

Universidade Estadual de Maringá  
Centro de Tecnologia - Departamento de Informática  
Especialização em Desenvolvimento de Sistemas para *Web*

**Proposta de um Sistema WEB de Computação  
Voluntária para Problemas de Otimização  
Combinatória.**

**Diego Ramos de Bairros**

Marco Aurélio Lopes Barbosa  
Orientador:

Maringá/PR, 2008

Universidade Estadual de Maringá  
Centro de Tecnologia - Departamento de Informática  
Especialização em Desenvolvimento de Sistemas para *Web*

Diego Ramos de Bairros

**Proposta de um Sistema WEB de Computação  
Voluntária para Problemas de Otimização  
Combinatória.**

**Trabalho submetido à Universidade Estadual de Maringá  
como requisito para obtenção do título de Especialista  
em Desenvolvimento de Sistemas para Web.**

## AGRADECIIMENTOS

Os meus sinceros agradecimentos a todos que colaboraram para mais este passo em minha vida.

Muitíssimo obrigado:

- ü Deus, que me dá paciência e tranquilidade e sempre esta presente em minha vida;
- ü Meus pais que mesmo distantes sempre estão com seus pensamentos em seus filhos;
- ü Ana Carolina minha esposa e companheira, que sempre esta comigo em todos os momentos;
- ü Minha família e amigos pelas palavras de incentivo e apoio;
- ü Meu orientador pela paciência com esse orientando.

Obrigado a todos.

## RESUMO

Computação Voluntária ou Computação Filantrópica é o ramo da computação que visa à utilização do tempo ocioso de processamento dos computadores pessoais espalhados pelo planeta, como uma grade computacional mundial.

Para alcançar um nível de acesso e adesão mundial, leva-se muito tempo em pesquisa, desenvolvimento e distribuição, embora isso seja muito complexo temos hoje programas que, utilizando-se da *internet* buscam alcançar esse patamar de comunicação e adesão, como é o caso do BOINC (*Berkeley Open Infrastructure for Network Computing*), que se propõe a fazer a ligação entre projetos de computação e os usuários voluntários.

Como os problemas de otimização combinatória exigem muito processamento, a idéia é aproveitar, de uma forma inteligente, o tempo ocioso de processamento de qualquer máquina que esteja conectada à *internet* (usuários voluntários) de forma a termos, assim, ganho de poder computacional para ser aplicado à solução de problemas de otimização combinatória.

**Palavras Chaves:** Computação Voluntária, Otimização Combinatória, BOINC.

## ABSTRACT

Volunteer or Philanthropic Computation is the computational branch that aims the utilization of the personal computer idle processing time that is spread around the planet forming a powerful worldwide computational grid.

To achieve this access level it takes a hard research and development effort and time but nowadays we have programs that gathers this solution, as the case of BOINC (*Berkeley Open Infrastructure for Network Computing*) that works by doing the liason of the computational project and the volunteers.

As the Combinatory Optimization requires a very high complexity level and a great processing capacity, the idea is to take advantage, in a clever way, of the idle processing capacity of any machine connected to the internet (volunteer users) resulting in a free and great computational power directed toward the Combinatory Optimization problems solving.

## Índice

<b>RESUMO</b> .....	IV
ABSTRACT .....	V
Índice de Figuras .....	VII
Índice de Tabelas .....	VIII
Introdução .....	9
Objetivos .....	11
Justificativas .....	12
1 – Computação em Grade ( <i>Grid Computing</i> ) .....	13
1.1 - Computação Voluntária ou Filantrópica .....	15
1.2 - BOINC (Berkeley Open Infrastructure for Network Computing).....	16
<b>1.2.1 – Criando um Projeto BOINC</b> .....	21
1.2.1.1 - O que é um projeto BOINC? .....	21
1.2.1.2 - Ambientes de Execução e Versões de Software .....	22
1.2.1.3 – O que é uma <i>Workunit</i> ?.....	23
1.2.1.4 – Redundância Homogênea.....	25
1.3 – Questões de Segurança em Computação Voluntária .....	25
1.4 - Implementando um projeto .....	26
1.4.1 - Desenvolvendo a aplicação para o BOINC.....	27
1.4.2 – Gerar Unidades de Trabalho .....	29
1.4.3 – Etapa de Pós-Processamento.....	30
1.5 – Passos para implantar um projeto BOINC .....	30
2 – Computação Voluntária Via WEB .....	32
2.1 – O Servidor WEB .....	32
2.2 – Softwares de Controle Interno .....	33
2.3 – Montando a Aplicação.....	33
2.4 – Segurança.....	34
3 - Otimização Combinatória .....	36
3.1 – Otimização Combinatória e Computação Distribuída .....	37
Conclusão .....	39
Trabalhos Futuros .....	44
Referências .....	45

## Índice de Figuras

Figura 1: [2] Grid Computing .....	14
Figura 2: [7] Arquitectura BOINC .....	18
Figura 3: [8] Esquema Cliente/Servidor BOINC .....	20

## Índice de Tabelas

Tabela 1 - Tabela de Plataformas, retirada de <a href="http://boinc.berkeley.edu/trac/wiki/BoincPlatforms">http://boinc.berkeley.edu/trac/wiki/BoincPlatforms</a> , em 21/01/2008.....	22
Tabela 2 : Projetos mais ativos no BOINC.....	40
Tabela 3: Países mais ativos no BOINC.....	41
Tabela 4: Resumo de dados do projeto BOINC.....	42
Tabela 5: Tabela contendo os 5 (cinco) maiores computadores do mundo.....	42

## Introdução

Estima-se que os computadores pessoais, esses que temos em nossas casas e trabalho, permaneçam inativos cerca de 60% do tempo que estão ligados, isto é, estão ligados, mas não estão executando tarefa alguma, apenas consumindo energia. A computação voluntária, ou filantrópica, busca utilizar-se desse tempo computacional para outros fins, como por exemplo, projetos que realizam análises em cadeias de DNA buscando encontrar curas de doenças; ou outros tipos de projetos, como o que busca encontrar vida fora do planeta Terra, analisando ruídos captados por enormes antenas de rádio construídas para captar ondas vindas do espaço.

Enfim, a idéia é que qualquer pessoa possa colaborar processando partes do projeto, executando um pequeno pacote da computação necessária para analisar toda a informação; dessa forma, vários pequenos pacotes formam um grande pacote com as informações processadas. Para isso, uma complexa rede de compartilhamento e análise das informações se faz necessária, formando uma *grade computacional*, onde, máquinas de qualquer ponto do planeta executem pequenos pacotes de determinada tarefa, somando assim seus processamentos.

Temos, dentro da computação, várias áreas que necessitam de grande poder computacional, principalmente àquelas ligadas ao cálculo de probabilidades e soluções matemáticas, pois checar as inúmeras possibilidades de uma equação é algo que consome muitos recursos computacionais. Montar rotas para uma melhor logística, prever um possível terremoto ou uma tempestade, são tarefas que demandam anos em processamento e simulações, portanto quanto mais rápido for possível realizar essas simulações, melhores serão os resultados; portanto, a busca por melhores formas de se ganhar tempo de processamento é algo sempre buscado e almejado na computação e a computação em grade é uma das alternativas para se conseguir ganho de processamento.

Uma alternativa seria a compra de supercomputadores, mas estes são extremamente caros, impossibilitando, na maioria dos casos, sua aquisição, principalmente para instituições de ensino público, ou mesmo empresas privadas de pequeno e médio porte.

Para um efetivo ganho de desempenho utilizando-se computação em grade é necessário que vários usuários compartilhem seus ciclos de processamento, e uma forma de

conseguir uma maior adesão de usuários é que a participação ocorra de forma simples, uma das propostas do projeto consiste em desenvolver uma interface *web*<sup>1</sup> para a prática e participação dos trabalhos, de forma que o voluntário possa participar rapidamente de um dos projetos.

---

<sup>1</sup> Já é tido como um sinônimo de *www* ou *World Wide Web*, que significa “Rede de Alcance Mundial”, ou mais conhecida como *internet* – A rede mundial de computadores.

## Objetivos

O objetivo desse trabalho é apresentar como a computação voluntária funciona e pode ser de grande importância e muito bem aproveitada pela comunidade acadêmica e científica, conseguindo aumento no poder computacional de instituições que geralmente não possuem disponibilidade de recursos para a aquisição de grandes computadores. Os objetivos específicos são:

- Pesquisar sobre o software de computação voluntária BOINC.
- Propor um sistema de computação voluntária, utilizando uma interface que seja acessível através da *web*, para resolver problemas de otimização combinatória.

## Justificativas

Os programas existentes hoje para a prática da computação voluntária, são sistemas que necessitam ser instalados nas máquinas clientes, o que demanda tempo e conhecimento do usuário, a idéia é que o usuário apenas se identifique com usuário e senha, após um breve cadastro (apenas com os dados essenciais), e já possa participar de algum projeto.

Para que isso ocorra deve ser utilizada uma tecnologia *web* que execute funções e procedimentos diretamente na máquina do usuário, um *applet*<sup>2</sup> *Java*. Mais detalhes sobre a tecnologia *java* e o *applet java*, serão dados nos capítulos seguintes.

