestre que roda o daemon pbs_server, que é o gerenciador de sistema de lotes do gerenciador, e diversos nós computacionais que rodam o daemon pbs_mon, que gerencia a execução dos jobs escalonados, estabelece limites para os recursos, e avisa o servidor que os jobs terminaram. O nó mestre também roda um demon de escalonador, que interage com o pbs_server quanto a políticas locais para alocação de recursos e de nós para os jobs. A instalação básica do TORQUE usa um escalonador FIFO e possuí ferramentas para que escalonadores mais complexos sejam implementados.

2.1.Commuinity Scheduler Framework

O CSF4 é um meta-escalonador WSRF (Web Services Resource Framework) que permite trabalhar com diferentes escalonadores locais. Desta forma, integra gerenciadores de clusters, entre os quais LSF, SGE, PBS e Condor. Ele mantém

independência dos escalonadores locais, mas tem uma visão global dos recursos, permitindo uma melhor utilização dos mesmos. Além disso, está incluído no Globus Toolkit GT4 e GT2.

Utilizando um escalonador global como o CSF4 é possível submeter jobs tanto para clusters individuais como para múltiplos clusters. Ao mesmo tempo controlar a execução dos jobs nos diversos ambientes heterogêneos. Ou seja, em uma configuração onde haja clusters com diferentes gerenciadores, como o SGE, LSF, e até máquinas individuais, o CSF pode enviar um job para um único gerenciador, acrescentando no comando para qual gerenciador está enviando. Ao mesmo tempo, também pode enviar integrar múltiplos clusters heterogêneos para processamento, utilizando o MPI, por exemplo.

4.Proposta

Ambientes multi-clusters não são naturalmente utilizados, pois não existem ferramentas gerenciadoras de recursos e de escalonamento projetadas especificamente para eles, o que obriga os interessados em usar as disponíveis para clusters ou grids, com desempenho abaixo do esperado. Pesquisas nessa área não parecem evoluir devido consolidação no uso de aplicações clássicas, como o Globus Toolkit, Condor, Plataform LSF ou Oracle GridEngine.

A proposta deste trabalho não inviabiliza o uso dessas soluções, pelo contrário, faz uso delas para gerenciar localmente cada cluster funcional do ambiente. Ela se dispõe a servir de interface entre o usuário e o sistema, abstraindo as complicações de uso de operações em linha de comando, ou necessidade de conhecimento e/ou treinamento específico com as ferramentas já existentes. Nossa solução também é simplificada pois é executada como uma aplicação Web, evitando problemas de compatibilidade durante a instalação e abstraindo a necessidade de se conhecer as configurações dos clusters que formarão o multi-cluster. Pode-se ter uma ideia do conceito observando a Figura 1.

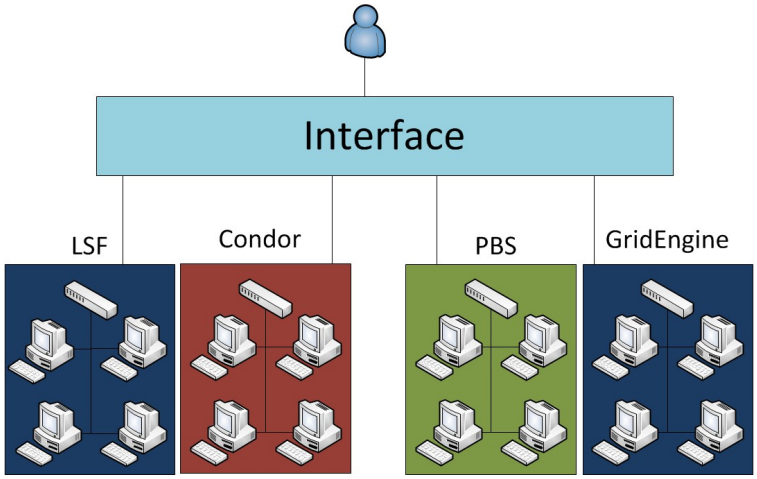


Figura 1. Proposta da Ferramenta

A configuração física do ambiente envolveu dez computadores que rodam CentOS formando o cluster Linux e que se conectam na rede através de um switch FastEthernet, são máquinas de baixos recursos, processadores Pentium III 733, 15 GB de HD e 256 MB de RAM. Dois nodos Windows formando o cluster Windows e se conectam

diretamente à rede por suas placas de fábrica, um Core2Duo com 230 GB de HD e 2 GB de Ram, e um Sempron 2600+ de 40 GB de HD e 512 MB de RAM, por fim, um último computador será o hospedeiro da ferramenta e será chamado de Portal, sua configuração não importa, pois ele não realiza nenhuma computação de fato. Todo o sistema se encontra conectado à rede da Universidade Federal de Santa Catarina, e se localiza no Laboratório de Processamento em Sistemas Distribuídos no Departamento de Informática e Estatística.

