FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Computação Voluntária

Girson César Silva Monteiro

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Prof. Jorge Manuel Gomes Barbosa

26 de Julho de 2010

Computação Voluntária

Girson César Silva Monteiro

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Doutor Armando Jorge Miranda de Sousa (Professor Auxiliar)

Vogal Externo: Doutor Pedro Miguel do Vale Moreira (Professor Adjunto)

Orientador: Prof. Jorge Manuel Gomes Barbosa (Professor Auxiliar)

_____

26 de Julho de 2010

Resumo

A computação voluntária tem vindo a ganhar uma importância crescente devido a capacidade de computação disponível actualmente nos computadores pessoais. Para muitos trabalhos científicos estima-se que esta estratégia possa complementar (ou até substituir) a necessidade de grandes investimentos em infra-estruturas de computação de elevado desempenho.

Esta forma de computação surgiu em 1996 com o projecto GIMPS que consistia na procura computorizada de grandes números primos, utilizando a fórmula de Mersenne, 2^p-1, em que "p" representa o número primo. Qualquer pessoa com computador e com uma ligação a Internet podia instalar o programa cliente do GIMPS e participar no projecto disponibilizando poder de computação durante períodos de tempo de não utilização.

Seguindo esta analogia seguiram-se os projectos distributed.net em 1997 e SETI@home em 1999, sendo este último o mais popular de todos.

Com base neste paradigma de computação muitos projectos de investigação científica foram desenvolvidos, trabalhos na área da física, astronomia, matemática, biologia molecular, normalmente ligados a instituições universitárias que podem beneficiar desta forma de computação, o principal problema é saber como fazer isso e que tecnologias utilizar. Para solucionar esse problema foram propostos protótipos, utilizando as tecnologias BOINC, Condor e Hadoop, onde se pôde avaliar a oportunidade da introdução destes serviços em Instituições Universitárias. Para este fim foram realizados alguns testes nos sistemas implementados com as tecnologias já referidas e definida uma matriz de comparação destas tecnologias focando em características que definem um sistema distribuído.

Aos Meus Pais

Agradecimentos

Agradeço às pessoas e entidades que de alguma forma contribuíram para a realização desta dissertação, com especial destaque ao meu orientador Jorge Manuel Gomes Barbosa por sempre me ajudar a ordenar e centrar as ideias no caminho certo.

Ao Aníbal Leite, Centro de informática Prof. Correia Araújo, Faculdade de Engenharia, da Universidade do Porto, que durante a fase de implementação ajudou-me muito na configuração das máquinas utilizadas nos testes.

Ao Luís António Diniz Fernandes de Morais Sarmento, Departamento de Informática, Faculdade de Engenharia, da Universidade do Porto, pelos esclarecimentos acerca do Hadoop que foram importantes nesta investigação.

Ao Tito Carlos Soares Vieira, responsável pelo Centro de informática Prof. Correia Araújo, Faculdade de Engenharia, da Universidade do Porto, pela oportunidade de poder realizar esta investigação.

À minha família pelo apoio incondicional que sempre me deram ao longo destes anos. Aos meus pais que são a minha inspiração, à minha irmã pela sua alegria, dedicação e apoio nas horas difíceis.

O meu profundo agradecimento a todas as pessoas que contribuíram para a concretização desta dissertação, estimulando-me intelectual e emocionalmente.

Índice

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

FLOPS	*FLoating Point Operations Per Second*
HTTP	HyperText Transfer Protocol
XML	eXtensible Markup Language
CPU	Central Processing Unit
BOINC	*Berkeley Open Infrastructure for Network Computing*
SETI	*Search for Extra-Terrestrial Intelligence*
VO	Virtual Organizations
URL	Uniform Resource Locator
DRM	Distributed Resource Management
RU	Remote-Unix
DAG	Direct Acyclic Graph
IPC	Inter-process communication
KLT	Kernel-Level Thread
MPI	Message Passing Interface
I/O	Input / Output
GT	Globus Toolkit
PBS	Portable Batch System
UNICORE	*Uniform Interface to Computing Resources*
EC2	Amazon Elastic Compute Cloud
NQS	Network Queuing System
JVM	Java Virtual Machine
RAM	Random Access Memory
GSI	Grid Security Infrastructure
UID	User Identifier
HDFS	Hadoop Distributed File System
GIMPS	*Great Internet Mersenne Prime Search*
SSH	Secure Shell

1 Introdução

1.1 Contexto/Enquadramento

Actualmente, face ao aumento do poder de processamento dos computadores pessoais, a computação voluntária tem vindo a ganhar algum destaque na comunidade científica. Projectos de investigação como o SETI@home e Folding@home contribuíram muito para a divulgação desta forma de computação distribuída.

Computação voluntária têm por base o trabalho voluntário, onde recursos computacionais e de armazenamento são cedidos a projectos de investigação, sem qualquer custo. Esta técnica permite a participação dos cidadãos na investigação científica de uma maneira directa e em tempo real, oferecendo tempo de processamento dos seus computadores para realização de cálculo computacional de interesse científico por técnicas de computação distribuída semelhantes às da computação Grid. Visa aproximar os cidadãos da investigação e providenciar-lhes a possibilidade de doarem tempo do seu computador, quando está inactivo, para a produção de conhecimento científico e, ao mesmo tempo, fornecer à comunidade científica uma poderosa ferramenta de cálculo.

