Distribuição de tarefas em computação distribuída
Estudo de caso: BOINC

Alex Massao Morinaga
Hugo Posca de Vasconcelos

Orientador: Prof. Dr. Alfredo Goldman vel Lejbman

MAC0499 - Trabalho de Formatura Supervisionado

Dezembro de 2009

Sumário

Parte I
Parte Técnica

1 Introdução

Atualmente existem muitos projetos, em sua maioria científicos, em que há uma necessidade de um grande poder computacional. Porém, o custo para a obtenção e manutenção de computadores com tal poder é algo tão elevado que muitas vezes pode ser um empecilho para a realização de projetos.

O **BOINC**, um programa criado pela Universidade de Berkeley, visa tentar solucionar tal problema. A ideia do BOINC é utilizar recursos de computadores voluntários, como tempo de processamento e armazenamento em disco, para realizar tarefas de um certo projeto. Dependendo da quantidade de voluntários, um projeto pode obter um poder computacional equivalente a um supercomputador, com um custo bem reduzido.

Neste trabalho, analisaremos como o BOINC funciona, tanto em servidores de projetos como nos computadores de voluntários, sua estrutura, verificação de resultados e métodos de segurança. Assim como também analisaremos sua instalação e utilização como cliente e servidor.

2 Conceitos

2.1 Computação Distribuída

Originalmente, o termo "distribuído"referia-se a redes de computadores fisicamente distribuídos em um certo local. Atualmente o termo possui um sentido mais abrangente, podendo-se entender também como vários processos rodando em um mesmo computador mantendo uma interação entre si por mensagens.

Assim, hoje em dia, os termos "computação distribuída", "sistemas distribuídos", "programação distribuída"e "algoritmos distribuídos"referem-se a sistemas que possuam entidades computacionais autônomas onde cada uma possui a sua própria memória e a comunicação é realizada através da transmissão de mensagens. Geralmente cada computador conectado à rede não conhece totalmente como ela está organizada, simplesmente sabe de quem receber informações e para onde enviá-las.

Com essa conexão entre computadores e suas capacidades de processamento é possível dividir uma certa tarefa, para que estes computadores a executem colaborativamente, dividindo-a em pedaços e transmitindo-os para cada computador. Assim, cada computador ao invés de realizar toda a tarefa realiza apenas uma parte. Desta forma pode-se executar uma tarefa muito mais rápido do que se a executássemos em apenas um computador.

Existem três grandes razões para se utilizar sistemas distribuídos:

- Primeiro, a própria natureza da aplicação pode necessitar o uso de uma rede de comunicação que conecte vários computadores;
- Segundo, existem vários casos em que o uso de um único computador pode ser possível a princípio mas a utilização de um sistema distribuído pode ser benéfico por vários motivos, como, por exemplo, pode ser mais barato obter o mesmo nível de desempenho usando um *cluster* de vários computadores não muito potentes em comparação a um único computador de alto desempenho;
- Terceiro, o sistema é tolerante a falhas: caso ocorra algum problema em algum dos computadores, ele não comprometerá o sistema por inteiro.

2.1.1 História

A ideia de se aproveitar os ciclos do processador não utilizados é tão antiga quanto às primeiras redes que posteriormente tornaram-se a Internet. A verdadeira aplicação desta ideia evoluiu com estas redes, baseado em que terminais estavam disponíveis. Os projetos de pesquisa distribuída em operação atualmente foram criados apenas nos últimos anos com o crescimento da quantidade de usuários da Internet com acesso rápido e sempre ativo, usuários de banda rápida.

Os primeiros programas distribuídos foram dois programa chamados *Creeper* e *Reaper* (Trepadeira e Colheitador) que se espalharam através dos computadores da ARPANET, a antecessora da Internet, nos anos 1970. O *Creeper* vinha primeiro e era um programa do tipo *worm* (verme), usava os ciclos ociosos de processadores da ARPANET para se copiar para o próximo sistema e apagava-se do anterior. Posteriormente foi modificado para permanecer em todos os computadores pelos quais passava. O *Reaper* foi criado para viajar através da mesma rede e apagar as cópias do *Creeper*. Desta forma *Creeper* e *Reaper* foram os primeiras programas infecciosos e são geralmente considerados como os primeiros vírus de rede. Entretanto não faziam nenhum mal aos computadores em que passavam e foram importantes em explorar a possibilidade de se fazer uso do poder computacional ocioso.

O primeiro projeto de computação distribuída utilizando a Internet iniciou-se em 1988 pela *DEC Systems Research Center* (*Digital Equipment Corporation Systems Research Center*). O projeto enviava tarefas para voluntários, através de e-mails, que poderiam executá-las durante algum tempo ocioso e enviar os resultados de volta para a DEC e receber uma nova tarefa. O projeto era utilizado para se fatorar números grandes e por volta de 1990 tinha cerca de 100 usuários.

Nos anos 1990 a DEC e outros grupos começaram a utilizar uma interface Internet para distribuir tarefas com o propósito de realizar cálculos, impulsionada por desafios patrocinados pelo centro de pesquisa da *RSA Security Inc.*, o *RSA Laboratories*. Eram desafios como fatorações, busca de números primos e quebra de encriptações, e que ofereciam recompensas em dinheiro dado que o tempo e o poder computacional gastos deveriam ser altos para se resolver os problemas. A RSA patrocinou estes desafios como ferramentas de pesquisa, assim ela poderia resolver grandes problemas computacionais e fazer pesquisas em criptografia tendo vários grupos competindo para en-

contrar as respostas.

O grupo de maior destaque, considerado como o primeiro a verdadeiramente utilizar a Internet para distribuir dados para cálculos e coletar os resultados, criou um projeto fundado em 1997 chamado distributed.net [1]. Eles utilizaram computadores independentes, assim como a DEC havia feito, porém permitiram que os usuários baixassem o programa que utilizaria os ciclos ociosos de seus processadores, fazendo com que não houvesse a necessidade de se receber tarefas e enviar os resultados por e-mail. A distributed.net completou vários desafios de criptografia propostos pela RSA Laboratories, assim como outros institutos de pesquisa, com a ajuda de milhares de usuários.

O projeto que verdadeiramente popularizou a computação distribuída e que mostrou que ela poderia funcionar foi o SETI@HOME, feito pelo *Search for Extraterrestrial Intelligence* (SETI) (Busca por inteligência extraterrestre) [2] na Universidade da Califórnia em Berkeley. O projeto começou em maio de 1999 para analisar sinais de rádio que eram captados pelo *Arecibo Radio Telescope* (Rádio Telescópio de Arecibo) em Porto Rico. O projeto conseguiu alcançar a marca de três milhões de usuários que se voluntariaram para utilizar o tempo ocioso de seus computadores para pesquisar por sinais que possam não ter se originado na Terra.

2.1.2 Programando em Sistemas Distribuídos

Em uma arquitetura distribuída, os processadores têm sua própria memória e interagem com outros processadores usando uma rede de comunicação ao invés de uma memória compartilhada. Assim, processos não podem se comunicar diretamente através de variáveis compartilhadas, ao invés disso eles precisam trocar mensagens entre si.