Como linguagem para a programação da ferramenta, foi escolhida o PHP, devido a ser uma linguagem voltada à Web, e a possibilidade de submissão de programas remotamente é um dos focos dela. O PHP também foi selecionado pois é independente do sistema operacional onde executa, permitindo maior flexibilidade na instalação da ferramenta, característica também compartilhada pelo Apache, que foi escolhido como servidor web. Como ambiente paralelo foi selecionado o OpenMPI por sua alta disponibilidade quanto a arquiteturas e sistemas operacionais suportados. Já como gerenciadores de recursos, os selecionados foram o Oracle Grid Engine e o Condor, o primeiro por sua promissora evolução em sistemas Linux, e o segundo por sua facilidade de instalação em máquinas Windows. Além disso, ambos foram escolhidos por serem ferramentas conhecidas e robustas.

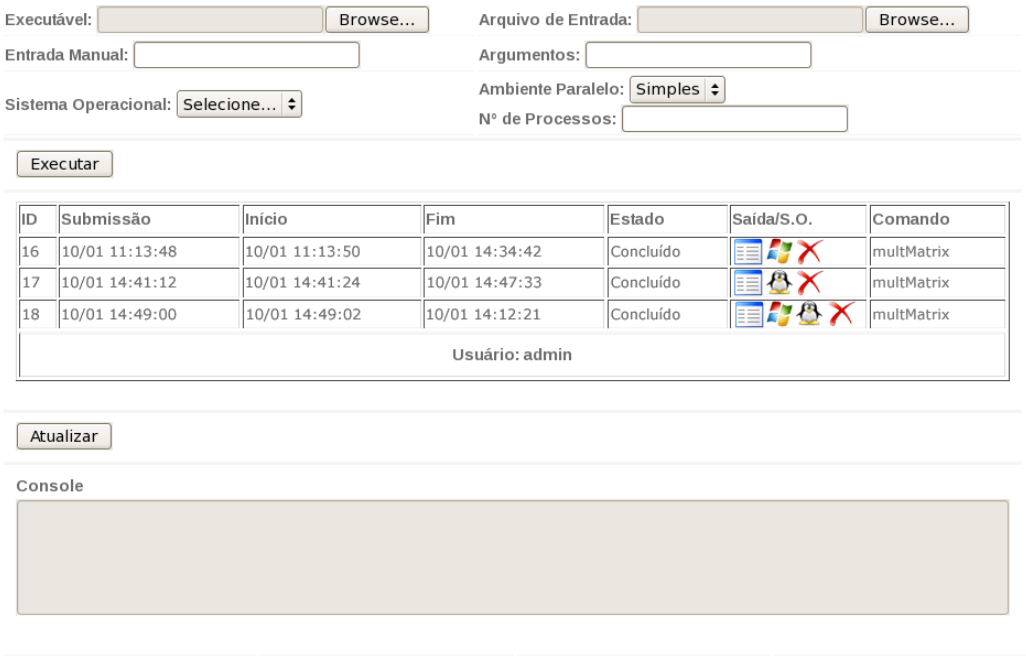
A ferramenta é composta de uma série de arquivos PHP cada qual responsável por uma função específica, uma figura apresentando a interface visual da aplicação pode ser vista na próxima seção. Além destes arquivos, o sistema conta com arquivos de layout escritos em CSS, uma série de imagens para a visualização da execução e do resultado dos programas submetidos. Nos testes realizados, os arquivos que sofriam upload, e os arquivos de saída, eram todos carregados no Portal, mas essa configuração pode ser alterada se se quiser que ele sirva apenas para a submissão de processos.

5.Resultados

Para a avaliação da ferramenta proposta, foi utilizado um programa simples de multiplicação de matrizes, sem a necessidade de arquivos de entrada ou passagem de parâmetros, mas matrizes do produto são geradas na execução. Para este trabalho, foram selecionados três estudos de caso: o primeiro em submissão para o cluster Windows, em seguida para o cluster Linux, e por fim, simultaneamente para ambos. A Figura 2 mostra a aplicação após as três execuções. Há a hora da submissão do programa do portal para o cluster remoto, o momento em que ele começa a ser executado, e a hora em que ele termina. A coluna Estado assume os valores "Submetido", "Executando" e "Concluído", mas também pode servir para indicar a ocorrência de algum erro e qual o tipo, dado que o cluster se comunique de forma adequada com o portal.