Divulgar e expor as idéias da computação voluntária para que sejam usadas para em pesquisas que demandam de muito processamento, mas que não possuem recursos financeiros disponíveis para adquirir um supercomputador.

Um dos problemas da computação voluntária é conseguir a adesão de usuários, os voluntários, a proposta é de um sistema que facilite a adesão desses voluntários aos projetos disponíveis. Além da busca por ganho de tempo e poder de processamento, que são de fundamental importância para o avanço da ciência e tecnologia.

---

<sup>2</sup> É um aplicativo que é executado dentro de outro programa, em nosso caso o *applet* será executado dentro de um navegador para internet.

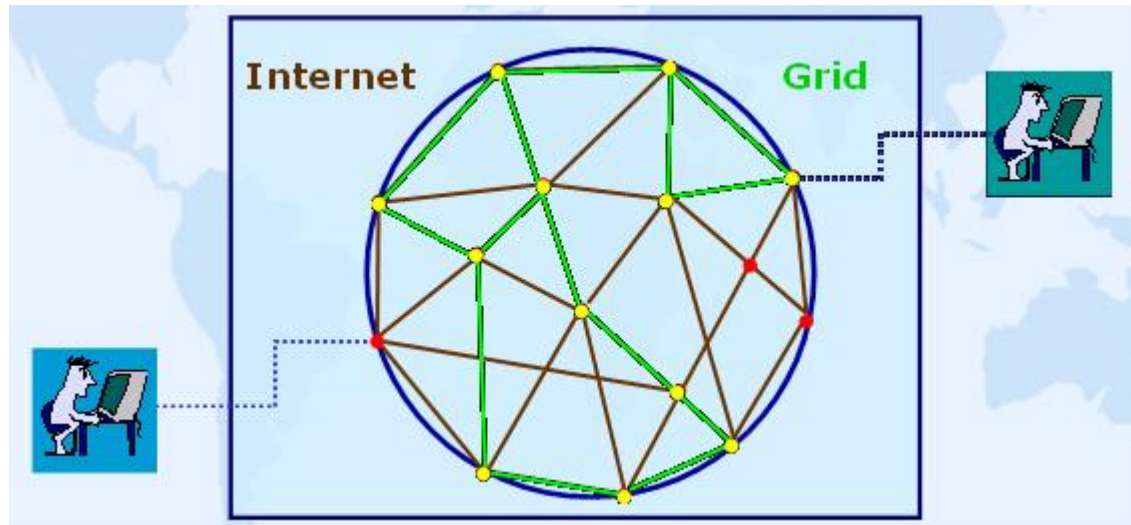
## 1 – Computação em Grade (*Grid Computing*)

Segundo Nazário *apud* Buyya, *Grid computing* “é um tipo de sistema paralelo e distribuído que permite o compartilhamento, seleção e agregação de recursos distribuídos em múltiplos domínios *administrativos*, dependentes de suas disponibilidades, capacidades, performances, custos, e qualidade de serviço requerida por usuários”.

Na Figura 1, temos uma ilustração de como funciona uma grade computacional, usuários distantes podem participar de um mesmo projeto, desde que, estejam conectados a mesma rede, que pode ser uma rede local ou mesmo a *internet*.

As tarefas (partes) que serão distribuídas devem seguir um critério, de forma a garantir o máximo de utilização dos recursos disponíveis no GRID, por isso uma máquina que possui um maior poder de processamento, provavelmente irá receber um grupo maior de instruções para resolver (balanço de carga).

Provavelmente, pois outros fatores influenciam no balanço de carga, como por exemplo, disponibilidade, um computador pode ter um grande poder de processamento, mas ficar pouco tempo conectado a *internet*, portanto não irá comunicar-se com o GRID e assim, não receberá outras instruções. Com essa visão, temos que um GRID não possui um escalonador de recursos, já que a complexidade de uma grade computacional é muito grande (Grande Escala, ampla distribuição e existência de múltiplos domínios administrativos), mas possui um escalonador de aplicação, que necessita conhecer detalhes da aplicação e saber quanto tempo cada recurso vai levar para realizar uma tarefa, para então saber como dividir e distribuir os recursos.



**Figura 1:** [2] Grid Computing

Podemos agrupar as tecnologias que fazem parte de uma grade computacional em quatro grandes partes, sendo elas:

- A camada de aplicações, são os portais de acesso e os programas que buscam aproveitar as potencialidades do GRID;
- Ambiente para os usuários, é o ambiente de programação em grade, utilizados por aqueles que desejam submeter algum processamento para ser realizado através de computadores em grade;
- *Middleware* são os serviços utilizados na administração dos recursos a serem utilizados e distribuídos;
- Estrutura básica, computadores pessoais, servidores, sistemas operacionais, bancos de dados.

Podemos citar um exemplo de GRID aqui mesmo no Brasil: é o GridRio, uma iniciativa do estado do Rio de Janeiro, que liga a UFF (Universidade Federal Fluminense), PUC-Rio (Pontifícia Universidade Católica do Rio de Janeiro), LNCC (Laboratório Nacional de Computação Científica) e o CBPF (Centro Brasileiro de Pesquisas Científicas). Este GRID liga os *Clusters*<sup>3</sup> destes quatro institutos, colaborando assim na pesquisa de todos, este é um “GRID fechado”, pois para fazer parte dele são necessários vários requisitos.

<sup>3</sup> É um conjunto de computadores, muitas vezes computadores pessoais, ligados em rede de forma a trabalharem como se fossem um único equipamento.

Existem outros tipos de GRID, como o que a arquitetura BOINC (*Berkeley Open Infrastructure for Network Computing*) sugere e aplica, ele é baseado no modelo *open source*<sup>4</sup>. Analisaremos o BOINC com mais detalhes no capítulo 1.2.

## 1.1 - Computação Voluntária ou Filantrópica

Computação voluntária ou computação filantrópica ocorre quando usuários de computadores doam ciclos de processador, quando estes estão ociosos, para realizar trabalhos para determinada organização, a fim de colaborar na resolução de cálculos muito complexos, seja para procurar vida extraterrestre (*SETI@Home*), ou para cálculos de dobradura de proteínas (*Folding@home*), entre outros.

Isso significa que temos uma rede distribuída de computadores trabalhando para um mesmo objetivo, podendo alcançar um poder computacional que pode ultrapassar a barreira do *Petaflop*<sup>5</sup>. Veja que estamos pensando em computação voluntária e não comercial, se olharmos pelo prisma da área comercial, temos dois fatores preponderantes, segurança e disponibilidade, itens que não temos, com total controle ou sanções, na computação voluntária, tendo sempre em mente que estamos falando de computadores pessoais, que estão sujeitos a reinicializações e demais inconvenientes caseiros (falta de energia, cachorro, *logoff*<sup>6</sup>, desligamento, etc), além do mais, temos a questão da conexão que é lenta, pois estamos tratando de uma conexão pela *internet*, que em países como o Brasil ainda está longe de ser um serviço de alta qualidade.

A idéia da computação voluntária ficou conhecida no mundo através do projeto *Seti@home*, que é parte do projeto *Seti (Search for Extra-Terrestrial Intelligence)* norte-americano, que busca sinais de vida extraterrestre através de sinais de rádio captados por instrumentos como o radiotelescópio de Arecibo, em Porto Rico, mais de 3 milhões de voluntários já instalaram o programa desde 1999.

A idéia é simples, o usuário (voluntário), faz o cadastro no site e faz o *download* do *software* cliente para sua máquina, este *software* é quem se comunicará com o servidor

---

<sup>4</sup> “Código aberto”, tipo de *software* ou estrutura onde o código fonte é visível publicamente.

<sup>5</sup> *Flop* é um acrônimo para uma operação de ponto flutuante. Um *MegaFlop* é a medida de um milhão de *Flops*, um *GigaFlop* 1000 *Flops*, um *TeraFlop* 1000 *GigaFlops* e 1 *PetaFlop* representa 1000 *TeraFlops*, ou seja, um quintilhão de operações em ponto flutuante.

<sup>6</sup> Término de uso por um usuário, pode ser tratado como uma reinicialização rápida, já que todos os programas que estavam em execução no referido usuário serão fechados.

onde estão os dados do projeto, arquivos, banco de dados, e também controla os ciclos de uso do computador automaticamente, o software não toma parte do processamento do equipamento enquanto outros sistemas estiverem em uso.

A computação voluntária pode trazer vários avanços para a área científica, tornando mais ágil o processo de obtenção de resultados, principalmente para as instituições de ensino e laboratórios, que se vêm sempre com dificuldade de conseguir recursos para a compra de grandes equipamentos, com a distribuição da carga para voluntários utilizando uma arquitetura de grade computacional, pode-se tornar viável, para essas instituições, a pesquisa científica com maior grau de complexidade.

Segundo [3], a computação voluntária é importante por vários motivos:

- § Devido ao grande número de microcomputadores existentes no mundo e ao crescimento desse número, permitindo para a comunidade científica alcançar um grande poder computacional, que não poderia ser conseguido de outra forma;
- § A computação voluntária não é comprada, ela deve ser doada pelos voluntários. Projetos científicos que possuem pouco recurso, mas um grande apelo popular, podem conseguir um grande poder computacional;
- § Incentiva o interesse público para a ciência, proporcionando ao público voz na direção das investigações científicas.