Para se escrever programas para uma arquitetura de memória distribuída é primeiro necessário definir as interfaces de comunicação de rede. Isto pode ser simplesmente definir operações de leitura e escrita análogas às operações de leitura e escrita com variáveis compartilhadas, entretando, isto pode significar que programas possam ter que utilizar sincronização 'busy-waiting'. Uma melhor aproximação é definir operações de rede especiais que incluem sincronização, de forma análoga como operações de semáforo são operações especiais em variáveis compartilhadas. Tais operações em rede são chamadas de primitivas de transmissão de mensagens. De fato, a

transmissão de mensagem pode ser vista como uma extensão de semáforos para transmitir dados assim como prover sincronização. Para a transmissão de mensagens os processos devem compartilhar canais, cada canal provê o caminho de comunicação entre processos e portanto é uma abstração de uma rede de comunicação que provê um caminho físico entre processadores.

Em um programa distribuído os canais são geralmente o único objeto que os processos compartilham, assim cada variável é local a um processo. Isto implica que as variáveis nunca são alvo de acesso concorrente e portanto não há necessidade de um mecanismo especial para exclusão mútua. Isto também implica que processos precisam se comunicar para poder se interagir.

Muitos mecanismos diferentes foram propostos para computação distribuída, eles variam na forma em que os canais são nomeados e utilizados e também na forma em que a comunicação é sincronizada.

2.1.3 O Futuro

Devido ao grande avanço da computação distribuída no futuro, indivíduos e especialmente empresas poderão simplesmente se conectar a uma "rede de poder computacional"de forma semelhante a como se conectam à rede elétrica, enviar suas tarefas para esta rede e posteriormente receber resultados. Assim, será necessário apenas pagar pelo o que foi gasto e não investir excessivamente em hardware.

Redes assim foram recentemente anunciadas nos Estados Unidos e na Grã Bretanha. Estas redes poderão ser usadas não apenas para pesquisa científica mas também para uso comercial, o que pode ser muito rentável.

Algumas companhias como a *Sun Microsystems* já possuem algum tipo de rede de computadores distribuídos, chamada "fazenda de computadores". Distribuindo recursos de processamento ela foi capaz de aumentar o uso dos processadores de 5-10% para 80-90%.

Algumas empresas como por exemplo a *Distributed Science* [3] planejam pagar a seus clientes para que eles permitam a execução de aplicações distribuídas em suas máquinas pagando o suficiente para os custos mensais de uma conexão à Internet.

Companhias de entretenimento podem explorar a computação distribuída como um meio de reduzir custos, permitindo a criação de filmes cada vez mais rápido. Como por exemplo podemos citar a Pixar, o estúdio de ani-

mação, que utilizou uma rede de 100 computadores durante 20 horas para se criar algumas cenas de ação para o filme Toy Story [4, 5].

2.2 Clusters e Computação em Grade

Um *cluster* ("aglomerado") é um grupo de computadores ligados entre si, trabalhando juntos, equivalendo, em muitos aspectos, a um único computador. Eles são geralmente utilizados para melhorar o desempenho e/ou disponibilidade em relação a um único computador, além disso, seu custo geralmente é menor do que utilizar um único computador com desempenho semelhante.

Uma característica que difere um cluster do outro é a ligação entre os computadores. Por exemplo, um computador pode precisar se comunicar frequentemente com outros computadores para conseguir prosseguir com seu trabalho, este tipo de construção é geralmente chamado de *Beowulf cluster*. Por outro lado, os computadores podem precisar de muito pouco ou até mesmo nenhuma comunicação com os outros.

Computação em grade (ou o uso de grades computacionais) é a combinação de recursos computacionais de múltiplos domínios administrativos aplicados a uma mesma tarefa, geralmente para um problema científico, técnico ou empresarial que requer um grande número de ciclos de processamento ou necessita processar grandes quantidades de dados.

Uma das principais estratégias da computação em grade é utilizar *softwares* para dividir e repartir pedaços de um programa entre vários computadores, podendo chegar a milhares. A computação em grade é um *cluster* de grande escala, assim como uma forma de processamento paralelo distribuído em rede. O tamanho da computação em grade pode variar de pequena, como uma rede de estações de trabalho dentro de uma corporação, a grande, como colaborações públicas entre várias companhias e redes.

2.3 Computação voluntária

Computação voluntária refere-se a uma ideia onde pessoas, voluntárias, oferecem os recursos computacionais de seus computadores a projetos, e estes utilizam tais recursos para uma computação distribuída e/ou armazenamento. Os voluntários geralmente são pessoas do público em geral, que

possuem computadores com conexão à Internet, assim como também podem ser organizações como escolas e empresas.

Os projetos normalmente são acadêmicos (universitários) e realizam pesquisas científicas mas também existem projetos feitos por companhias particulares como o GIMPS [6] e distributed.net.

Os voluntários são anônimos, ou seja, apesar de terem de se registrar e fornecer informações como e-mail, eles não estão ligados com suas identidades reais. Devido a isso, os voluntários não são responsabilizados pelos projetos. Se um voluntário fizer algo problemático como, por exemplo, fornecer resultados errados intencionalmente ou burlar o sistema de pontuação, que alguns projetos utilizam, o projeto não pode puni-lo de alguma forma.

Os voluntários também devem confiar no projeto que estão dispostos a ajudar, em vários aspectos: eles devem confiar que o projeto não enviará aplicativos que possam danificar o computador ou invadir sua vida pessoal; que o projeto diz a verdade sobre que tipo de trabalho seus computadores estão fazendo, e que os resultados serão utilizados de forma apropriada; e que o projeto seguirá práticas de segurança adequadas, para que hackers não possam utilizá-lo como um veículo para atividades maliciosas.

A computação voluntária é interessante pelo fato de poder aproveitar o poder computacional de muitos computadores do mundo (podendo chegar a milhares ou mesmo milhões), sendo uma opção muito mais barata que a obtenção de um supercomputador, por exemplo. Porém, para obter tal poder computacional os projetos devem merecê-lo, fazendo "propaganda"de si para o público e ao fazerem isto os projetos incentivam o interesse do público sobre a ciência.

A computação voluntária se diferencia da computação em grade em alguns fatores. Por exemplo, na computação em grade, cada parte da grade pode agir como produtor ou consumidor de recursos e também nela não existe anonimidade, fazendo com que cada computador possa receber punições caso não faça o que deve.

2.3.1 História

O termo "Computação Voluntária"foi estabelecido por Luis F. G. Sarmenta [7], o desenvolvedor do projeto Bayanihan [8].

O primeiro projeto de computação voluntária a ser implementado foi o

"Great Internet Mersenne Prime Search"(GIMPS), que pesquisa os números primos de Mersenne, e que começou em 1996 e em 1997 foi seguido pelo distributed.net. Em 1997 e 1998 muitos projetos de pesquisa acadêmica desenvolveram sistemas de computação voluntária baseados em java como o próprio Bayanihan, Popcorn [9], Superweb e Charlotte [10].

Em 1999 lançaram-se os projetos SETI@home e Folding@home [11], responsáveis pela busca por inteligência extra terrestre e simulações intensas para o estudo do enovelamento de proteínas, respectivamente. E em seus lançamentos receberam uma considerável cobertura da mídia o que ajudou a atrair centenas de milhares de usuários.