Projectos de investigação desenvolvidos na sua grande maioria em instituições universitárias têm muitas vezes a necessidade de recorrer a super computadores que devido ao seu elevado custo não são acessíveis. Deste modo a computação voluntária mostra-se como uma alternativa viável e menos dispendiosa para casos em que o paralelismo utilizado é o de dados, como é o caso do projecto Climateprediction.net [Cpred] em que o objectivo é desenvolver um modelo avançado de prognóstico do clima, em que milhares de dados são processados por diferentes clientes de forma independente executando a mesma operação.

1.2 Descrição do Problema

O proposto para esta tese consiste em analisar os desenvolvimentos na área da computação voluntária, computação com PC, computação distribuída e sistematizar as tecnologias emergentes na área e avaliar a adequação e a oportunidade da introdução destes serviços em Instituições Universitárias, ou seja, estudar *softwares* como o BOINC, Condor e Hadoop, entre outros considerados relevantes, compara-los nos aspectos considerados mais importantes para este tipo de serviço e identificar os mais adequados para ambientes Universitários.

Pretende-se com este estudo fazer uma análise de tecnologias associadas a computação voluntária e desenvolver um ou mais protótipos onde se possa avaliar a adequação e oportunidade da introdução destes serviços em Instituições Universitárias.

1.3 Motivação e Objectivos

A computação voluntária tem vindo a despertar cada vez mais o interesse da comunidade científica e académica. Esta forma de computação distribuída, para muitos trabalhos científicos, pode ser a alternativa à necessidade de grandes investimentos em infra-estruturas de computação de elevado desempenho, e estando muitos destes trabalhos ligados a projectos desenvolvidos em Instituições Universitárias há a necessidade de responder a pergunta “Qual ou quais o(s) *software(s)* de computação voluntária mais adequado(s) para Instituições Universitárias”, a solução para este problema pode ser determinante na introdução destes serviços nos ambientes universitários, assim como o sucesso de projectos que necessitarem destes serviços.

As questões que se colocam é como determinar que um determinado *software* pode ser o mais adequado para uma instituição universitária e como avaliar os resultados obtidos.

A importância deste estudo advém da necessidade de Instituições Universitárias conciliarem os recursos disponíveis com as necessidades dos seus projectos. A melhor forma de fazer isso é fazer uma análise detalhada de ferramentas existentes na área da computação distribuída.

Neste sentido definiram-se os seguintes objectivos a serem cumpridos durante o processo de investigação:

- Analisar os desenvolvimentos na área da computação voluntária, computação distribuída e sistematizar as tecnologias emergentes na área.
- Estudar os *softwares* BOINC, CONDOR e HADOOP, entre outros que se identifiquem como relevantes.
- Definir uma matriz de comparação para os *softwares* seleccionados.
- Comparar os *softwares* nos aspectos mais relevantes e identificar o(s) mais apropriado(s) para ambientes Universitários.
- Implementar protótipos com os *softwares* BOINC, Condor e Hadoop.
- Avaliar os resultados obtidos e apresentar as devidas conclusões.

1.4 Estrutura da Dissertação

A dissertação foi estruturada em cinco capítulos que reflectem genericamente as fases do trabalho desenvolvido de acordo com os objectivos propostos:

No Capitulo 1 foi realizada uma introdução ao trabalho, onde foram esgrimidas as motivações e formulação do problema que traçaram os objectivos propostos.

No Capítulo 2 é efectuada uma descrição um tanto técnica aliada a uma análise bibliográfica sobre o paradigma da computação distribuída de forma a introduzir alguns conceitos considerados relevantes na compreensão deste tipo de computação. Finalmente são analisadas certas características, potencialidades e desafios na implementação de sistemas de computação distribuída focando mais concretamente em sistemas de computação voluntária.

No Capítulo 3 são descritas as tecnologias utilizadas ao longo do estudo desenvolvido, realçando os aspectos mais relevantes como por exemplo, o funcionamento, a arquitectura, o desempenho, a segurança e a fiabilidade.

No Capítulo 4 são apresentadas as avaliações dos resultados obtidos de testes efectuados aos protótipos implementados e uma análise comparativa dos *softwares* testados.

Por fim no Capitulo 5 são apresentadas as principais conclusões tiradas do presente estudo e são apresentadas algumas perspectivas de desenvolvimento.

2 Revisão Bibliográfica

Neste capitulo pretende-se descrever o estado da arte, estudos feitos na área da computação voluntária e distribuída, e apresentar trabalhos relacionados com o tema de forma a mostrar o que existe neste domínio e identificar quais os problemas que ainda podem ser estudados. Esta revisão tem por objectivo providenciar uma visão mais ampla sobre o tema e o seu impacto na comunidade científica focando principalmente na sua importância e contributos para a ciência em geral.

2.1 Introdução

Computação voluntária é considerada uma forma de computação distribuída [LFS01] que tem por base o trabalho voluntário em que pessoas, organizações e instituições públicas e privadas de todo o mundo podem contribuir para projectos de investigação doando tempo de processamento não utilizado dos seus computadores pessoais ou *workstations* simulando um super-computador a baixo custo.