## **1.2 - BOINC (*Berkeley Open Infrastructure for Network Computing*)**

O BOINC é um *software* desenvolvido pela Universidade de Berkeley nos Estados Unidos, ele é uma Infra-estrutura aberta para processamento distribuído. Com o BOINC o usuário tem a possibilidade de participar de vários projetos de computação distribuída e o controle para dividir os ciclos de processador entre os projetos em que estiver cadastrado. Podemos dizer que o BOINC é uma plataforma para desenvolvimento de projetos de computação distribuída.

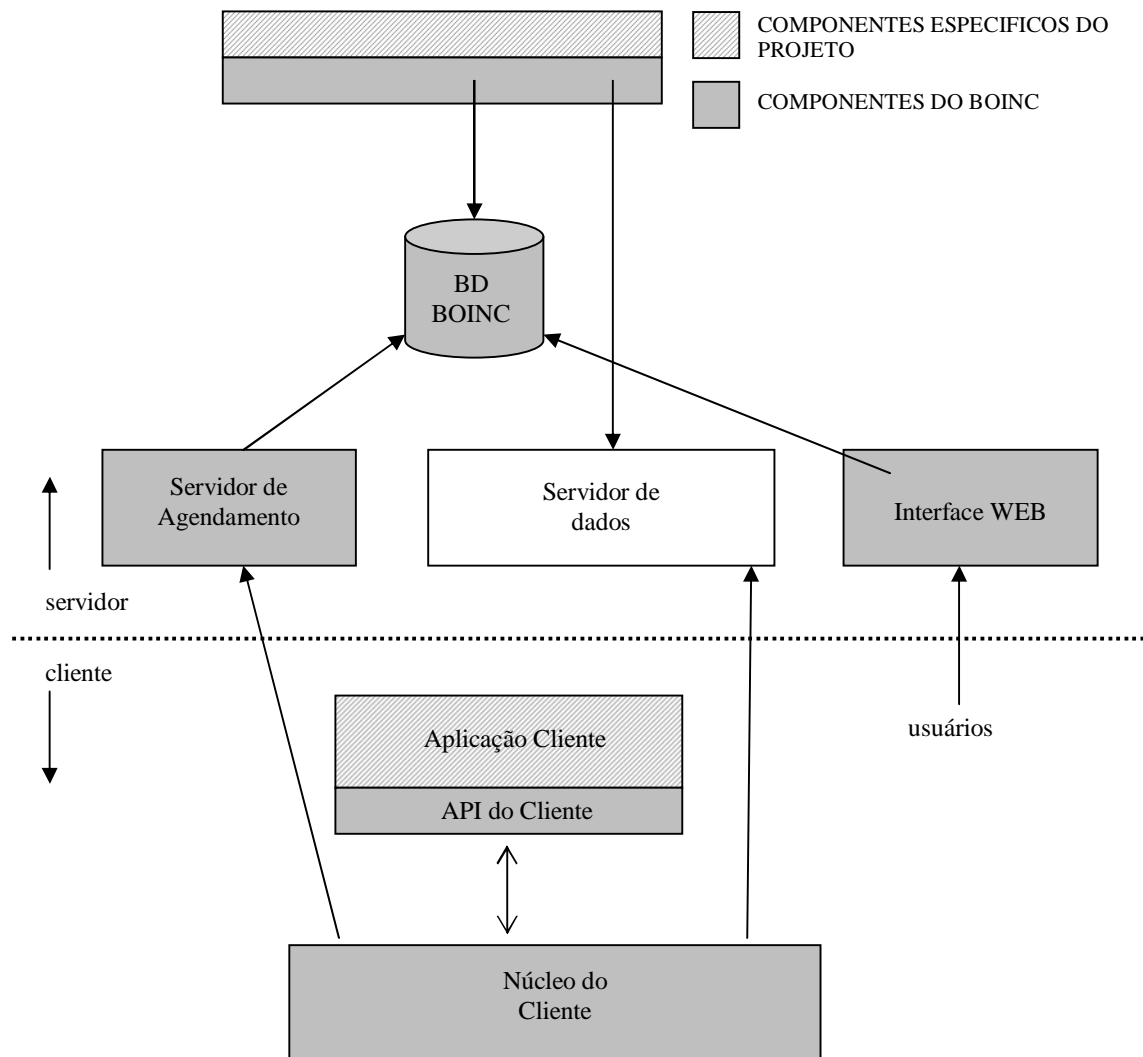
A arquitetura do BOINC é cliente/servidor. Temos um servidor, ou servidores, que é responsável por controlar os usuários, agrupar os dados dos projetos, que são as *unidades*

*de trabalho*<sup>7</sup> disponíveis, enviadas e processadas, ele é responsável pela distribuição dos arquivos (unidades de trabalho) e do recebimento das respostas, bem como do escalonamento dos trabalhos para os usuários, de acordo com a produtividade de cada um. No lado cliente, temos um software que é comum a todos os projetos, com as interfaces de comunicação com o servidor, e os códigos dos projetos em que o usuário estiver participando, o cliente BOINC carrega o programa específico do projeto de acordo com as configurações encontradas no arquivo de definição do projeto, por exemplo, para o projeto rosetta@home, o programa carregado pelo BOINC para processar as unidades de trabalho é o “*rosetta\_beta\_5.93\_windows\_intelx86.exe*”, isso para um computador com sistema operacional *windows*, arquitetura Intel/x86 e com a versão disponível do software rosetta o *rosetta\_beta\_5.93*.

Na **Figura 2**, é apresentada a arquitetura do sistema BOINC.

---

<sup>7</sup> São os arquivos, dados, informações que deverão ser processadas pelos usuários, sempre referentes a cada um dos projetos que o voluntário estiver participando.



**Figura 2:** [7] Arquitetura BOINC

Segundo Taurion [4], as principais características do BOINC são:

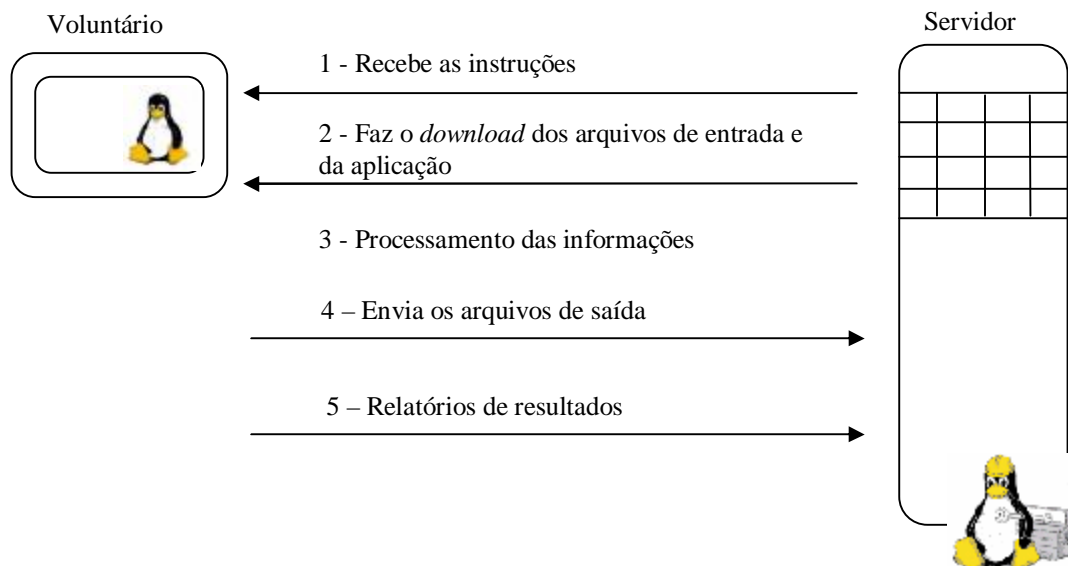
- Permitir a operação de aplicações já existentes, escritas em linguagens como Fortran, C e C++, com pouca ou nenhuma modificação;
- Oferecer um maior grau de segurança e proteção contra vírus, inclusive com uso de assinaturas digitais baseadas em criptografia de chaves públicas;
- Permitir o uso de vários servidores. O programa cliente ao identificar que o servidor principal está fora do ar, tentará acesso a servidores alternativos;
- Código fonte aberto, com a restrição de não poder ser utilizado em produtos comerciais. Os projetos que executam sob o BOINC precisam ter seu código fonte aberto;

Ainda segundo Taurion [4], como o BOINC (e todos os sistemas de computação voluntária) trabalha enviando dados aos clientes e recebendo suas respostas, algumas situações podem acontecer, durante este percurso.

- a) O programa cliente executa corretamente a computação e retorna os resultados;
- b) O programa cliente computa erroneamente seus algoritmos e envia resultados errados ao servidor;
- c) O usuário não consegue estabelecer conexão com o servidor e, portanto, não consegue executar *downloads* ou *uploads*;
- d) A aplicação não funciona na máquina do usuário;
- e) O usuário nunca retorna o resultado de sua computação, inclusive por desistir do processo no meio do caminho;

O BOINC trabalha com um sistema de redundância de informações de forma a garantir a exatidão dos resultados, um mesmo pacote é enviado a mais de um colaborador, e seus dados somente são confirmados após encontrar consenso entre as partes que executaram a tarefa, se um consenso não for alcançado, outros clientes são chamados para executar a tarefa até que se chegue a uma definição. Essas definições de redundância são configuradas nos servidores e é variável para cada projeto, cada um deve especificar o quorum mínimo de respostas consensuais para que os dados sejam gravados no banco de dados de resultados confirmados.