2.3.2 Estrutura da computação voluntária

No início o software cliente dos projetos de computação voluntária consistia em um único programa que combinava toda a infraestrutura da computação distribuída e da computação científica, esta arquitetura monolítica era inflexível, o que prejudicava, por exemplo, o lançamento de novas versões. Recentemente os projetos de computação voluntária utilizam um sistema de middleware que provê uma infraestrutura de computação distribuída independente da de computação científica.

A maioria dos sistemas atuais segue a mesma estrutura básica para seu funcionamento: o programa cliente roda no computador de um voluntário e este programa periodicamente contacta os servidores do projeto a que é afiliado, requisita trabalhos novos e reporta os resultados de trabalhos completos. A utilização deste "sistema de empurrão"é necessária porque os computadores de muitos voluntários estão atrás de firewalls que não permitem conexões de entrada. Para alguns projetos há o controle dos créditos de cada usuário, que são fornecidos assim que uma resposta correta é enviada aos servidores e que serve como uma medida numérica de quanto trabalho o computador de cada usuário fez para o projeto.

Os sistemas de computação voluntária devem saber lidar com muitos aspectos problemáticos dos computadores dos usuários como por exemplo:

- Sua heterogeneidade, já que os usuários possuem computadores com configurações diferentes;
- Sua disponibilidade, já que um usuário pode não utilizar o computador durante todo o dia como também pode disponibilizar os recursos do

computador em apenas algumas horas do dia;

- Deve garantir que a execução da aplicação não interfira no desempenho do computador durante o uso pelo usuário.

Assim como saber lidar com problemas estruturais como:

- Garantir a exatidão nas respostas, tendo que evitar que respostas incorretas sejam aceitas e créditos atribuídos aos usuários que as enviaram;

- Replicar tarefas, já que não temos nenhuma garantia que após o envio de uma tarefa ela será concluída pelo computador do usuário a que foi enviada.

2.3.3 Custos da computação voluntária para o voluntário

Ao participar de um projeto o voluntário deve estar ciente que está sujeito a ter alguns gastos. Como o processador do computador ao invés de ficar inativo realiza as tarefas do projeto e de vez em quando o usuário deixa o computador ligado durante as madrugadas com o intuito de ajudar o projeto, há um aumento no gasto de energia elétrica. Há também o custo de desempenho da utilização do computador enquanto se realizam tarefas para um projeto e o voluntário o utiliza para seus fins normais ao mesmo tempo, embora este seja geralmente baixo não pode ser totalmente desconsiderado.

3 BOINC

3.1 Introdução

O BOINC, **B**erkeley **O**pen **I**nfrastructure for **N**etwork **C**omputing [12], é um sistema de middleware para computação voluntária e distribuída especialmente voltado para aplicações do tipo mestre-escravo, e desenvolvido sob a LGPL (*GNU Lesser General Public License*), ou seja, seu código é livre para ser usado por todos, mas os trabalhos que o utilizam não precisam ser necessariamente abertos. Projetos que utilizam-se de tal arquitetura podem obter um grande poder computacional utilizando computadores de voluntários ao redor do mundo a um custo muito reduzido se comparado a supercomputadores com desempenho semelhante.

Originalmente o BOINC foi desenvolvido para o projeto SETI@home e teve sua primeira versão em 2002, mas sua primeira versão estável foi lançada apenas em 2004. O primeiro projeto que o utilizou foi o Predictor@home [13], que estudava o desdobramento de proteínas e posteriormente o SETI@home migrou totalmente para o BOINC. Em suas primeiras versões seu grau de segurança não era algo muito elevado, o que permitiu que alguns usuários tentassem obter mais créditos do que deveriam e também que resultados errados fossem submetidos e aceitos. Devido a seu potencial e à grande quantidade de usuários que o utilizava acabou-se tornando útil para outras aplicações distribuídas em áreas diversas como matemática, medicina, biologia molecular, climatologia e astrofísica.

Atualmente (2009) conta com 62 projetos afiliados das mais diversas áreas como medicina, biologia, ciências naturais, matemática, física, astronomia, inteligência artificial e jogos. Por dia atualmente são processados 2.7 PetaFLOPS nos aproximadamente 586.000 computadores dos 327.000 voluntários ativos [14]. Para se ter uma ideia o atual supercomputador mais rápido do mundo, o *IBM RoadRunner*, consegue executar por volta de 1.5 PetaFlops [15].

3.2 Estrutura

O BOINC pode ser dividido em duas grandes partes, o cliente e o servidor. O cliente é o aplicativo que usuários instalam em suas máquinas e que posteriormente recebe tarefas dos servidores, as processam e devolve os resultados obtidos para os servidores. O servidor é onde os dados de um

projeto são gerados, distribuídos para os usuários como tarefas e após os resultados serem recebidos são armazenados e analisados.

3.2.1 Sistema de créditos

No BOINC existe o chamado sistema de distribuição de créditos, que consiste basicamente em recompensar o voluntário com créditos em uma quantidade baseada em sua contribuição para o projeto. Devido à grande comunidade de usuários é comum encontrar em fóruns pessoas colocando uma lista de créditos ganhos por projeto.

3.2.2 Linguagens aceitas

Inicialmente, o BOINC foi feito para suportar aplicações feitos em C/C++. Atualmente, existem certas maneiras de aceitar outras linguagens: FORTRAN (o site oferece diversas maneiras para este), Java (atualmente, para Windows apenas) e Python, tanto para aplicações como para scripts.

3.2.3 Segurança

A computação voluntária possui um problema: como são pessoas desconhecidas participando dos projetos, nem todas são confiáveis. Por isso, o BOINC utiliza certas medidas de segurança para evitar eventuais fraudes ou ataques aos projetos.

Descreveremos aqui alguns problemas de segurança comumente encontrados e as soluções adotadas pelo BOINC.

Falsificação de resultados e créditos

Dois dos problemas mais comuns na computação voluntária são a possibilidade de fraude na obtenção de créditos e no envio de resultados. Geralmente para obterem mais créditos alguns usuários mal intencionados modificam seus programas de tal forma que estes aleguem que o tempo de CPU utilizado para processar determinada tarefa foi maior do que o que realmente foi gasto. Para tentar se evitar tais fraudes, o BOINC conta com um sistema de validação de respostas que é configurado pelos administradores de cada projeto.

Algumas das opções de validação de respostas são:

- Sem replicação: Onde uma tarefa é enviada para apenas um voluntário e após o resultado ser recebido créditos são cedidos ao usuário. Este esquema é útil quando é possível detectar um erro com uma alta probabilidade. Quanto ao problema de fraude na obtenção de créditos existem algumas opções:
  - Todos os usuários sempre recebem uma mesma quantia de créditos - útil para tarefas uniformes;
  - Cria um limite máximo para os créditos concedidos; ainda é possível existir fraudes, pois ainda é possível tentar aumentar a quantidade de créditos, se não ultrapassar tal limite;
  - Se a quantidade de créditos pedida pelo cliente for maior que um certo limite, replica a tarefa.
- Com replicação: Onde uma tarefa é enviada para mais de um usuário e após estes devolverem uma resposta ela é comparada. Se forem iguais, dependendo dos critérios utilizados, todos os usuários são recompensados com créditos.