O estudo levado a cabo durante a preparação desta dissertação teve como finalidade a recolha e identificação de informação relevante, a compreensão do paradigma da computação distribuída, destacando tipo de sistemas associados a esta forma de computação, focando essencialmente nos sistemas de computação voluntária, assim como nas suas potencialidades e desafios enfrentados na implementação.

2.2 Paradigma da computação distribuída

Definição de um sistema distribuído segundo Andrew S. Tanenbaum :

> "*Collection of independent computers that appear to the users of the system as a single computer.*" [ASM, chap. Introduction]

Basicamente Andrew S. Tanenbaum refere-se ao principal objectivo a ser atingido na implementação de um sistema distribuído, um sistema que aos olhos do utilizador funcione como um único computador. Seguindo este contexto a computação distribuída consiste em distribuir a carga computacional de execução de uma tarefa por diferentes computadores ligados a um ou mais sistemas distribuídos. Melhor dizendo, diferentes objectos ou elementos de um programa são executados em processadores diferentes com o objectivo de solucionar o mesmo problema.

2.2.1 Sistemas de Computação distribuída

Sistemas de computação distribuída foram arquitectados para que um conjunto de computadores heterogéneos, *workstations* e servidores funcione e comporte-se como um único sistema de computação [HS04, chap. 1]. Nestes ambientes de computação utilizadores podem aceder de forma uniforme os recursos locais e remotos, e correr processos de qualquer lugar no sistema, sem se preocupar em que computadores estes processos são executados. Avanços nas tecnologias e ferramentas de processamento e de rede tornaram possível atingir as seguintes vantagens:

- Aumento de *performance* - A existência de múltiplos computadores em um sistema distribuído permitiu que aplicações pudessem ser processadas em paralelo e isto reflectiu-se em melhorias na *performance* da aplicação e do sistema. Como por exemplo, o desempenho de um sistema pode ser melhorado através da replicação dos seus serviços ou funções a mais do que um computador. Esta melhoria é visível no tráfico da rede associado ao acesso de ficheiros em rede o que reduz o congestionamento na rede e nas filas de espera.
- Partilha de Recursos - Sistemas distribuídos apresentam uma boa relação custo-eficiência e disponibilizam acesso eficiente a todos os sistemas de recursos. Utilizadores podem partilhar recursos especiais de hardware e *software* dispendiosos, assim como servidores de base de dados, computação, informação multimédia e impressoras. Neste caso de computação distribuída estes sistemas permitem a partilha eficiente de recursos computacionais.
- Aumento da extensibilidade - Sistemas distribuídos são desenhados ou concebidos para serem flexíveis e adaptáveis de forma a que para computações específicas, o sistema tenha capacidade definir uma configuração que permita incluir o maior número de computadores e recursos. Limitações a nível da capacidade do sistema de

ficheiros e poder de computação podem ser contornados através da adição de mais computadores e servidores de ficheiros ao sistema.

- Aumento da fiabilidade, disponibilidade, e tolerância a falhas - A existência de múltiplos recursos de computação e armazenamento num sistema distribuído torna-o mais susceptível a introdução de mecanismos de tolerância a falha, o sistema se apresenta aos utilizadores mais eficiente sem grandes custos. Servindo-se da replicação dos serviços e funções do sistema para garantir que o sistema funcione mesmo em casos de avaria ou falha de um ou mais componentes.
- Custo/eficiência – Com o desempenho dos computadores a duplicar a cada dois anos enquanto os seus custos diminuem para metade a cada ano e juntando o facto de que as ligações de rede estão mais rápidas, a implementação de sistemas de computação distribuída têm-se tornado mais atractivas em termos de custo/*performance.*

2.2.1.1 Sistemas de computação Grid

Sistemas de computação Grid surgiram nos anos 90 como proposta a uma nova infra-estrutura de computação distribuída concebida para auxiliar cientistas e engenheiros em trabalhos científicos com grandes exigências de computação, é um novo paradigma da computação distribuída onde serviços são organizados como serviços *web* que podem ser alocados e acedidos de forma flexível e dinâmica, normalmente para resolver problemas que demandam recursos espalhados por diferentes domínios administrativos [PCS04].

> "*The Grid is an emerging infrastructure that will fundamentally change the way we think about – and use – computing. The word Grid is used by the analogy with the electric power grid, which provides pervasive access to electricity and, like the computer and a small number of other advances, has had a dramatic impact on human capabilities and society. Many believe that by allowing all components of our information technology infrastructure – computational capabilities, databases, sensors, and people – to be shared flexibly as true collaborative tools, the Grid will have a similar transforming effect, allowing new classes of applications to emerge*." [ICK03, Preface]

Existem varias definições na literatura sobre o que é a Grid, mas estas, constantemente se confundem com o real significado de sistemas Grid, por esta razão Ian Foster, considerado o pai da Grid, sugeriu uma *checklist* englobando a essência de outras definições [IFT02]:

- Recursos coordenados não se sujeitam a um controlo centralizado – Sistemas Grid coordenam e integram recursos e utilizadores em diferentes domínios de controlo, por exemplo computador pessoal contra computação central; diferentes unidades administrativas da mesma empresa; ou empresas diferentes; e abordam questões relacionadas com a segurança, política, pagamento, adesão, e assim por diante que resultam destas configurações. Caso contrário, estamos lidando com sistemas de gestão local.