Os ícones adjacentes indicam um acesso ao arquivo de saída, o sistema operacional sob o qual o programa foi executado, e a opção de se apagar os arquivos relativos a uma execução (os scripts de submissão, arquivos transmitidos e a saída). Por fim, é indicado qual programa foi executado. A janela abaixo da tabela serve pra mostrar o estado do cluster durante a execução de um programa.

Figura 2. Resultado Após as Três Submissões



O único resultado que de alguma forma é realmente válido para este trabalho, porém, não pode ser claramente visto. Com a visualização da interface gráfica, pode-se apenas perceber o estado da execução dos programas, e seu resultado final, o que pode levar a crer que não passa de um visualizador de sistema e arquivos. Porém, arquivos de configuração e de controle desses programas foram criados e replicados pelos clusters de forma consistente, e graças a eles que as execuções ocorreram dentro do esperado. Além disso, dentro do escopo de dois usuários, os arquivos pertinentes a cada um deles permaneceram isolados e inacessíveis, tanto quanto os usuários distintos, quanto pelos programas de cada um deles, uma garantia da integridade e segurança dos dados, mesmo que simples neste primeiro protótipo.

6.Conclusões e Trabalhos Futuros

Este trabalho se propôs a revisar o estado da arte das tecnologias de sistemas distribuídos, suas ferramentas de programação e aplicações de gerência de recursos. A partir desse conhecimento, também foi proposta e implementada uma ferramenta de meta-escalonamento de processos em ambientes multi-clusters, algo que até onde pode-se pesquisar, não existe. A ferramenta pode ser considerada como uma alternativa interessante por possuir apenas cerca de 380 KB, sem contar as instalações de PHP e Apache que costumam ser padrão em quase todas as distribuições Linux. Além disso, conta com uma interface clara, e independente de plataforma, o sistema por si só também não necessita ser instalado em um computador muito potente, visto servirá basicamente de mensageiro entre os usuários e os clusters.

Os testes, apesar de simples, foram realizados com sucesso, retornando as informações úteis que se esperava, e a ferramenta exibe um potencial para continuar sofrendo melhorias para se tornar mais útil em seu escopo de funcionalidade. Com isso, chegamos a conclusão que os objetivos propostos foram atingidos satisfatoriamente.

Existem diversas características inerentes ao escalonamento de processos em ambientes

distribuídos que não foram cobertos neste trabalho por se tratarem de tópicos mais avançados, mas que podem ser assumidos em trabalhos futuros relacionados:

- Reserva Antecipada: Algumas aplicações necessitam de garantia de tempo de execução e disponibilidade de recursos para fornecer uma certa qualidade de serviço;
- Outros gerenciados de recursos: LSF, Globus e PBS podem ser enquadrados como ferramentas a serem acopladas à nossa proposta;
- Segurança: Módulos de autenticação e login mais confiáveis e seguros.

References

BARAK, A.; LA'ADAN, O. The MOSIX multicomputer operating system for high performance cluster computing. Future Generation Computer Systems, Elsevier, v. 13, n. 4-5, p. 361–372, 1998.

DANTAS, M. Computação Distribuída de Alto Desempenho: Redes, Clusters e Grids Computacionais. [S.l.: s.n.], 2005.

MPI. Message Passing Interface. 2010. Acessado em: 04 ago. 2010. Disponível em: <http://www.mcs.anl.gov/research/projects/mpi/>.

ORACLE. Oracle Grid Engine. 2010. Acessado em: 15 set. 2010. Disponível em: <http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html>.

PLATAFORM. Load Sharing Facility. 2010. Acessado em: 03 set. 2010. Disponível em: <http://www.platform.com/workload-management/high-performance-computing>.

PROJECT, C. Charlotte Project. 2010. Acessado em: 20 set. 2010. Disponível em: <http://cs.nyu.edu/milan/charlotte/frmain.html>.

PVM. PVM: Parallel Virtual Machine. 2010. Acessado em: 03 ago. 2010. Disponível em: <http://www.csm.ornl.gov/pvm/>.

SQUYRES, J.; LUMSDAINE, A. A component architecture for LAM/MPI. Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer, p. 379–387, 2003.

SUNDERAM, V. PVM: A framework for parallel distributed computing. Concurrency: practice and experience, Wiley Online Library, v. 2, n. 4, p. 315–339, 1990.