  Um problema deste método é que podem haver pequenos erros entre dois resultados, devido aos computadores poderem fazer contas em ponto flutuante com precisões diferentes. Existem certas maneiras de arrumar tal problema:
  - Eliminar tais discrepâncias, selecionando compiladores, opções de compilador e bibliotecas matemáticas apropriadas, tendo certeza de que o resultado será completamente preciso; este método é difícil de se realizar e geralmente reduz o desempenho do programa;
  - Se a aplicação for numericamente estável (pequenas discrepâncias resultam em pequenos erros no resultado), pode-se comparar os resultados com uma certa tolerância;
  - Após mandar uma tarefa para um cliente, a mesma tarefa é mandada para outros computadores numericamente equivalentes, isto é, que devolverão resultados identicamente iguais por *bit*. A noção de "numericamente equivalentes"depende da aplicação e de como ela foi compilada.

- Com replicação adaptativa: Uma tarefa é enviada para um usuário. Se a taxa de erro deste for baixa, ou seja, se o resultados fornecidos por este usuário forem geralmente certos, a chance de replicar tal tarefa será baixa. Como há a necessidade de se armazenar a taxa de erro do usuário, esta abordagem funciona melhor com usuários que já participam do projeto há um certo tempo.

Distribuição de executáveis maliciosos

Isto pode acontecer quando um servidor é atacado e os arquivos executáveis são modificados com o intuito de fazer com que os clientes executem programas maliciosos. Cada projeto BOINC e seus respectivos executáveis possuem uma assinatura digital única, fazendo com que o programa cliente apenas aceite os programas que possuam as assinaturas corretas. Desse modo, mesmo que o servidor seja atacado os clientes não aceitarão um programa malicioso pois este não terá a assinatura correta.

É importante salientar que para que este mecanismo de defesa funcione corretamente é necessário, do lado do projeto, que as assinaturas fiquem armazenadas em um computador não conectado à rede e fisicamente protegido contra uso indevido.

Transbordamento dos servidores de dados

Isto pode acontecer quando quem está atacando o projeto repetidamente envia arquivos grandes para os servidores de dados do BOINC, enchendo seus discos e tornando-os inutilizáveis por falta de espaço.

Para prevenir isto o BOINC não permite o envio de arquivos maiores que um determinado tamanho, que é estabelecido pelos administradores do projeto.

Roubo de contas por ataque ao servidor

Isto pode acontecer quando quem está atacando um projeto invade um servidor BOINC e rouba endereços de e-mail e outras informações de contas.

Para evitar isto cada projeto deve abordar o roubo de informações privilegiadas das contas utilizando práticas convencionais de segurança. Todas as máquinas utilizadas pelos servidores devem ser protegidas por *firewalls* e ter todos os seus dispositivos de rede que não são utilizados desabilitados,

assim como o acesso a tais máquinas deve ser feito apenas por protocolos encriptados como por exemplo o *SSH*.

Recomenda-se que os projetos apenas sejam empreendidos por organizações que possuam conhecimento e recursos suficientes de segurança, pois um ataque bem sucedido pode danificar a reputação de todos os projetos que utilizam o BOINC, assim como a computação voluntária em geral.

Roubo de arquivos de entrada/saída

Os arquivos de entrada/saída utilizados pelas aplicações do BOINC não são encriptados. Se um projeto quiser pode utilizar aplicativos para encriptar e desencriptar os dados; porém isto tem um efeito muito pequeno na tentativa de se evitar o roubo destes arquivos, já que os dados residem em formato texto na memória do computador, onde são facilmente acessados utilizando-se um *debugger*.

Abuso dos projetos aos participantes

Podem existir dois tipos de abuso: O intencional e o acidental.

O abuso intencional ocorre quando um projeto lança uma aplicação que abusa do computador do usuário, por exemplo roubando informações armazenadas em arquivos. Para se evitar isto o BOINC utiliza um sistema de isolamento de contas (*account-based sandboxing*) fazendo com que as aplicações sejam executadas por uma conta de usuário especialmente criada. Se as permissões de arquivos e diretórios estiverem devidamente configuradas, as aplicações não conseguirão acessar arquivos externos ao diretório do BOINC.

O abuso acidental ocorre quando um projeto lança uma aplicação que involuntariamente abusa do computador do usuário, como por exemplo apagando arquivos ou paralisando o sistema. O BOINC previne alguns problemas: ele pode, por exemplo, detectar que uma aplicação utiliza muito espaço em disco, memória ou tempo de processamento e abortá-la. Projetos podem minimizar a probabilidade de causar problemas realizando pré-lançamentos de aplicações teste, assim como também devem testar suas aplicações cuidadosamente em todas as plataformas e com todas as possíveis entradas antes de as lançarem oficialmente.

3.2.4 Servidor

Um servidor BOINC pode consistir de um ou mais computadores, o que o torna configurável para projetos de qualquer tamanho. Os servidores BOINC rodam em sistemas Linux e utilizam Apache e PHP como base para web e MySql como base para banco de dados. O servidor é responsável por distribuir tarefas para os computadores dos usuários e os resultados recebidos são posteriormente analisados e validados.

Configurações recomendadas

Para se criar um servidor BOINC recomenda-se:

- Ter uma conexão com a Internet com desempenho adequado e garantia que desconexões não sejam frequentes, assim como ter um endereço IP estático;
- O servidor deve ter uma boa velocidade de processamento, recomendando-se processadores Xeon ou Opteron, no mínimo 2 GB de RAM e 40 GB de espaço livre em disco. Para um projeto com grande tráfego de dados recomenda-se máquinas com 8 GB de RAM ou mais e processadores de 64 bits;
- Fazer o máximo possível para garantir que o servidor esteja sempre disponível, como por exemplo colocá-lo em uma sala com temperatura controlada e garantir o fornecimento contínuo de energia elétrica;
- Tomar medidas de segurança como colocá-lo atrás de um firewall e desabilitar serviços de rede desnecessários.

Workunits

No BOINC a computação a ser realizada pelo usuário é chamada de *workunit*. O tamanho e o conteúdo de uma *workunit* é estritamente dependente do projeto e seu tamanho pode variar de pequeno - cerca de 200 *kilobytes* - até muito grande - cerca de alguns *megabytes*. O tempo de processamento também varia de acordo com as especificações e dependências de cada projeto, com tempos conhecidos de alguns minutos a alguns meses.

Como o tamanho médio das *workunits* de um projeto é conhecido, um usuário pode saber com antecedência se tem condições de disponibilizar o seu

computador para tal projeto. Mais exatamente, como as partes do servidor responsáveis pelo agendamento e pelos dados saberão as características do computador de um determinado participante, eles não enviarão um trabalho que não pode ser completado por aquele computador.

Workunits são descritas através de um modelo de *workunit* e um modelo de resultado, cada um em um arquivo XML. O modelo de *workunit* descreve o conjunto de dados que será utilizado pelo computador que realizará a tarefa por ela determinada, e o modelo de resultado descreve o conjunto de resultados e como estes devem ser enviados de volta para o servidor.