- Utilizar interfaces e protocolos de propósito gerais, *standard* e abertos – A Grid é construída a partir de protocolos e interfaces de múltiplos propósitos que abordam questões fundamentais como autenticação, autorização, descoberta de recursos, e acesso aos recursos. Por este motivo os protocolos e interfaces devem ser *standard* e abertos. Caso contrário, estamos lidando com uma aplicação, um sistema específico.
- Providenciar qualidades de serviço não triviais - Sistemas em Grid permitem que os recursos do cliente sejam utilizados de forma coordenada de modo a providenciar várias qualidades de serviço, relatando, por exemplo, o tempo de resposta, *throughput*, disponibilidade, segurança e/ou a recolocação de múltiplos tipos de recursos para se adequar às complexas exigências do utilizador, para que a utilidade de um sistema combinado seja significativamente maior que a soma das partes que o constituem.

O problema real e específico que está por trás do conceito de Grid é coordenar a partilha de recursos e resolver problemas em uma organização virtual dinâmica e multi-institucional. A partilha de recursos não corresponde unicamente a troca de ficheiros mas também acesso directo a computadores, *software*, dados, e outros recursos. Políticas e regras de partilha e acesso devem ser definidas para um maior controlo dos recursos disponíveis na Grid formando organizações virtuais (VO), que são um conjunto de indivíduos e/ou instituições definidas por estas políticas e regras [ICS01].

Para melhor entender o que constitui uma VO podemos recorrer a alguns exemplos tais como: *Application Service Providers*[1] (ASP), *Storage Service Providers*[2] (SSP), *Cycle Providers*, entre outros. Estes exemplos representam uma abordagem a computação e resolução de problemas baseados na colaboração em ambientes ricos em dados e computação.

Para implementar estas infra-estruturas que integram recursos de múltiplas instituições, cada um com as suas políticas e mecanismos podemos recorrer a ferramenta Globus Toolkit que permite utilizar os protocolos abertos e de propósito geral para negociar e gerir a partilha, assim como abordar as múltiplas dimensões da qualidade de serviço, incluindo segurança, fiabilidade e *performance*.

2.2.1.2 Sistemas de Computação Voluntária

Computação voluntária é uma forma de computação que aposta no trabalho voluntário para obter o poder de computação que necessita para resolver problemas complexos ligados a projectos de investigação em áreas científicas como astrofísica, biologia molecular, física, medicina, estudo do clima, etc. Este paradigma tem por base o aproveitamento do tempo de processamento não utilizado por milhares de computadores espalhados pelo mundo, ou seja, qualquer computador que não esteja a ser utilizado durante um período tempo e tenha uma ligação a Internet pode fazer parte de um super computador virtual e contribuir para um projecto de investigação. Por esta razão a computação voluntária é considerada actualmente uma

1 Empresas que disponibilizam serviços ou aplicações informáticas com base na *Web*.

2 Empresas que disponibilizam capacidade de armazenamento e serviços de gestão relacionados.

alternativa a grandes investimentos em infra-estruturas de computação de elevado desempenho, uma oportunidade única de explorar projectos que antes, por falta de recursos, não podiam ser explorados sem se recorrer a super computadores.

Deste modelo de computação surgiram diferentes sistemas de computação voluntária servindo diferentes propósitos como é o caso de verdadeiros sistemas de computação voluntária, como o SETI@home ou distributed.net, cujo principal propósito é aproveitar ao máximo possível o poder de computação de computadores pessoais ligados à Internet. Para além destes sistemas existem outros para fins privados, colaborativos e comerciais [LFS01].

Verdadeiros sistemas de computação voluntária

Os projectos de investigação mais populares como SETI@home, Folding@home e Climateprediction.net são considerados verdadeiros sistemas de computação voluntária. Os participantes podem anexar-se aos projectos sem quaisquer encargos, podem desistir da participação a qualquer altura, gozam do total anonimato e não esperam ser compensados pela sua contribuição. Estas condições permitem que estes sistemas tenham o potencial de adquirir poder de computação de milhões de computadores ligados na Internet.

Com o aumento do poder de processamento dos computadores pessoais, da velocidade da Internet, do número de computadores ligados a rede mundial este tipo de sistema vai ganhando cada vez mais adeptos que vêem nesta forma de computação uma oportunidade de criar um super computador a nível mundial mais poderoso que qualquer super computador convencional.

Apesar de todas as vantagens apresentadas a dependência de voluntários é bastante significante. O sucesso de um projecto está muito dependente da sua capacidade de angariar participantes através da divulgação na Internet por envio de correio electrónico ou anúncios.

Problemas científicos com grande importância para a humanidade como por exemplo a cura de doenças, estudos relacionados com mudanças climáticas, aquecimento global, despertam um grande interesse nas pessoas, sentem-se orgulhosas de contribuir para um bem comum. Para além destes problemas científicos que afectam a sociedade e a humanidade, alguns projectos que estão mais focados em resolver desafios ligados a matemática, como o caso do GIMPS, ou então a criptografia e a procura de vida inteligente extra-terrestre (SETI) podem ser apelativos e também despertar o interesse de um número considerável de voluntários.