A idéia de redundância é muito válida nesse ambiente, tendo em vista a quantidade de possibilidades de uma unidade de trabalho não ser executada de forma correta ou nem chegar a ser executada, o ônus disso é o tempo de processamento gasto para uma mesma unidade de trabalho, sendo assim, as unidades de trabalho devem ser bem ranqueadas de forma a tentar evitar o desperdício de processamento com unidades de trabalho mais simples, mas não menos importantes é claro. O esquema de interações entre o cliente BOINC e o servidor BOINC pode ser visto na **Figura 3**.



**Figura 3:** [8] Esquema Cliente/Servidor BOINC

O BOINC trabalha ainda com conjuntos de chaves criptográficas para garantir o recebimento do resultado verdadeiro, buscando com isso diminuir o número de fraudes nos resultados.

Algumas características devem ser analisadas para submeter um projeto ao BOINC, (lembrando que na computação voluntária a conexão com a internet é fundamental e imprescindível), a aplicação deve atrair um grande público, possuir um forte apelo popular, motivando assim o voluntariado. A aplicação deve ser paralelizável, portanto deve ser possível parti-la em pequenos pacotes, com pouca dependência entre elas. Deve ter um sistema de tolerância à falhas, como não podemos considerar os resultados da computação voluntária 100% corretos, deve haver um mecanismo de redundância para minimizar erros, se a aplicação necessitar de resultados 100% corretos a computação voluntária pode não ser o melhor caminho a seguir.

## 1.2.1 – Criando um Projeto BOINC

### 1.2.1.1 - O que é um projeto BOINC?

Podemos dizer que um projeto BOINC é o conjunto de uma ou mais aplicações, que podem ser distribuídas, organizadas e mantidas por uma empresa, governo, ou organização, e cada projeto é totalmente independente do outro, com suas bases de dados, seus servidores e suas regras, não interferindo em outros projetos. Por exemplo, o projeto *seti@home* e o projeto *rosetta@home*, são projetos totalmente distintos, tanto na questão científica quanto em suas equipes e localização, mas ambos utilizam o BOINC como base, como interface com seus colaboradores e um voluntário pode fazer parte de ambos projetos tendo instalado o BOINC em sua máquina.

Vemos claramente que o BOINC fornece os mecanismos para que uma aplicação com aspiração distribuída possa alcançar seus objetivos, que são: conseguir voluntários para executá-la e por conseqüência aumentar o seu poder computacional com os ciclos doados pelos voluntários.

Já foi dito que o BOINC utiliza uma estrutura cliente/servidor, tendo em vista essa arquitetura podemos enxergar que do lado servidor o BOINC divide-se, basicamente, em duas partes, a primeira responsável por disponibilizar os recursos do projeto para os voluntários (serviços, aplicativos e manipulação dos dados computacionais), e a segunda, responsável por distribuir as unidades de trabalho e realizar o recolhimento dos dados processados.

A estrutura de armazenamento do BOINC é baseada em arquivos (unidades de trabalho, arquivos de resultado, executáveis, bibliotecas e etc), portanto é de alta complexidade e de missão crítica o tratamento desses arquivos na arquitetura BOINC. Todas as transferências de arquivos são realizadas utilizando o protocolo ‘http’, tanto o envio (*upload*) dos arquivos quanto o seu recebimento (*download*).

### 1.2.1.2 - Ambientes de Execução e Versões de Software

Em uma aplicação distribuída, no estilo do BOINC, os computadores onde serão executadas as instruções possuem as mais diversas plataformas operacionais, *windows*, *linux*, *Mac Os*, *Unix*, etc. Temos ainda máquinas com múltiplos processadores, e outras diferenças referentes a arquitetura do equipamento. Quando um projeto for disponibilizado, este deve informar para quais plataformas ele esta habilitado a ser executado.

Veja abaixo na Tabela 1, uma lista de plataformas suportadas pelo BOINC, isso não significa que todos os projetos possuem suporte a estas arquiteturas, como foi dito acima, cada desenvolvedor deve portar sua aplicação para as arquiteturas que lhe forem convenientes.

Tabela 1 - Tabela de Plataformas, retirada de <http://boinc.berkeley.edu/trac/wiki/BoincPlatforms>, em 21/01/2008

Plataforma	Descrição
<b>Windows_intelx86</b>	Microsoft Windows (98 ou posterior) sendo executado em uma CPU compatível com intel x86
<b>Windows_x86_64</b>	Microsoft Windows (98 ou posterior) sendo executado em um AMD x86_64 ou intel EM64T CPU
<b>i686-pc-linux-gnu</b>	Linux sendo executado em um CPU compatível com Intel x86
<b>x86_64-pc-linux-gnu</b>	Linux sendo executado em um AMD x86_64 ou Intel EM64T CPU
<b>powerpc-linux-gnu</b>	Linux sendo executado em um processador 32-bit PowerPC
<b>ppc64-linux-gnu</b>	Linux sendo executado em um processador 64-bit Powerpc
<b>alpha-hp-linux-gnu</b>	Linux sendo executado em um Alpha
<b>ia64-linux-gnu</b>	Linux sendo executado em um IA64 (Itanium)
<b>sparc-sun-linux-gnu</b>	Linux sendo executado em um SPARC
<b>sparc64-sun-linux-gnu</b>	Linux sendo executado em um SPARC 64-bit
<b>powerpc-apple-darwin</b>	Mac OS X 10.3 ou posterior sendo executado em um Motorola PowerPC
<b>i686-apple-darwin</b>	Mac OS 10.4+ sendo executado em um Intel CPU
<b>x86_64-apple-darwin</b>	Mac OS 10.5+ sendo executado em um Intel 64-bit CPU
<b>sparc-sun-solaris2.7</b>	Solaris 2.7 sendo executado em uma CPU compatível com SPARC
<b>sparc-sun-solaris</b>	Solaris 2.8+ sendo executado em uma CPU compatível com SPARC
<b>sparc64-sun-solaris</b>	Solaris 2.8+ sendo executado em um CPU SPARC 64-bit
<b>hppa-hp-hpux</b>	HPUX sendo executado em um 32-bit HPPA

<b>hppa64-hp-hpux</b>	HPUX sendo executado em um 64-bit HPPA
<b>alpha-hp-tru64</b>	Tru64 Unix sendo executado em um Alpha
<b>ia64-hp-hpux</b>	HPUX sendo executado em um IA64
<b>powerpc-ibm-aix</b>	AIX sendo executado em um PowerPC
<b>i686-pc-freebsd</b>	FreeBSD com arquitetura x86
<b>x86_64-pc-freebsd</b>	FreeBSD com arquitetura compatível com Intel 64-bit
<b>i686-pc-openbsd</b>	OpenBSD com arquitetura x86
<b>x86_64-pc-openbsd</b>	OpenBSD com arquitetura compatível com Intel 64-bit

Um projeto pode gerenciar várias aplicações que por sua vez gerenciam seus programas, geralmente mais de um programa por aplicação, visto que para cada plataforma temos um programa, e também as suas unidades de trabalho (*workunits*). Lembrando que, apesar do BOINC possuir versões para essas plataformas, não significa que a aplicação possuirá o suporte nas mesmas, para isso o desenvolvedor deverá criar um aplicativo para cada plataforma, pois em cada uma delas existem diferenças nos resultados em ponto flutuante, que deverão ser tratados para que os resultados estejam corretos.

### 1.2.1.3 – O que é uma *Workunit*?

*Workunit* significa unidade de trabalho, e é exatamente isso que este arquivo é, nele estão contidos os cálculos que devem ser executados. Cada unidade de trabalho é indicada por uma linha no banco de dados da aplicação. O BOINC, fornece funções em C e um aplicativo (que é um *script*), para ajudar na criação das unidades de trabalho, e são nas unidades de trabalho que está grande parte das configurações de funcionamento do BOINC, por exemplo, o controle de quorum, indicando quantos resultados deverão ser informados para que a unidade de trabalho seja considerada executada. O preenchimento dos parâmetros, da unidade de trabalho, deverá ser preenchido com total segurança e certeza, pois são eles que controlam como o BOINC deve comportar-se na execução e processamento da unidade de trabalho.

Um dos pontos positivos da arquitetura BOINC é o seu controle de redundância contra erros, a arquitetura funciona com o servidor enviando as tarefas aos clientes e estes processando e retornando os resultados, mas durante esse processo podem ocorrer algumas situações que devem e são tratadas pelo BOINC, vejamos: podem ocorrer casos onde o

cliente recebe a tarefa processa e envia os resultados com erro, ou casos ainda onde, o cliente não consegue receber nem retornar tarefas, ou simplesmente pega a tarefa e remove o sistema de sua máquina, portanto nunca retornando o resultado. Para tratar casos como esse é que o BOINC possui um controle de redundância, onde cada tarefa é executada em mais de um voluntário, dessa forma os resultados devem entrar em um consenso, caso contrário, uma nova tarefa poderá ser enviada para obter esse resultado.