Exemplo de modelo de *workunit*:

```
<file_info>
  <number>0</number>
</file_info>
<workunit>
  <file_ref>
    <file_number>0</file_number>
    <open_name>in</open_name>
  </file_ref>
  <min_quorum>1</min_quorum>
  <target_nresults>1</target_nresults>
</workunit>
```

Exemplo de modelo de resultado:

```
<file_info>
  <name><OUTFILE_0/></name>
  <generated_locally/>
  <upload_when_present/>
  <max_nbytes>10000</max_nbytes>
  <url><UPLOAD_URL/></url>
</file_info>
<result>
  <file_ref>
    <file_name><OUTFILE_0/></file_name>
    <open_name>out</open_name>
  </file_ref>
</result>
```

Funcionamento

O servidor consiste em um conjunto de programas escritos em C++ do tipo *daemon*, ou seja, programas que são executados continuamente e que têm uma função específica dentro do sistema BOINC.

A seguir colocaremos o que estes programas fazem seguindo a ordem normal de execução de um projeto:

O *daemon* agendador cuida de pedidos vindos dos clientes, recebe resultados de tarefas completas e envia novos trabalhos a serem executados. O agendador não pega tarefas diretamente da base de dados, em vez disso existe um *daemon* alimentador que carrega novas tarefas e as mantém em um bloco de memória compartilhada que o agendador lê. O alimentador periodicamente completa espaços vazios no bloco de memória compartilhada depois que o agendador as enviou para algum cliente.

Após todos os resultados de uma *workunit* serem completados e coletados o *daemon* validador os compara. O validador pode utilizar código especificado pelo projeto para fazer um determinado tipo de comparação ou simplesmente realizar uma comparação *bit* a *bit*. Se os resultados forem equivalentes a *workunit* é marcada como válida e um resultado é escolhido como resultado base.

Em seguida o *daemon* assimilador processa o resultado base utilizando código específico do projeto. Por exemplo, alguns projetos podem analisar o arquivo e armazenar dados obtidos do arquivo em uma base de dados, enquanto outros podem simplesmente copiar o arquivo em algum lugar. O assimilador, se especificado, também pode gerar mais *workunits* baseado no resultado obtido.

O *daemon* limpador de arquivos apaga arquivos recebidos depois que o assimilador os processou, e apaga também arquivos que são enviados para os usuários mas que já não serão mais utilizados.

O *daemon* limpador de banco de dados é responsável por eliminar entradas na base de dados que não serão mais utilizadas, geralmente porque todos os trabalhos que a utilizavam já foram concluídos. Isto permite que a base da dados tenha um tamanho quase constante mesmo com o projeto sendo executado por muito tempo.

3.2.5 Cliente

Os voluntários que estão dispostos a ajudar em projetos ligados ao BOINC devem instalar o programa cliente em um ou mais computadores. As configurações necessárias para poder se executar o cliente são muito baixas, sendo que as configurações recomendadas são em média de processadores com 500 MHz, 64 MB de RAM e 50 MB de espaço em disco disponível. Após a instalação, informações sobre o *hardware* do computador onde o cliente foi instalado são coletados para permitir que apenas tarefas capazes de serem realizadas por este computador sejam atribuídas a ele. E depois que resultados são enviados para os servidores dos projetos, os créditos obtidos podem ser observados através de um gráfico, o que auxilia na observação do quanto um usuário contribui para cada um.

Instalação

A instalação é bem simples, basta baixar o arquivo oferecido pelo próprio site do BOINC [16] e seguir as instruções. Para usuários Linux, existe a opção de verificar se um pacote para a sua distribuição é fornecido.

Ao instalar o programa cliente do BOINC no computador, três componentes são instalados automaticamente:

- ***Daemon* do BOINC**: é o responsável pelo controle e execução das aplicações. Recebendo "ordens"do gerenciador do BOINC através de mensagens;

- **Gerenciador do BOINC**: é a *GUI* (*Graphical User Interface* - Interface gráfica do Usuário) do BOINC. O usuário pode verificar o progresso das aplicações, mensagens do servidor, créditos obtidos até o momento, além de modificar preferências (como espaço utilizado e horários de funcionamento do programa), entrar ou sair de um projeto, suspender tarefas, entre outros;

- **Protetor de tela do BOINC** (se o sistema operacional aceitá-lo): aplicações que possuem esta opção mostrarão gráficos relacionados à tarefa sendo feita no momento. Caso não tenha tal opção, ele mostrará as aplicações e o quanto foi rodado, ou se não existem tarefas sendo executadas.

Conectando-se a um projeto

Para conectar-se a um projeto, além de possuir o programa cliente do BOINC instalado no computador é necessário criar uma conta no projeto que se deseja ajudar. Para isto basta, a partir do próprio programa cliente, escolher qual o projeto se quer ajudar e criar uma conta informando apenas um e-mail e uma senha.

Uma vez participando de um projeto, o usuário pode desconectar-se dele quando desejar. Também pode escolher suspender ou cancelar tarefas, não receber tarefas, reiniciar projetos, entre outros.

Preferências do usuário

Usuários podem editar certas preferências em relação a um ou todos os projetos.

Existem dois tipos de preferência:

- **Preferências gerais**: São as preferências aplicadas a todos os projetos que um usuário é afiliado. Isto inclui preferências como: dias e horários para se executarem tarefas; Limite máximo de uso de CPU e espaço; Tempo de alternação entre projetos; Limite de *download* e *upload*; Número máximo de processadores utilizados;

- **Preferências do projeto**: preferências aplicadas a apenas um projeto como, por exemplo, a cor do protetor de tela deste projeto.

Para as preferências gerais, existem dois modos de modificá-las:

- Pela *web*: é possível modificá-las a partir do site de qualquer projeto em que se é afiliado. Deste modo, todos os projetos serão afetados, porém não imediatamente. O projeto inicial será modificado na hora e eventualmente, os computadores ligados àquele projeto terão suas preferências modificadas, e então todos os projetos ligados àquele usuário e computador também serão modificados;

- Pelo gerenciador do BOINC: Apenas o computador em questão terá suas preferências modificadas. Estas preferências sobrepõem as feitas pela *web*.

4 Atividades realizadas

4.1 Participando de um projeto

Para se analisar como é a utilização do BOINC por um voluntário desde março nos tornamos voluntários e utilizamos o BOINC em nossos próprios computadores. Assim pudemos verificar quão complicada é realmente a instalação e quais são os impactos causados por deixar o BOINC realizando tarefas enquanto utilizamos nossos computadores para fazer atividades cotidianas.

Em um computador com processador Pentium de 1.6 GHz e com 256 MB de RAM, a instalação no Windows XP foi relativamente fácil, bastando apenas baixar o arquivo do site e realizar a instalação de forma corriqueira sem diferenças em relação à instalação de outros programas. No Ubuntu 9.04, a instalação foi feita através do gerenciador de pacotes *apt-get*, e também foi instalado sem problemas. Em ambos os sistemas, utilizou-se a mesma conta para realizar tarefas de dois projetos: *Einstein@home* [17] e *World Community Grid* [18], que utilizam dados do detector de ondas gravitacionais LIGO na busca por pulsares e fazem pesquisar humanitárias, respectivamente.