Com verdadeiros sistemas de computação voluntária, uma organização com poucos recursos e com um bom propósito pode implementar um sistema de computação voluntária e convidar utilizadores na Internet a “emprestar” tempo de computação não utilizado através da instalação de um programa que actua como *screensaver* ou então pela visita à uma página *Web*.

Sistemas privados de computação voluntária

Sistemas privados de computação voluntária são mais indicados para organizações como universidades, empresas e laboratórios, providenciam capacidades de super-computação a um baixo custo. A maioria das empresas e universidades dispõe de uma rede interna com muitos computadores que permanecem a maior parte do tempo sem utilização. Estes recursos não utilizados podem ser aproveitados durante o horário laboral, quando em tarefas não-computacionais como processamento de texto, ou à noite no horário pós-laboral.

No caso da Faculdade de Engenharia da Universidade do Porto (FEUP), que é uma instituição universitária, este tipo de sistema é o mais adequado as suas necessidades computacionais, dispõem de vários computadores ligados em rede nos departamentos, laboratórios e nas salas de computadores que a noite podem ser aproveitados para a resolução de problemas ligados a engenharia tais como: simulações de *Monte Carlo*, processamento de imagens, computação gráfica, programação distribuída, sistemas dinâmicos, biologia molecular, etc.

Apesar da limitação quanto ao poder de computação que se pode adquirir a partir dos recursos existentes sistemas privados de computação voluntária apresentam algumas vantagens em relação aos verdadeiros sistemas de computação voluntária no que diz respeito a fiabilidade, segurança e tolerância a falhas, há um maior controlo dos recursos disponíveis, menor probabilidade de ataques maliciosos, uma maior confiança nos resultados obtidos e o *bottleneck* que é a transferência de dados pela Internet pode ser reduzido recorrendo a redes internas mais rápidas como por exemplo a *fast ethernet*.

Sistemas colaborativos de computação voluntária

Os mesmos mecanismos utilizados em sistemas privados de computação voluntária podem ser aproveitados para implementar computação voluntária entre organizações, instituições públicas e privadas a trabalharem juntos em projectos de investigação que necessitem de grande poder computacional. As organizações podem cooperar entre si através da troca de recursos computacionais utilizando um sistema colaborativo de computação voluntária. Por exemplo, duas universidades podem estabelecer uma parceria em que tendo uma delas recursos computacionais disponíveis a outra pode usufruir destes recursos para projectos que deles necessitem.

Esta forma de computação pode reunir as condições necessárias para desenvolver um laboratório virtual a nível mundial, um local onde cientistas de todo o mundo podem reunir e partilhar ideias, dados, experiências e poder de computação. Tecnologias actualmente em desenvolvimento como videoconferência, *whiteboards* partilhados, instrumentos acedidos remotamente, e bases de dados partilhadas, podem tornar estes laboratórios virtuais possíveis.

Um outro cenário plausível para a utilização destes tipos de sistemas, considerando a possibilidade geográfica, seria a “permuta” de recursos computacionais entre organizações em diferentes fusos horários. Como por exemplo, um laboratório nos Estados Unidos poderia disponibilizar os seus computadores à noite para processar dados de um laboratório no Japão e em troca utilizava os seus computadores de manhã, desta feita ambos usufruem de um maior poder de computação durante as horas que precisarem.

Sistemas comerciais de computação voluntária

Com o desenvolvimento de mecanismos de comércio electrónico apropriados e fiáveis a possibilidade de vender, comprar e trocar poder de computação tornou-se uma realidade, pessoas ou grupos em todo o mundo podem comercializar os seus recursos computacionais dependendo das suas necessidades.

Empresas que necessitem de poder de processamento extra podem contactar uma máquina que faz a gestão dos recursos computacionais disponíveis e esta por sua vez pode tentar adquirir estes recursos de outras empresas que estejam a vender o tempo de CPU livre.

Aplicações comerciais de computação voluntária podem incluir também sistemas baseados em contractos, em que o poder de computação é utilizado como forma de pagamento para bens e serviços recebidos pelos utilizadores.

2.2.2 Potencialidades e Desafios na Implementação

O interesse em sistemas de computação distribuída tem vindo a aumentar, a proliferação de sistemas de alto-desempenho e o surgimento de redes de alta velocidade tem sido os principais factores a contribuir para este fenómeno.

Com o aumento de requisitos a nível da computação quanto ao poder e diversidade, é óbvio afirmar que uma única plataforma de computação não irá satisfazer todos estes requisitos. Apenas sistemas de computação distribuída tem o potencial de atingir a tão desejada integração de recursos e tecnologias com credibilidade mantendo a usabilidade e flexibilidade. Explorar este potencial requer alguns avanços na ciência da computação [HS04]:

- Tecnologia de processamento – O enorme poder de processamento das futuras gerações de microprocessadores não pode ser aproveitado a não ser que se verifique melhorias significativas em componentes como memórias e sistemas I/O.
- Tecnologia de rede – A *performance* de algoritmos distribuídos depende inteiramente da largura de banda e da latência da comunicação entre os nós da rede. Atingir grande largura de banda e baixa latência envolve não só *hardware* rápido, mas também protocolos de comunicação eficientes que minimizam a sobrecarga do *software*.
- *Software,* ferramentas e ambientes – O desenvolvimento de aplicações distribuídas não é um processo trivial, requerer um total conhecimento da aplicação e sua arquitectura. Apesar dos sistemas de computação distribuída providenciarem um enorme poder de computação e grande flexibilidade, esta flexibilidade implica um aumento dos graus de liberdade que deve ser optimizado para que benefícios destes tipos de sistemas sejam totalmente explorados. Durante o desenvolvimento de *software* é necessário que o programador escolha a melhor configuração de hardware para uma determinada aplicação, a melhor decomposição do problema com base na configuração seleccionada, a melhor estratégia de comunicação e sincronização, e assim por diante. Todo este processo pode ser muito complexo em ambientes de grande dimensão, escolher a melhor alternativa pode ser uma tarefa difícil. Ferramentas simples e portáteis de desenvolvimento de *software* podem ser necessárias para auxiliar os programadores na distribuição adequada das aplicações fazendo o uso eficiente dos recursos computacionais.

Aproveitar todo o potencial que esta forma de computação tem a oferecer implica enfrentar grandes desafios na sua implementação, ultrapassar vários obstáculos. Como por exemplo,

reduzir a latência da rede que requer progressos nos protocolos de comunicação de forma a minimizar a sobrecarga do processamento do protocolo, melhorar a interface de rede do sistema operativo e melhorar o algoritmo computacional para esconder a latência [HS04].

Estes problemas são considerados aspectos fundamentais para o sucesso da computação distribuída, facto esse que tem vindo a ser motivo de investigação científica dado a sua importância.

Melhorias nestes aspectos podem trazer grandes benefícios a sistemas de computação distribuída com grande potencial como é o caso da computação voluntária.

Sistemas de computação voluntária tem o potencial de oferecer um enorme poder de computação, o número de computadores ligados na Internet está a crescer rapidamente e estima-se que atinja mil milhão em 2015 disponibilizando muitos PETA FLOPS em poder de computação. Se 100 milhões de utilizadores disponibilizarem 10 Gigabytes de armazenamento cada, o total de um Exabyte (10^{18} bytes) pode ultrapassar a capacidade de armazenamento de qualquer sistema centralizado de armazenamento [Dpa04].

Projectos como o SETI@home e World Community Grid são casos exemplares de sucesso na implementação de sistemas de computação voluntária.

SETI@home é um dos mais bem sucedidos projectos de computação voluntária, lançado em 1999 rapidamente tornou-se na maior rede computacional jamais construída reforçando a ideia de que esta forma de computação poderá ser usada para vários outros projectos científicos, que demandam grandes capacidades de processamento. Actualmente este projecto conta com um total de 1.102.688 participantes e com um poder de computação que ronda os 728,857 TERAFLOPS [Bstats], consegue reunir maior poder computacional que o mais rápido super computador existente, o *Cray Jaguar* que consegue atingir uma performance 2,331 TERAFLOPS [TOP500].

Sistemas de computação voluntária são utilizados para resolver problemas em diversas áreas da ciência tais como: astronomia, física, química, biologia molecular, medicina, matemática, estudo do clima e de jogos de estratégia. A maioria destes projectos implementa estes sistemas recorrendo a plataforma BOINC que funciona como um sistema *middleware*, esta ferramenta permite criar e implementar projectos de computação voluntária, um investigador com habilidades limitadas com computadores pode criar e operar um grande projecto com aproximadamente uma semana de trabalho inicial e uma hora semanal dedicada a manutenção.

Os principais aspectos da computação voluntária, acessibilidade e aplicabilidade, podem ser considerados as principais vantagens sobre outras formas de meta-computação. Tornando os projectos acessíveis é a solução para angariar o maior número possível de voluntários e a aplicabilidade permite demonstrar a utilidade e a importância destes sistemas de computação [LFS01].

2.3 Conclusão

A computação distribuída é uma área em constante evolução e ascensão no presente momento tem vindo a ganhar também o seu espaço no sector empresarial, na comunidade científica está mais do que provado a sua potencialidade muitos projectos envolvendo a implementação de sistemas de computação distribuída, como a Grid e computação voluntária provaram que esta forma de computação tem as condições necessárias para ser uma alternativa fiável a grandes investimentos em infra-estruturas de computação de elevado desempenho.

Sistemas de computação distribuída como a Grid, computação em nuvem e computação voluntária apresentam várias vantagens quanto ao poder de computação, armazenamento e recursos que podem disponibilizar e uma boa relação custo-eficiência. Sistemas desta envergadura podem providenciar um alto-desempenho a um baixo custo, o que os torna bastante atractivos para o sector comercial e instituições universitárias, que tem como objectivo reduzir ao máximo os custos, sem perder a qualidade dos seus serviços.

Apesar de todas estas vantagens implementar tais sistemas apresenta alguns desafios como por exemplo, a escalabilidade, a heterogeneidade, a segurança, as tecnologias de processamento, as infra-estruturas de rede (protocolos e interfaces de rede), dificultam o processo de desenvolvimento. Por esta razão a comunidade científica têm-se debruçado sobre estas questões de forma a encontrar soluções fiáveis, sendo que uma delas passa pelo desenvolvimento de sistemas distribuídos cada vez mais flexíveis e adaptáveis as condições do ambiente onde estão inseridos.