Um ponto importante a ser destacado é que o BOINC realiza o tratamento dos dados com pouca intervenção humana, todas as manipulações são realizadas por programas que ficam nos servidores, responsáveis por realizarem as tarefas, como por exemplo, limpar do servidor de arquivos as unidade de trabalho já processadas, calcular o valor de crédito a ser dado para os voluntários que completaram a tarefa, entre outros, mas existem dois pontos onde o desenvolvedor da aplicação é envolvido, são eles: Validação e Equiparação.

**Validação:** Quando um número suficiente de resultados positivos é devolvido, o BOINC compara-os e checa se existe um “consenso” entre eles, é nesse ponto que o desenvolvedor “entra”, não que seja ele que irá analisar os arquivos manualmente, mas sim implementar a lógica de validade dos arquivos. Faz parte da aplicação dizer quando um resultado é considerado como consenso (regras e especificações de plataforma devem ser tratadas pelo desenvolvedor), uma vez que essas regras são cumpridas, esse resultado é salvo na tabela de resultados e este é tido como resultado definitivo, voluntários que enviarem seus resultados (para essa unidade de trabalho), após o resultado definitivo ser encontrado, receberão os créditos desde que seu resultado seja igual ao resultado definitivo.

**Equiparação:** é o mecanismo através do qual o projeto é informado da conclusão (sucesso ou insucesso), de uma unidade de trabalho, é executado apenas uma vez para cada unidade de trabalho, se ocorreu sucesso o BOINC lê o arquivo de saída e manipula as informações, gravando, por exemplo, no banco de dados do projeto, caso ocorra um insucesso, o sistema poderá enviar um e-mail, ou realizar qualquer outro procedimento implementado pelo desenvolvedor para a aplicação.

#### 1.2.1.4 – Redundância Homogênea

Outro recurso interessante do BOINC, para o tratamento dos resultados, é a Redundância Homogênea, além de poder enviar uma unidade de trabalho para vários voluntários, a fim de encontrar um consenso, o BOINC ainda classifica os voluntários de acordo com os resultados retornados. Como um resultado depende de alguns fatores, como: arquitetura, sistema operacional, parâmetros e compilador utilizado, o BOINC classifica os computadores através de uma equivalência numérica de classes, onde dois computadores estão na mesma classe numérica caso retornem resultados idênticos para suas aplicações, sendo assim, o BOINC, envia uma unidade de trabalho apenas para máquinas com a mesma classe numérica, permitindo assim um maior rigor nos resultados.

### 1.3 – Questões de Segurança em Computação Voluntária

Aos trabalharmos com computação distribuída, devemos ter muito cuidado com o quesito segurança, não só a segurança da aplicação e seus resultados, mas também a segurança dos voluntários, dados pessoais referentes a cadastro, ou mesmo algum software mal intencionado ser distribuído como sendo parte do projeto.

No próprio site do projeto BOINC [3], encontramos as tentativas de ataque mais comuns utilizadas, por *hackers*<sup>8</sup> ou mesmo por usuários mal-intencionados:

- Retorno de Resultados Incorretos, usuários mal intencionados modificam os arquivos de resultados na intenção de causarem discrepâncias nos resultados encontrados;
- *Hackers* podem tentar invadir o servidor BOINC da aplicação a fim de modificarem o banco de dados e distribuírem seus próprios executáveis, vírus, etc.;
- Usuários retornam resultados alegando mais tempo de CPU do que foi realmente utilizado;

---

<sup>8</sup> Indivíduo que aplica o seu conhecimento e inteligência para conseguir um resultado inteligente uma melhora, para invasores de computador ou usuários mal intencionados o termo mais utilizado hoje é o *cracker*, mas ainda hoje o termo *hacker* ainda é visto com alguém com más intenções.

- *Hackers* podem tentar causar um *crash* no servidor enviando muitos arquivos de grande tamanho, superlotando os discos, ficando o servidor inutilizado;
- Outro ataque comum é a tentativa de se conseguir informações sobre os usuários através de ataques aos protocolos de envio e recebimento de dados

## 1.4 - Implementando um projeto

Para implementarmos um projeto, podemos utilizar uma ferramenta que vem junto com o BOINC, chamada *make\_project*, presente na pasta “*tools*” no servidor onde foi instalado o BOINC.

A sintaxe do script *make\_porject* segue abaixo:

```
tools/make_project
  --project_root <path>
  --db_user <database_user>
  --db_password <database_password>
  --key_dir <key_directory>
  --url_base <url_base> <nome_curto> <nome_longo>
```

Efetivamente o que o script faz é:

- Cria o projeto com seus diretórios e subdiretórios;
- Cria as chaves de criptografia;
- Cria e inicializa o banco de dados;
- Copia fontes e arquivos executáveis;
- Gera o arquivo de configuração do projeto.

Ao término da execução do script, o arquivo *nome\_projeto.http.conf* deverá ser copiado para a pasta “*/etc/apache/http.conf*”, dessa forma estará disponível aos usuários a página do projeto, que foi criada pelo *script*, onde os usuários poderão se cadastrar e ver os resultados do projeto. Será disponibilizado também um outro endereço com o mesmo nome acrescido de “*\_ops*”, este endereço possui uma interface administrativa para manutenção do projeto, portanto ele deve ser protegido contra usuários não autorizados.

Após a criação do projeto devemos configurar o arquivo “project.xml”, para adicionar as aplicações e as plataformas disponíveis para o projeto. Podemos ver um exemplo da estrutura do arquivo logo abaixo:

```
<boinc>
  <platform>
    <name>i686-pc-linux-gnu</name>
    <user_friendly_name>Linux running on an Intel x86-compatible
    CPU</user_friendly_name>
  </platform>
  <platform>
    <name>windows_intelx86</name>
    <user_friendly_name>Windows /x86</user_friendly_name>
  </platform>
  <app>
    <name>aplicacao</name>
    <user_friendly_name>Aplicação Principal</user_friendly_name>
  </app>
</boinc>
```

Após realizarmos as modificações no arquivo devemos utilizar um outro script disponível no BOINC, o *xadd*, presente no diretório *bin* do servidor, este script realiza o envio das informações do arquivo xml para o banco de dados.

### 1.4.1 - Desenvolvendo a aplicação para o BOINC

Para que a aplicação possa ser executada dentro do BOINC alguns ajustes devem ser realizados na aplicação de forma a deixá-la preparada para a estrutura do BOINC. Para isso o BOINC disponibiliza aos desenvolvedores algumas APIs de desenvolvimento de forma a facilitar o trabalho de portabilidade, as APIs disponíveis são:

**Basic API:** Nesta API estão disponíveis os comandos básicos para se executar uma aplicação no ambiente BOINC, os principais são:

- *int boinc\_init()*: comando responsável por iniciar o BOINC;
- *int boinc\_finish(int status)*: finaliza a comunicação com o BOINC, onde, nesse momento a unidade de trabalho é liberada para *upload*;
- *int boinc\_resolve\_filename(char \*logical name, char \*physical name, int len)*: retorna o nome físico do arquivo, mas para isso deve-se passar o nome lógico;
- *FILE\* boinc\_fopen(char\* path, char\* mode)*: Realiza abertura de arquivos;
- *int boinc\_time\_to\_checkpoint()*: Verifica se esta na hora de fazer o *checkpoint*;
- *void boinc\_checkpoint\_completed()*: Avisa todo o sistema que o *checkpoint* esta completo;
- *void boinc\_ops\_per\_cpu\_second(double floating point ops, double integer ops)*: Retorna a quantidade de operações sobre inteiros e sobre ponto flutuante que o computador pode realizar;
- *void boinc\_ops\_cumulative(double floating point ops, double integer ops)*: Retorna a quantidade de operações sobre inteiros e sobre ponto flutuante que a aplicação realizou;

**Diagnostics API:** Responsável pelo diagnóstico da aplicação;

**Graphics API:** API que permite a criação de proteção de telas e gráficos para a aplicação;

**Trickle messages API:** Responsável pelas trocas de mensagens entre a aplicação e o servidor;

**Intermediate upload API:** Responsável por fazer *uploads* intermediários, isto é, antes do término da unidade de trabalho.

## 1.4.2 – Gerar Unidades de Trabalho

Para criarmos tarefas para o BOINC devemos utilizar o *script create\_work*, localizado no diretório bin, o *script* necessita que sejam passados alguns parâmetros para a sua correta execução, são eles o nome da unidade de trabalho, nome do arquivo que será enviado para processamento, e também dois *templates*, um para arquivos de saída e outro para arquivos de entrada.

Abaixo, exemplo de um *template* para arquivo de entrada:

```
<file_info>
  <number>0</number>
  [ <sticky/>, other attributes ]
</file_info>
[ ... ]
<workunit>
  <file_ref>
    <file_number>0</file_number>
    <open_name>Nome do arquivo</open_name>
  </file_ref>
  [ ... ]
  [ <command_line>-parâmetros</command_line> ]
  [ <rsc_fpop_est>x</rsc_fpop_est> ]
  [ <rsc_fpop_bound>x</rsc_fpop_bound> ]
  [ <rsc_memory_bound>x</rsc_memory_bound> ]
  [ <rsc_disk_bound>x</rsc_disk_bound> ]
  [ <delay_bound>x</delay_bound> ]
  [ <min_quorum>x</min_quorum> ]
  [ <target_nresults>x</target_nresults> ]
  [ <max_error_results>x</max_error_results> ]
  [ <max_total_results>x</max_total_results> ]
  [ <max_success_results>x</max_success_results> ]
  [ <credit>X</credit> ]
</workunit>
```

É no arquivo de entrada que temos as configurações para o processamento da unidade de trabalho, por exemplo, o quorum necessário para a aceitação do resultado, tempo máximo de processamento, total de créditos (que serão dados ao usuário após passar pelo validador, através de uma função especial), entre outros.