Em um computador com processador AMD Athlon de 2 GHz e com 1536 MB de RAM foi realizada a instalação apenas no Ubuntu 8.04 através de um arquivo de instalação que é conseguido acessando-se o site de download do BOINC e ocorreu sem problemas, sendo parecida com as instalações feitas via gerenciador de pacotes. Neste sistema participamos do projeto *Einstein@home*.

Utilizando ferramentas como *top*, *htop* e *GKrellm* vimos que quando executado em sistemas Linux o cliente BOINC executa as tarefas em modo '*nice*', o que pode ser confirmado observando-se que sua prioridade tanto no *top* quanto no *htop* é muito baixa: 39. E também utilizando-se o utilitário *GKrellm* pudemos observar que enquanto o cliente BOINC executa suas tarefas o gráfico de execuções utiliza a cor estipulada para representar processos em '*nice*'.

4.2 Criando um projeto

Após utilizarmos nossos computadores para ajudar os projetos, analisamos como é ter um projeto. Para isto precisamos instalar um servidor BOINC e configurá-lo corretamente.

Para instalar um servidor BOINC utilizamos o computador AMD Athlon de 2 GHz com 1536 MB de RAM e com Ubuntu 8.04 instalado. Antes de começar a instalação do servidor em si precisamos instalar os aplicativos que ele depende, ou seja, MySql, PHP e Apache. Depois baixamos a última versão (6.11.0) do código fonte do servidor e também algumas bibliotecas de MySql, PHP e Apache que são necessárias para a compilação.

Após compilarmos o servidor precisávamos estudar como ele distribui tarefas e obtém resultados. Para isto utilizamos um programa exemplo que vem junto com o código fonte e que converte um texto para letras maiúsculas e utiliza um laço para consumir tempo, compilamos ele, o ligamos às chaves de segurança que queríamos utilizar e o adicionamos na base de dados do BOINC.

Posteriormente criamos as *workunits* que seriam utilizadas por este programa, e que foram criadas utilizando-se o mesmo padrão apresentado anteriormente, e então as enviamos para o banco de dados do servidor.

Após o programa teste e suas respectivas *workunits* estarem registradas no banco de dados, precisamos efetivamente iniciar o servidor, para isto ativamos a execução dos aplicativos *daemon* e após um pequeno intervalo de tempo o servidor estava funcionando.

Com os clientes instalados em nossos computadores nos afiliamos ao nosso próprio projeto e começamos a receber tarefas e observar sua execução. Depois de um tempo, conforme as tarefas eram concluídas os resultados eram devolvidos ao servidor.

Do lado do servidor pudemos observar os resultados das tarefas sendo gravados no diretório estipulado para recepção e vimos que estes eram como esperado.

Após a execução do programa teste o modificamos para utilizar um laço que gastava mais tempo e assim observamos no programa cliente uma execução mais lenta e um aumento mais gradual na barra de execuções.

Segue abaixo tal laço. Ele realiza 2^{34} operações simples de soma.

```
1 static void burnCPUTime() {
2     double a = 0;
3
4     for (int i = 0; i < 1<<17; i++)
5         for (int j = 0; j < 1<<17; j++)
6             a += 0.000001;
7 }
```

4.3 Análise de desempenho

Para analisar o impacto causado por se utilizar o BOINC enquanto o computador é utilzado pelo usuário pensamos no seguinte teste: se um programa executado pelo usuário está consumindo uma porcentagem fixa do processador, gostaríamos de saber qual seria o impacto se este mesmo programa fosse executado juntamente com o BOINC.

Para conseguir realizar tal teste de uma forma mais automática precisamos encontrar uma forma de limitar a utlização do processador para um determinado aplicativo, o que foi possível ao se utilizar o programa *cpulimit* [20]. Com ele é possível estipular qual a porcentagem máxima que um aplicativo pode utilizar do processador.

Criamos um *script* que executa o aplicativo definido pelo usuário com o BOINC parado e com o BOINC em execução. Nele temos um laço que executa o aplicativo 5 vezes, na primeira iteração com limitação de 10% de utilização do processador e com as demais iterações com incremento de 10% até 100%. A cada iteração o tempo de cada execução é armazenado; após realizarmos todos os testes a média é calculada, e geramos um gráfico comparativo das médias e mostrando os tempos máximo e mínimo de cada caso.

Assim, criamos um programa, que utilizando 100% do processador leva em média 25,6 segundos no AMD Athlon de 2 GHz e 43,6 segundos no Pentium de 1,6 GHz, e o utilizamos em nosso *script*. Devido à grande quantidade de execuções e às restrições de utilização do processador estes testes levaram cerca de duas horas para serem executados no AMD Athlon de 2GHz e cerca de quatro horas e meia no Pentium de 1,6 GHz.

Código do aplicativo utilizado para fazer os testes:

```
#define MAX 100000

int main (void) {
    int cont;
    int cont2;

    for( cont = MAX; cont > 0; cont-- ) {
        for( cont2 = MAX; cont2 > 0; cont2-- ) {
        }
    }
}
```

Código do nosso *script* responsável por executar os testes:

```
#!/bin/bash

if [ $# != 1 ]; then
    echo $0 "[nome do aplicativo]"
else
    echo "Iniciando testes"
    rm saida dados 2> /dev/null
    touch saida # so cria um arquivo saida vazio

    #
    # Primeira parte dos testes
    #
    sudo /etc/init.d/boinc-client stop # para o BOINC

    # Roda 5 vezes o aplicativo a 10%, 5 a 20% ... 5 a 100%
    for porcentagem in `seq 10 10 100`
    do
        for vez in `seq 1 5`
        do
          ./$1 &
          pid=$!
          (time sudo cpulimit -p $pid -zl $porcentagem) 2>> saida
        done
    done

    echo "Fim da primeira parte"

    #
    # Segunda parte dos testes
```

```
30      #
31      sudo /etc/init.d/boinc-client start # inicia o BOINC
32
33      # Roda 5 vezes o aplicativo a 10%, 5 a 20% ... 5 a 100%
34      for porcentagem in `seq 10 10 100`
35      do
36          for vez in `seq 1 5`
37          do
38            ./$1 &
39            pid=$!
40            (time sudo cpulimit -p $pid -zl $porcentagem) 2>> saida
41          done
42      done
43
44      echo "Fim da segunda parte"
45
46      #
47      # Terceira parte dos testes
48      # Encontra a media e plota o grafico com media,
49      # minimo e maximo de cada porcentagem
50
51      echo "Plotando grafico"
52
53      ./media < saida
54
55  gnuplot << EOF
56  set term postscript enhanced color
57  set output "grafico10pl.ps"
58  set title "Impacto no desempenho"
59  set xrange [0:109]
60  set xtics 10
61  set mxtics 2
62  set yrange [0:]
63  set ytics 20
64  set mytics 2
65  set xlabel "Porcentagem de uso da CPU"
66  set ylabel "Tempo (s)"
67  plot \
68     "dados" index 0 title "BOINC nao ativo" with yerrorbars lt 1,\
69     "dados" index 1 title "BOINC ativo" with yerrorbars lt 3,\
70     "dados" index 0 smooth csplines notitle lt 1,\
71     "dados" index 1 smooth csplines notitle lt 3
72  unset output
73  unset xrange
```

```
74 unset yrange
75 unset xlabel
76 unset ylabel
77 EOF
78 fi
79
80 echo "FIM."
```