3 Tecnologias

3.1 BOINC

BOINC é uma plataforma ou um *middleware* para a implementação de sistemas de computação voluntária [Dpa04] desenvolvido pelo laboratório de ciências espaciais da universidade de Berkeley pelo mesmo grupo que desenvolveu e ainda gere o projecto SETI@home. Esta ferramenta é considerada um complemento dos sistemas *Grid* que suportam a partilha de recursos entre instituições, mas não computação pública.

Actualmente é utilizada em projectos científicos ligados as áreas da física, química, biologia molecular, medicina, astrofísica, matemática, e em estudos relacionados com o clima e jogos de estratégia. Existem mais de 80 projectos baseados nesta plataforma com um aglomerado de 1.979.052 voluntários em que somente 312.822 estão activos, com um poder de computação que ronda os 5.400 TERAFLOPS. Sendo que o mais popular e com mais créditos é o projecto SETI@home [APstats] (ver Figura 3.1).

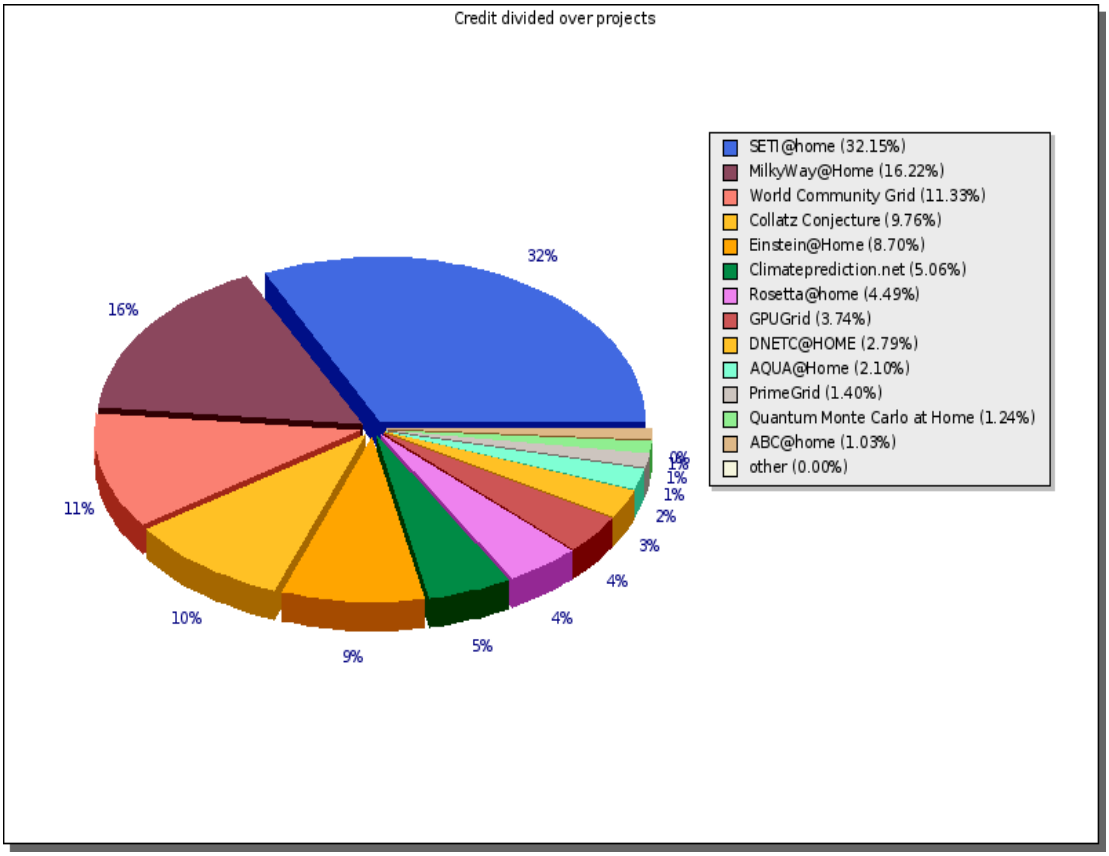

Figura 3.1: Crédito dividido pelos projectos

Fonte: http://www.allprojectstats.com/po.php?projekt=0

3.1.1 Funcionamento

Este *software* é o mais indicado [Dpa03] para projectos autónomos em que o objectivo principal é ter o maior número possível de voluntários a processar informação. Por esta razão a implementação de um projecto utilizando esta ferramenta deve ser fácil de concretizar e deve ser acessível ao maior número o possível de participantes, independente do sistema operativo utilizado ou tipo de conexão a Internet.

Um servidor para um projecto BOINC pode consistir em uma simples máquina configurada com *software open-source*, tais como Linux, Apache, PHP, MySQL e Python.

BOINC disponibiliza um mecanismo flexível e escalável para distribuição de dados, e os algoritmos de escalonamento inteligente que dispõem fazem a conjugação dos requisitos do projecto com os recursos computacionais disponíveis.

Projectos ligados ao BOINC praticam uma computação independente em que resultados obtidos de um voluntário não estão dependentes dos resultados de outrem. O servidor central, ligado a um projecto, envia dados aos voluntários que executam o programa cliente que computa os dados e retorna os resultados (ver Figura 3.2).