Abaixo, exemplo de um *template* de arquivo de saída, resultado:

```
<file_info>
```

```

<name><ARQSAIDA_0/></name>
<generated_locally/>
<upload_when_present/>
<max_nbytes>32768</max_nbytes>
<url><UPLOAD_URL/></url>
</file_info>
<result>
  <file_ref>
    <file_name><ARQSAIDA_0/></file_name>
    <open_name>result.sah</open_name>
  </file_ref>
</result>

```

A sintaxe do comando para a criação das unidades de trabalho é a que segue:

```
create work -appname name -wu name name -wu template filename -result template
filename infile
```

### 1.4.3 – Etapa de Pós-Processamento

Após a submissão e processamento das unidades de trabalho existe a fase chamada de pós-processamento que possui dois módulos, o validador e o assimilador de resultados.

**Validador de Resultados:** ele é o responsável por realizar as comparações entre os resultados redundantes e decidir quais são os resultados corretos, além disso, decide o valor dos créditos que serão dados aos voluntários.

**Assimilador de Resultados:** é o mecanismo responsável por notificar o projeto, da conclusão de uma unidade de trabalho, é realizado uma única vez por unidade de trabalho, sendo esta concluída com sucesso, organiza as informações fornecidas e realiza a gravação no banco de dados, se o resultado for falho, pode por exemplo, enviar um e-mail de alerta ao administrador do projeto.

## 1.5 – Passos para implantar um projeto BOINC

1. Baixar os arquivos de código fonte atualizados do BOINC;

2. Instalar um servidor *web*, por exemplo, o Apache<sup>9</sup>;
3. Instalar um servidor de banco de dados, por exemplo, o MySQL<sup>10</sup>;
4. Executar o *script make\_project*, com os parâmetros desejados;
5. Copiar o arquivo *nome\_do\_projeto.http.conf*, para dentro da pasta de configuração do Apache;
6. Adicionar as aplicações que o projeto terá bem como as plataformas disponíveis, editando o arquivo “*project.xml*”;
7. Executar o script *xadd*, para que as informações contidas no *project.xml* sejam adicionadas ao projeto;
8. Portar a aplicação ou criar uma aplicação utilizando a API de desenvolvimento disponibilizada pelo BOINC;
9. Criar os *templates* para os arquivos de entrada e de saída, que serão utilizados pelo script *create\_work*;
10. Utilizar o script *create\_work* para criar unidades de trabalho para o projeto;

Com esses passos, um projeto estará pronto para ser iniciado, neste momento, os usuários poderão fazer parte do projeto, cadastrarem-se e começarem a receber as unidades de trabalho para processamento.

---

<sup>9</sup> Servidor de páginas http de uso livre, principal projeto da *Apache Software Foundation*.

<sup>10</sup> Servidor de banco de dados (SGBD) de uso livre que utiliza o SQL (linguagem de consulta estruturada) como interface de acesso.

## 2 – Computação Voluntária Via WEB

Para termos um software de computação voluntária totalmente *web* devemos nos atentar para alguns detalhes que são muito importantes, por exemplo, usuários que utilizam o BOINC, geralmente são usuários mais experientes que já dominam o uso de um microcomputador, ao disponibilizarmos um software para ser utilizado na *web*, temos um número maior de possíveis usuários.

Devemos estar atentos, pois o número de usuários mal-intencionados e de curiosos tende a aumentar, para desenvolvermos um software para computação voluntária totalmente *web*, podemos e devemos utilizar técnicas de softwares de computação voluntária já consagrados como é o caso do BOINC, sempre tendo a visão de que o usuário não mais irá fazer o *download*, explicitamente, do software para seu computador e processar as informações quando lhe for mais conveniente, mas sim, serão processadas as informações em tempo real.

Basicamente pode-se utilizar uma estrutura bem parecida com a que o BOINC cria e utiliza, deve-se ter, é claro, um servidor *web*, um banco de dados, arquivos de trabalho, softwares que farão o serviço interno (validação, publicação, controle de arquivos, etc), e regras de segurança.

### 2.1 – O Servidor WEB

O servidor *web* é um dos pontos mais importantes, ele será o “*disponibilizador*”, se assim o pudermos chamar, das informações e o coletor das informações já processadas, em um sistema *web* não teremos mais as questões de plataforma, visto que o necessário para a execução da aplicação é o uso de um navegador *web* e nesse caso, onde utilizamos tecnologia java para o desenvolvimento do software, é necessário ter a máquina virtual JAVA, que será exemplificada mais a frente, também será necessário que o servidor suporte páginas jsp, os servidores *web* java mais conhecidos são o TomCat da Apache, e o GlassFish da Sun.

Neste caso o servidor deverá estar bem estruturado para que não ocorram falhas e atualizações nos programas que gerenciam o servidor e em seu sistema operacional devem ocorrer em momentos de pouca movimentação além de um aviso prévio de parada, para manutenção dos serviços, ser publicado com boa antecedência.

É clara a importância de o servidor estar bem configurado e monitorado constantemente, evitando demora no carregamento das páginas, ou que o serviço fique fora do ar por muito tempo.

## 2.2 – Softwares de Controle Interno

São os programas responsáveis por fazer a retaguarda da aplicação.

- Monitorar os diretórios do servidor;
- Checar os arquivos gerados pelos voluntários;
- Gravar no banco de dados as informações processadas;
- Remover arquivos já processados;
- Criar novas unidades de trabalho (nesse caso dependem de configuração do programador);

Após os arquivos serem processados deve ocorrer o mesmo procedimento adotado pelo BOINC, deverão ser executados o Validador e o Assimilador, eles são os responsáveis, respectivamente, pela integridade do arquivo gerado e pela absorção dos resultados pelo projeto.

## 2.3 – Montando a Aplicação

Para o desenvolvimento da aplicação uma das melhores alternativas é o Java, por ser uma linguagem poderosa e de fácil portabilidade para a *web*, além disso, não podemos utilizar linguagens como php, por exemplo, que é uma linguagem que executa os comandos no servidor, precisamos que o processamento seja realizado no cliente, para isso uma boa opção é utilizarmos *Java Applet*, com o *applet* temos um sistema sendo executado dentro de outra aplicação, neste caso um navegador para internet (como o *microsoft internet*

*explorer* ou o *mozilla firefox*), utilizando dessa forma recursos da máquina cliente, principalmente memória ram e processador.

O usuário de *internet* ao entrar no site e escolher a opção de ser um voluntário, não iria necessitar instalar um software para começar a colaborar, apenas a máquina virtual java deverá estar instalada, pode-se optar por fazer uma rápida identificação do usuário, a fim de que sejam contados créditos para os usuários, organizando assim um ranqueamento, como o que o BOINC utiliza, ao carregar o *applet*, o processamento logo seria iniciado, dessa forma, uma unidade de trabalho nunca ficaria pela metade, ou o usuário termina o processamento ou a unidade de trabalho é realocada para ser processada por um outro voluntário.

Seria fácil fazer parte de um projeto, do ponto de vista dos voluntários, seria solicitado o usuário e a senha, e logo as informações seriam carregadas e o processamento iniciado, uma vez terminado o processamento da unidade de trabalho, o *applet* se encarregaria de enviar as informações ao servidor da aplicação e continuar com outra unidade de trabalho até que o usuário interrompa o processamento.

Como atrativo para novos voluntários poderia ser disponibilizado jogos para os que estivessem participando dos projetos, dessa forma enquanto o usuário estivesse jogando uma partida de xadrez, por exemplo, a aplicação voluntária também estaria sendo executada na máquina do cliente.

## 2.4 – Segurança

Por se tratar de um software *web*, os possíveis ataques de *hackers* ou de usuários maliciosos, tende a ser bem maior que o encontrado em aplicações *desktop*, como o BOINC.

Boas regras de segurança devem ser implementadas, como por exemplo, regras de controle de intrusão, injeção de sql, sobrecarga do servidor. Para o controle de intrusão, podemos criar um sistema que utiliza um *captcha*<sup>11</sup>, com isso já poderíamos tentar evitar

---

<sup>11</sup> CAPTCHA é um acrônimo para “Completely Automated Public Turing Test to Tell Computers and Humans” ou Teste de Turing Público Automatizado para diferenciar Computadores de Humanos, desenvolvido pela universidade de Carnegie-Mellon, são aquelas imagens com letras e números que aparecem em alguns sites que devem ser digitados para validação, geralmente ao realizar acesso a usuários.

uma sobrecarga no servidor de autenticação, ou softwares de tentativa de acesso por força bruta (os chamados “robôs”).