Código do *script* que calcula as médias:

```
1  #!/usr/bin/perl
2
3  open ARQ, ">dados" or die "Nao consegui criar arquivo\n";
4  $soma = 0;
5  $min = 10000000;
6  $max = -1;
7  $vezes = 0;
8  $porc = 10;
9
10 while (<>) {
11     if (/^real/) {
12         $vezes++;
13
14         /(\d+)m(\d+).(\d+)s/;
15         $aux = $1 * 60;
16         $aux += $2 + ($3 * 0.001);
17         $soma += $aux;
18
19         $max = $aux if ($max < $aux);
20         $min = $aux if ($min > $aux);
21
22         if ($vezes == 5) {
23             print ARQ "$porc ", $soma/5," $min $max\n";
24
25             $soma = 0;
26             $min = 10000000;
27             $max = -1;
28             $vezes = 0;
29             $porc += 10;
30             if ($porc > 100) {
31                 print ARQ "\n\n";
32                 $porc = 10;
33             }
```

```
34              }
35          }
36  }
37
38  close ARQ;
```

Seguem abaixo os gráficos dos testes realizados nos dois computadores. Para tentar diminuir a interferência de outros programas efetuamos os testes com a menor quantidade de programas em execução possível.

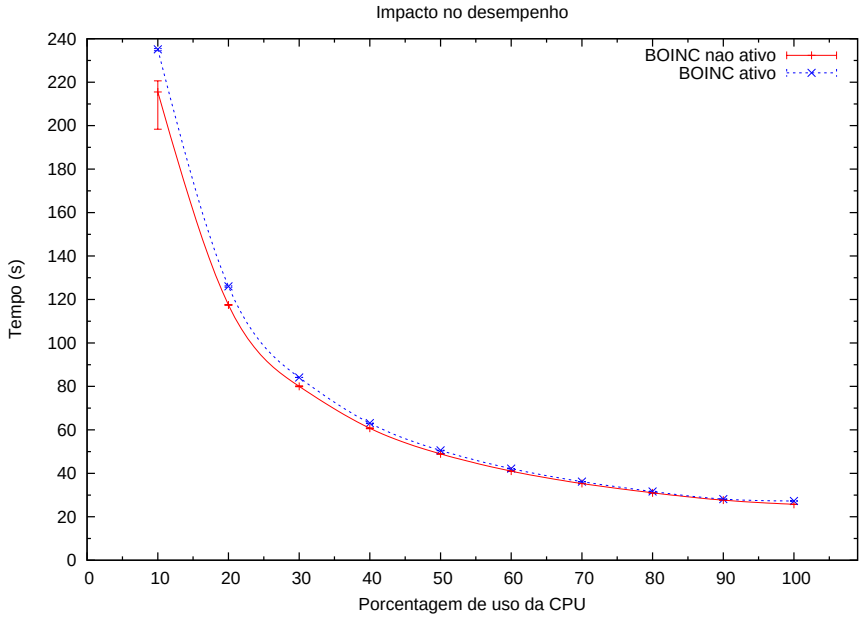

Figura 1: Gráfico ao realizarmos os testes no computador AMD Athlon de 2 GHz

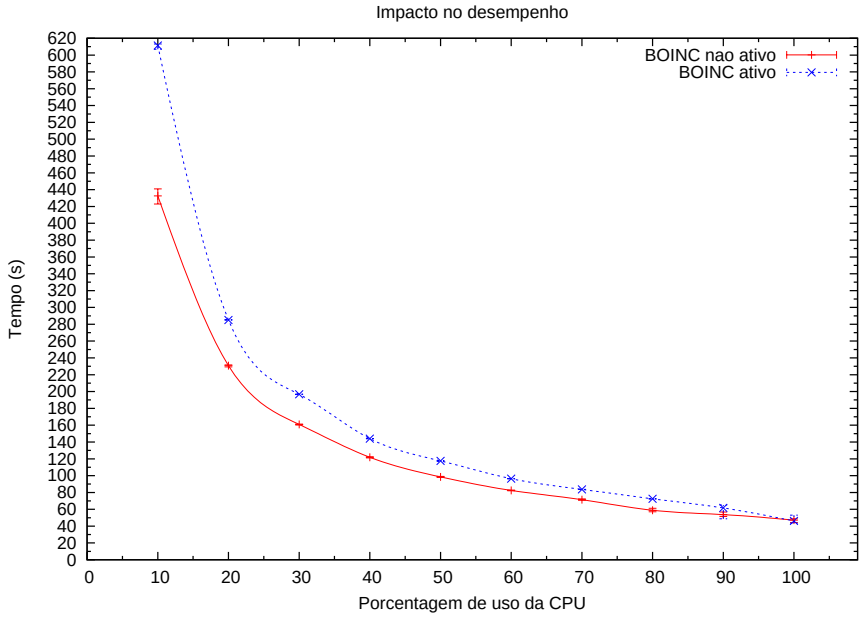

Figura 2: Gráfico ao realizarmos os testes no computador Pentium de 1,6 GHz

A partir dos gráficos, podemos perceber que quanto maior a utilização do processador pelo programa, menor a interferência causada pela utilização do BOINC.

5 Conclusões

Ao utilizarmos o BOINC como voluntários percebemos como os desenvolvedores focaram na sua utilização, fazendo com que a instalação e a execução de tarefas sejam processos simples, assim como também tomaram o cuidado de fazer com que as tarefas sejam executadas de modo a não interferir nas atividades do usuário enquanto ele utiliza o computador.

Toda esta simplicidade ajuda a atrair voluntários pois, tanto pessoas que estão acostumadas a instalar programas quanto aquelas que não estão acostumas e têm receio que novos programas afetem o desempenho de seus computadores podem instalar de forma fácil o programa cliente e assim, uma quantidade muito grande de pessoas pode ajudar os vários projetos em suas pesquisas. Foi devido a toda a esta simplicidade que o BOINC conseguiu ter a quantidade de usuários que tem hoje e se tornar algo tão atrativo para projetos que necessitem de um grande poder computacional.

Apesar de a instalação de um servidor BOINC parecer simples, na verdade é um processo complexo e com muitas dependências. Tivemos muitos problemas para conseguir com que tudo funcionasse, mas depois que conseguimos tudo ocorreu conforme o esperado. Devido às dificuldades na instalação e à grande quantidade de informações que podem ser geradas e obtidas é recomendável ter uma equipe dedicada ao funcionamento de um projeto.

Assim, com todo o potencial que o BOINC possui, tanto universidades quanto pequenas e médias empresas podem aproveitar todos os computadores em suas redes, com poucos custos adicionais, para realizar tarefas que demandem grande poder computacional.

Parte II
Parte subjetiva

5.1 Desafios e frustrações

Uma grande frustração que obtivemos foi quando tentamos instalar o servidor. Obtivemos alguns problemas durante a instalação do Apache e do MySql, e após estes terem sido resolvidos, quando tentamos instalar o servidor, o código não compilava. Depois de pesquisas, descobrimos que faltava instalar algumas bibliotecas de desenvolvimento no computador que não eram mencionadas como necessárias, nem no site do BOINC, nem no outro site [19] que utilizamos como referência.

Embora o BOINC seja um sistema utilizado por muitas pessoas e projetos, encontrar respostas para problemas específicos foi um grande desafio. Por exemplo, enquanto instalávamos o servidor obtivemos várias mensagens de erro e ao utilizar o Google para obter alguma informação, o que encontrávamos eram fóruns onde pessoas, que utilizavam o cliente, mostravam as mensagens de erro que obtiveram, e perguntavam o que poderia ter causado tal problema, o que em sua maioria eram causados por falhas nos servidores dos projetos. E nas raras vezes que encontrávamos alguma ajuda em relação à mensagem de erro, nem sempre ela era suficiente.

Depois de ter tudo instalado, vieram os problemas para conseguir fazer as coisas funcionarem como deviam. No começo não conseguíamos ligar o aplicativo de testes com as chaves de segurança, e quando conseguimos posteriormente tivemos problemas com o envio das tarefas, que não eram enviadas para nenhum computator.

Para resolver estes problemas adotamos a medida drástica de desinstalar tudo e instalar novamente, só que de uma forma diferente e fazendo configurações finas nos aplicativos e no BOINC em si. O que por sorte acabou funcionando, e ao final conseguimos fazer com que o BOINC funcionasse.

Todo este processo desde a instalação até conseguir distribuir as tarefas levou mais de um mês e meio. Possivelmente tivemos todos estes problemas porque nunca havíamos instalado MySql, PHP e Apache em nossas vidas, e sabemos que eles possuem inúmeras configurações e medidas de segurança que devem ser adotadas, mas causaram alguns transtornos durante a execução do nosso projeto.

5.2 Disciplinas relevantes

5.2.1 Alex

- **MAC0110 - Introdução à Computação**
  Esta matéria foi o meu primeiro contato com o mundo da programação.

- **MAC0122 - Princípios de Desenvolvimento de Algoritmos**
  Uma das matérias mais importantes do curso, pois é nela onde começamos a aprender algumas estruturas de dados, assim como começamos a nos preocupar com a eficiência de algoritmos. Acredito que também foi onde comecei a aprender e mexer mais no Linux.

- **MAC0211 - Laboratório de Programação I**
  Foi a primeira matéria onde tivemos um trabalho de grande porte, em grupo. Aprendi a importância em documentar e modularizar projetos, assim como pude aprender a mexer com o LaTeX.

- **MAC0431 - Introdução à Computação Paralela e Distribuída**
  O meu primeiro contato com a computação paralela e distribuída, e foi de grande importância para obter uma base de tais assuntos.

- **MAC0438 - Programação Concorrente**
  Também importante, aprendemos e treinamos técnicas de programação concorrente, assim como verificarmos os problemas que podem ocorrer.

- **MAC0412 - Organização de Computadores**
  Graças à monografia que tivemos que fazer nesta matéria, pude aprender sobre os limites da computação sequencial, e o quão importante a computação paralela e distribuída podem ser para vários projetos.

5.2.2 Hugo

- **MAC0110 - Introdução à Computação**
  Embora já tenha programado pouco em Java antes de entrar no BCC fazer MAC0110 ajudou a elucidar vários conceitos e a aprender a programar "de verdade".

- **MAC0122 - Princípios de Desenvolvimento de Algoritmos**
  MAC0122 foi uma das matérias mais importantes pois é nela que começamos a ver estruturas de dados e a como utilizar os recursos do computador de uma forma mais eficiente. E isto está em praticamente tudo o que envolve computação. Além do que foi a matéria que me "obrigou"a utilizar Linux e que fico muito agradecido por isso pois hoje utilizo o Linux como meu sistema operacional principal para fazer praticamente tudo.

- **MAC0211 e MAC0242 - Laboratório de Programação I e II**
  Ao fazer as matérias de laboratório de programação aprendemos muitos conceitos básicos que nos ajudam quando temos um projeto de porte maior que os tradicionais EPs, pois temos que planejar, fazer a documentação e controlar as versões de tudo o que é feito. Além de ter tido o primeiro contato com LaTeX, que mostrou-se uma ferramenta muito útil.

- **MAC0414 - Introdução à Computação Paralela e Distribuída**
  Atualmente os computadores são fabricados com processadores com mais de um núcleo e quis saber mais sobre quais são as vantagens e os problemas encontrados ao se programar para este tipo de processador, além de aprender sobre isto aprendi sobre programação distribuída - o que foi um motivador do nosso trabalho.

- **MAC0412 - Organização de Computadores**
  Organização de computadores foi muito útil pois foi nela que aprendemos mais sobre os conceitos gerais que envolvem a computação, desde partes de *hardware* até problemas encontrados atualmente, como os limites da computação sequencial e o avanço da programação paralela e distribuída - o que foi um dos motivadores do nosso trabalho.

5.3 Trabalho futuro

Um possível próximo passo seria transformar algum projeto existente que demande um grande poder computacional, e que possa ser distribuído utilizando a arquitetura mestre-escravo, para utilizar os computadores inativos de uma rede. Uma configuração que poderíamos tentar implementar para esta rede seria que enquanto um computador estivesse ligado mas sem nenhum usuário o utilizando, por exemplo quando o computador fica na tela de *login*, o cliente do BOINC executaria tarefas deste projeto. Isto seria muito útil em redes que precisam deixar os seus computadores ligados durante o dia inteiro mas em alguns horários, como por exemplo nas madrugadas, não têm muitos usuários ativos.

Referências

[1] http://www.distributed.net/

[2] http://setiathome.ssl.berkeley.edu/

[3] http://www.wired.com/science/discoveries/news/2000/06/37293

[4] http://findarticles.com/p/articles/mi_m0EIN/is_1995_Dec_4/ai_17812444/

[5] http://www.wired.com/wired/archive/3.12/toy.story.html

[6] http://www.mersenne.org

[7] http://web.media.mit.edu/~lfgs/

[8] http://groups.csail.mit.edu/cag/bayanihan/

[9] http://www.cs.huji.ac.il/~popcorn/

[10] http://cs.nyu.edu/milan/charlotte/frmain.html

[11] http://folding.stanford.edu/

[12] http://boinc.berkeley.edu/

[13] http://predictor.chem.lsa.umich.edu

[14] http://boincstats.com/stats/project_graph.php?pr=bo

[15] http://www.top500.org/lists/2008/11

[16] http://boinc.berkeley.edu/download.php

[17] http://einstein.phys.uwm.edu/

[18] http://www.worldcommunitygrid.org/

[19] http://www.boinc-wiki.info

[20] http://cpulimit.sourceforge.net/

[21] http://pirates.spy-hill.net/help/boinc-on-linux.html

[22] http://boinc.berkeley.edu/trac/wiki/CreateProjectCookbook

[23] http://myy.helia.fi/~karte/install_apache_on_ubuntu.html

[24] Abraham Silberschatz, Peter Galvin, Greg Gagne, *Operating System Concepts*. John Wiley & Sons, Inc., 7th Edition, 2005.