Utilizar uma ferramenta de IDS (*Intrusion Detection System*), como o próprio nome sugere é um sistema de detecção de intrusão, utilizado para detectar tentativas de ataques ou invasões por *crackers*<sup>12</sup> ou ferramentas de ataque automáticas, através da identificação de brechas de segurança, como inserção de código-fonte de scripts, vírus, *malware* ou *trojan horses*, transmitidos pela rede mundial de computadores (*internet*).

Entre outras regras básicas de segurança, como criptografia de tudo que for trafegado na grande rede, se possível um controle através de assinatura digital, conexões abertas por muito tempo sem contato com o servidor devem ser derrubadas, claro que isso dependendo do projeto em questão e da previsão de tempo necessária para realizar a tarefa em processamento.

---

<sup>12</sup> *Cracker* é o termo usado para designar quem pratica a quebra (ou *cracking*) de um sistema de segurança, de forma ilegal ou sem ética. Este termo foi criado em 1985 por *hackers* em defesa contra o uso jornalístico do termo hacker. O uso deste termo reflete a forte revolta destes contra o roubo e vandalismo praticado pelo *cracking*.

### 3 - Otimização Combinatória

É o ramo da ciência da computação que estuda problemas de otimização em conjuntos.

Segundo [5], problemas de otimização, na sua forma geral, têm como objetivo maximizar ou minimizar uma função definida sobre certo domínio. A teoria clássica de otimização trata o caso em que o domínio é infinito. Já no caso dos chamados problemas de otimização combinatória, o domínio é tipicamente finito, além disso, em geral é fácil listar os seus elementos e também testar se um dado elemento pertence a esse domínio. Ainda assim, a idéia ingênua de testar todos os elementos deste domínio na busca pelo melhor mostra-se inviável na prática, mesmo para instâncias de tamanho moderado.

Temos vários exemplos sobre otimização combinatória, um exemplo clássico seria o do caixeiro viajante, o problema consiste em, supondo que um caixeiro viajante deseje visitar  $N$  cidades (que seriam vértices de um grafo), e que entre as cidades existem caminhos (arcos ou arestas de um grafo), nos quais ele pode viajar de uma cidade à outra, cada caminho tem um número associado a ele, que pode indicar distância ou custo, de ir de uma cidade a outra, o caixeiro deseja visitar todas as cidades apenas uma única vez e, além disso, que tenha um custo menor que certo valor preliminar, onde o custo é composto pela soma dos valores das rotas percorridas pelo caixeiro viajante. Devemos notar que tal caminho nem sempre é possível, pois nem sempre conseguimos encontrar um caminho cujo custo seja menor que o custo esperado.

Algoritmos para esse tipo de problema são considerados *NP-completo* [6], as tentativas de soluções dividem-se em força bruta ou gulosa, onde temos: Força Bruta, é a solução por tentativa e erro, consiste em permutar todas as possibilidades de solução possíveis, sendo assim temos que o número de permutações é o fatorial do número de cidades menos um, o que torna o processamento impraticável; Gulosa, um algoritmo guloso consiste em buscar, na  $i$ -ésima iteração, a cidade  $x$  tal que o custo entre a cidade  $i$  e a cidade  $x$  seja o menor possível, este algoritmo não garante a solução ótima do problema, mas possui um tempo de resposta muito bom, ainda mais se comparado com algoritmos de força bruta.

Outro ponto importante é que problemas de otimização combinatória muitas vezes podem ser paralelizáveis, ou distribuídos, pois podemos analisar pedaços e depois juntar os resultados, buscando encontrar um resultado aceitável, quando um resultado ótimo não for possível. O fato de ser paralelizável também colabora para o uso de uma solução gulosa.

Existem vários tipos de problemas de otimização, em alguns será possível aplicar um algoritmo exato para a solução do problema, já em outros uma técnica não exata (heurística) será necessária ser aplicada, embora técnicas heurísticas nem sempre apresentem o resultado ótimo, mas um aproximado, o fator compensador em utilizar uma técnica heurística é que ela é muito mais rápida que as técnicas exatas, para problemas onde o número de iterações é muito grande.

Como técnicas de soluções exatas, temos, por exemplo:

- Algoritmos de grafos;
- Algoritmos gulosos;
- Algoritmo simplex;
- *Branch and bound*;

E como soluções aproximadas:

- Algoritmos genéticos;
- Busca tabu;
- GRASP (*greedy randomized adaptive search procedure*);
- Redes neurais;

### **3.1 – Otimização Combinatória e Computação Distribuída**

Nossa proposta é conseguir ganho nos resultados dos algoritmos de otimização utilizando-se de computação distribuída, mais especificamente, da computação voluntária. Portanto é necessário aplicar algoritmos que permitam a distribuição das tarefas de forma simples, um algoritmo que permite esse tipo de aplicação é o GRASP, segundo [11], podemos explicar o GRASP como: “O GRASP consiste em um método iterativo probabilístico, onde a cada iteração é obtida uma solução do problema em estudo. Cada iteração GRASP é composta de duas fases: a construtiva, que determina a solução que será

submetida à busca local, segunda fase do algoritmo, cujo objetivo é tentar obter alguma melhoria na solução corrente”.

Após a fase de construção o GRASP realiza uma busca local, por um número finito de iterações, procurando pelo melhor resultado, a vantagem do GRASP é que ele é aleatório, somente na fase de construção são passados parâmetros de controle para ajudar o construtor, após isso o algoritmo vai realizando uma busca local, até encontrar uma melhora na solução ou até o número de iterações definidas chegarem ao final.

Note-se que, para uma eficiência maior o construtor deve receber bons parâmetros, nessa fase já se pode eliminar resultados sabidamente muito fora do esperado, de forma a não perder tempo de processamento com esses resultados, após a construção, é iniciada a fase de busca local, que basicamente é buscar iterativamente a melhor solução entre a vizinhança da solução corrente.

Vemos então que aplicações desse tipo se encaixam perfeitamente na idéia de computação voluntária, basicamente os voluntários se encarregariam de realizarem as buscas locais, as unidades de trabalho já iriam com os parâmetros do construtor montados, e de preferência otimizados, e os voluntários (clientes) se encarregariam de varrer os conjuntos em busca das melhores soluções.

## Conclusão

Podemos concluir que o uso de computação distribuída para solução de problemas de otimização combinatória é extremamente viável, é claro que, para que tenhamos bons resultados o projeto deve possuir um apelo aos voluntários, muito bom, para que assim mais e mais pessoas possam colaborar com o desenvolvimento de soluções.

Num primeiro momento entrar em contato com instituições de ensino, principalmente as universidades, fazendo um trabalho de informação aos alunos, onde, provavelmente os alunos que se interessariam pelo assunto seriam os da área de tecnologia, mas enfim, esse trabalho de informar, é de extrema importância e valia para o bom andamento do projeto, visto que assim sugestões e críticas seriam enviadas de várias partes.

Ao estudarmos o *middleware* BOINC, que já é utilizado por várias instituições e em vários projetos de computação distribuída, por ser um software fácil de usar do ponto de vista dos desenvolvedores do projeto, visto que com o BOINC, um projeto é colocado para a comunidade em um tempo relativamente bem rápido.

O próprio BOINC disponibiliza para os desenvolvedores, ferramentas para auxiliar na migração de seus sistemas já existentes, ou mesmo, já desenvolver o projeto diretamente para o uso através do BOINC, com o uso de seus scripts de criação de estrutura de projeto e de criação de unidades de trabalho.

O BOINC conta com algumas características muito importantes, como por exemplo: controle de redundância a erros, integração de vários projetos, um voluntário pode participar de mais de um projeto ao mesmo tempo através do BOINC.

Utiliza criptografia em suas transferências de arquivos e também compactação de arquivos, minimizando o tráfego na rede. Já possui uma interface de administração web, ao montar um servidor BOINC. Além da página do projeto, é disponibilizada uma outra página onde é feita a administração do projeto.

Possui um sistema de pontuação para os usuários, criando com isso um ranqueamento, que serve apenas como uma lista de melhores usuários, que pode ser organizada por projeto, por país e outros índices.

O BOINC também conta com a experiência de vários anos de desenvolvimento, a idéia surgiu com o projeto SETI@Home em 1999, são 9 anos de amadurecimento e desenvolvimento do sistema.

O BOINC conta hoje com 1.283.862 (um milhão duzentos e oitenta e três mil e oitocentos e sessenta e dois voluntários) em 55 projetos diferentes, veja na Tabela 2 abaixo os 15 projetos mais ativos no BOINC, e na tabela 3, os 36 países com mais pontuação no BOINC, ambos retirados de <http://boincstats.com/index.php?pr=bo> em 25/02/2008:

**Tabela 2 : Projetos mais ativos no BOINC**

<b>Projetos</b>	<b>Descrição</b>	<b>Usuários</b>
BOINC geral	---	1,283,862
SETI@home	Analisar sinais de rádio captados por <a href="#">radiotelescópios</a> terrestres	783,811
World Community Grid	Visa ser o maior computador público do mundo, destina-se a analisar vários projetos.	114,579
Einstein@Home	Analisar dados de observatórios especializados em ondas gravitacionais, como o LIGO <sup>13</sup> e o GEO 600 <sup>14</sup> .	191,323
Climate Prediction	Prever as condições climáticas com mais precisão e maior antecedência.	150,053
Rosetta@Home	Procura determinar as formas tridimensionais de proteínas que podem ser usadas na pesquisa de medicamentos.	187,287
QMC@Home	Tem o objetivo de analisar os pares-base do DNA.	24,422
ABC@Home	Busca os chamados triplos-abc <sup>15</sup> ,	11,300
Cosmology@Home	Cálculos de estudos cosmológicos.	4,889
Spinhenge@Home	Obter estatísticas computacionais para Spin <sup>16</sup> dinâmicos.	36,356
SIMAP	Cálculo de semelhanças entre as proteínas.	23,038
Tanpaku	O objetivo é prever estruturas de proteínas, semelhante aos projetos rosetta@boinc, folding@home e predictor@home.	14,622
Malaria Control	Ajudar na busca de cura e controle da malária.	15,274
yoyo@Home		2,402
RieselSieve	Provar qual dos 509203 é o menor primo para qualquer número $n \geq 1$ , $k \cdot 2^n - 1$ usando a Linguagem de Programação PERL	7,092
PrimeGrid	Fatorar os números do RSA Factoring Challenge.	11,849

<sup>13</sup> *Laser Interferometer Gravitational Wave Observatory.*

<sup>14</sup> Observatório de ondas gravitacionais GEO 600 na Alemanha.

<sup>15</sup> Triplos-abc são números inteiros positivos a, b, c como esses  $a+b=c$ ,  $a < b < c$ , a, b, c que não tenham divisores comuns e  $c > \text{rad}(abc)$ , chamado de radical de abc.

<sup>16</sup> Spin é o movimento de rotação de uma partícula - um elétron, por exemplo.

O Brasil realizou em 2007 um projeto utilizando a estrutura BOINC com o projeto *Genome Comparison* (Comparação de Genomas), um projeto da Fiocruz (Fundação Oswaldo Cruz), este projeto já foi terminado com sucesso em 21 de julho de 2007, mais informações sobre este projeto pode ser encontrada em <http://www.dbbm.fiocruz.br/GenomeComparison>, última consulta em 25/02/2008.

**Tabela 3:** Países mais ativos no BOINC

Pos.	País	Pos.	País
1	Estados Unidos	19	Rússia
2	Alemanha	20	Taiwan
3	Sem país Definido	21	Dinamarca
4	Inglaterra	22	Noruega
5	Canadá	23	Hungria
6	Japão	24	China
7	França	25	Portugal
8	Austrália	26	Nova Zelândia
9	República Checa	27	Eslováquia
10	Holanda	28	África do Sul
11	Itália	29	Grécia
12	Polônia	30	Hong Kong
13	Espanha	31	Eslovênia
14	Finlândia	32	<b>Brasil</b>
15	Suécia	33	Croácia
16	Áustria	34	Irlanda
17	Suíça	35	Bulgária
18	Bélgica	36	Romênia

Veja abaixo na Tabela 4, um resumo geral de dados de todos os projetos BOINC, note a capacidade de processamento do BOINC, consultado o site <http://www.top500.org>, que lista os maiores computadores do mundo o primeiro lugar pertence ao computador IBM BlueGene/L com capacidade de 478.2 TeraFlops, podemos ver abaixo que a capacidade da estrutura BOINC hoje processa 918.224 TeraFlops, portanto podemos facilmente notar o tamanho da capacidade ao utilizarmos sistemas de computação voluntária. O maior computador brasileiro a figurar no top500 é um *Hewlett-Packard* modelo *Cluster Platform 3000 BL460c*, Xeon 53xx 2.33GHz, Infiniband, da PETROBRÁS com seus “modestos” 6,2 TeraFlops. Na Tabela 5, são listados os 5 maiores computadores do mundo em capacidade de processamento.

**Tabela 4:** Resumo de dados do projeto BOINC

	<b>Total</b>	<b>Ativos</b>
Usuários	1.283.862,00	317.352,00
Hosts <sup>17</sup>	2.730.023,00	559.225,00
Times	71.529,00	29.133,00
Países	246	227
Credito Total		
	47.473.615.339,00	
Média de Crédito Recente		
	91.822.444,00	
Media de operações por ponto flutuante por segundo		
	918,224.4 GigaFLOPS / 918.224 TeraFLOPS	

**Tabela 5:** Tabela contendo os 5 (cinco) maiores computadores do mundo

Pos.	Proprietário	Fabricante	Descrição	País	Processadores	GigaFlops
1	DOE/NNSA/LLNL	IBM	eServer Blue Gene Solution	USA	212992	478200
2	<i>Forschungszentrum Juelich (FZJ)</i>	IBM	Blue Gene/P Solution	Alemanha	65536	167300
3	<i>SGI/New Mexico Computing Applications Center (NMCAC)</i>	SGI	SGI Altix ICE 8200, Xeon quad core 3.0 GHz	USA	14336	126900
4	<i>Computational Research Laboratories, TATA SONS</i>	Hewlett-Packard	Cluster Platform 3000 BL460c, Xeon 53xx 3GHz, Infiniband	Índia	14240	117900
5	<i>Government Agency</i>	Hewlett-Packard	Cluster Platform 3000 BL460c, Xeon 53xx 2.66GHz, Infiniband	Suécia	13728	102800

Uma das dificuldades com o BOINC é por ser um *software desktop*, isso pode inibir alguns usuários que não são experientes e mesmo aqueles sem paciência, que não gostam de realizar o *download* e instalar o programa em sua máquina, outra situação é que muitas vezes o usuário realiza o *download*, instala, mas não realiza nenhum trabalho, e muitas vezes o BOINC, permanece instalado na máquina, mas foi executado somente quando da instalação do mesmo, após isso ficou sem ser iniciado novamente.

Essas situações não ocorrem em um sistema *web*, o usuário pode até desistir de terminar o trabalho, mas feito isso, logo sua unidade de trabalho volta para a fila, para que seja processada por outro voluntário.

Uma característica no sistema *web*, é que a unidade de trabalho deverá ser curta, pois a maioria dos usuários não gosta de ficar muito tempo esperando, mesmo que ele possa realizar outro trabalho durante o processamento, por isso, as unidades de trabalho devem ser curtas e rápidas, o que no BOINC não é um problema, o único empecilho, mas ai em

<sup>17</sup> Em informática entendemos por *host* qualquer computador conectado a uma rede.

ambos os casos (BOINC e *web*), é o tamanho, unidades de trabalho que processam muitos dados tendem a serem maiores, o que dependendo da conexão, pode tornar a experiência do usuário muito cansativa.

Com o BOINC, o desenvolvedor da aplicação deve ficar atento com as atualizações de software, sempre que for disponibilizar uma unidade de trabalho que se utiliza de alguma nova propriedade da aplicação mestre, devem ser distribuídas atualizações para as plataformas na qual o projeto já possui suporte. Já no ambiente *web*, sempre que uma nova atualização for necessária, automaticamente todos os voluntários estarão atualizados, afinal a aplicação será carregada em tempo de execução do servidor para o navegador do voluntário.

## Trabalhos Futuros

Como trabalho futuro fica a necessidade de se implementar o sistema *web* de computação voluntária e hospedá-lo em algum domínio para que seja feita a checagem de aceitação pelo público, seria interessante fazer um trabalho de contato com outras universidades para que fosse informado a classe acadêmica desse domínio, e é claro disponibilizar um endereço de e-mail para sugestões e críticas.

## Referências

- [1] Nazário, Marcos F. C. **PHILANTROPIC GRID SIMULATOR: UMA FERRAMENTA PARA O ENSINO/APRENDIZAGEM DE GRID COMPUTING.**  
<http://www.cci.unama.br/margalho/portaltcc/tcc2005/PDF/012.pdf>
- [2] <http://www.cbpf.br/cat/grid/> Projeto GRID do CBPF, Acesso em 17/10/2007.
- [3] <http://boinc.berkeley.edu/> Projeto BOINC, Acesso em 19/10/2007
- [4] Taurion, Cezar **Grid Computing: um novo paradigma computacional**, Brasport, 2004.
- [5] <http://www.ic.unicamp.br/~fkm/problemas.html>, Acesso em 05/11/2007
- [6] Feofiloff, Paulo. [http://www.ime.usp.br/~pf/analise\\_de\\_algoritmos/](http://www.ime.usp.br/~pf/analise_de_algoritmos/) , Acesso em 10/12/2007
- [7] Sottrup, Christian U. e Pedersen, Jakob G. **Developing Distributed Computing Solutions Combining Grid Computing and Public Computing**, Departamento de Ciência da Computação da universidade de Copenhagen, 2005.
- [8] Perez, Juan A.  
<https://twiki.cern.ch/twiki/pub/LHCAtHome/LinksAndDocs/boincciemat06.pdf>, acesso em 15/01/2008.
- [9] Anderson, David P., Christensen, Carl e Allen, Bruce. **Designing a Runtime System for Volunteer Computing**
- [10] Anderson, David P. e Fedak, Gilles. **The Computational and Storage Potential of Volunteer Computing**
- [11] RANGE, Maria Cristina, ABREU, Nair Maria Maia de e BOAVENTURA-NETTO, Paulo Oswaldo. **GRASP para o PQA: um limite de aceitação para soluções iniciais.** *Pesquisa. Operacional.* Junho de 2000, vol.20, no.1, p.45-58. ISSN 0101-7438.
- [12] <http://www.top500.org>, acesso em 25/02/2008