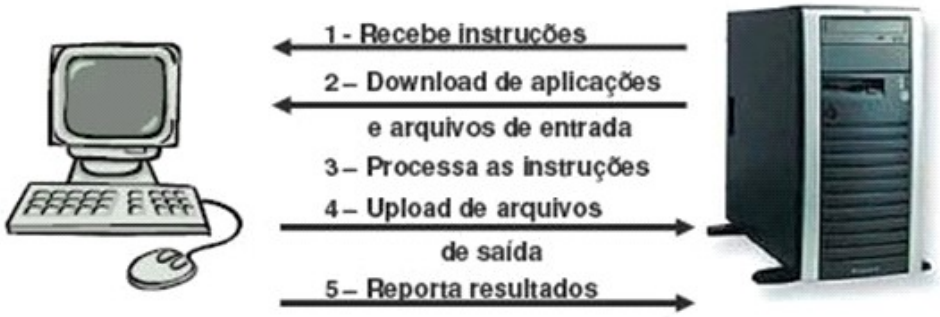

Figura 3.2: Funcionamento do BOINC.

Fonte:http://www.inf.pucrs.br/~gustavo/disciplinas/tppd1/material/TPPDI%20-%20Apresentacao%203%20-%20Jeferson%20Prevedello%20e%20Rafael%20Antonioli.pdf

3.1.2 Arquitectura

A arquitectura do BOINC consiste no modelo cliente/servidor (ver Figura 3.3). O cliente BOINC corre as aplicações dos projectos em que está anexo. Estas aplicações estão ligadas a um sistema *runtime* em que suas funções incluem controlo de processos, controlo de *checkpoint*, e animações geradas por computador (*graphics*) [DJM07].

Nesta arquitectura centralizada utilizada pelo BOINC todas as comunicações são iniciadas pelo cliente. O cliente estabelece comunicação com o servidor de tarefas de um projecto por HTTP e envia um ficheiro XML com a descrição do seu hardware e disponibilidade, com a lista de tarefas concluídas, e um pedido de trabalho adicional expresso em termos de tempo de CPU. A mensagem de resposta do servidor inclui uma lista de novas tarefas descritas por um ficheiro XML com informação da aplicação, ficheiros de entrada (*input*) e de saída (*output*), inclusive um conjunto de servidores de dados onde cada ficheiro pode ser descarregado.

A computação no BOINC é sempre anexo a um projecto BOINC que pode conter um ou mais aplicações. O cliente BOINC contacta um projecto através do seu URL. Num navegador este mesmo URL aponta para a página local do projecto, é gerado automaticamente quando o projecto é criado. Através desta página o voluntário do projecto pode verificar os seus créditos, e o administrador do projecto monitorizar a base de dados do mesmo.

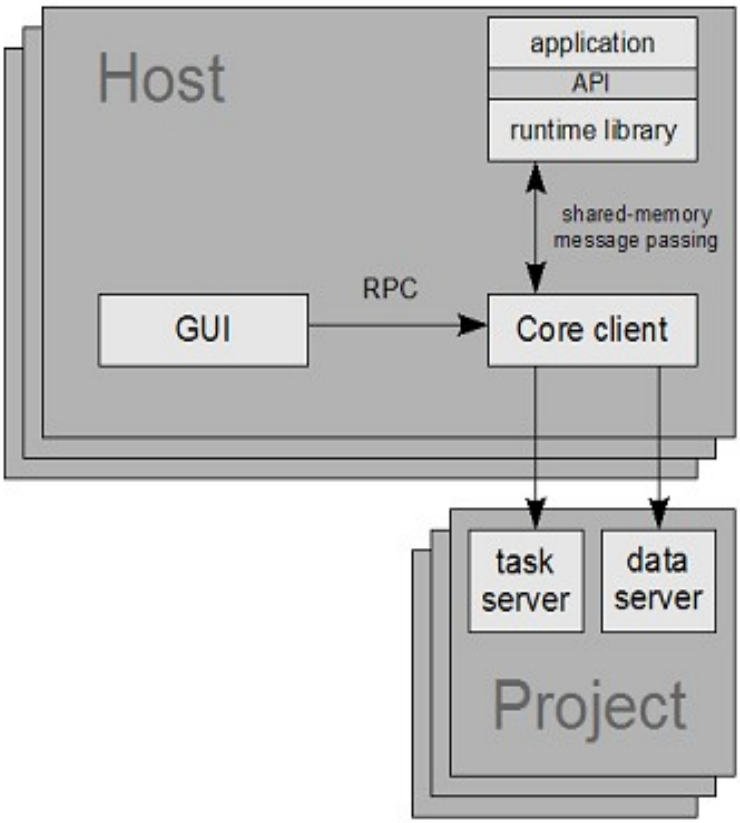

Figura 3.3: Arquitectura do BOINC.

Fonte: http://boinc.berkeley.edu/boinc_papers/sched/paper.pdf

Servidor

Cada projecto gere um servidor que consiste em três componentes [DER05]:

- Interfaces Web – para gestão de uma conta ou equipa, quadros de mensagens, e outras funcionalidades.
- Servidor de tarefas – que cria tarefas e os envia aos programas cliente e processa tarefas retornadas.
- Servidor de dados – que faz *downloads* de ficheiros *input* e executáveis, e *uploads* de ficheiros *output*.

Estes componentes partilham dados armazenados em disco, incluindo bases de dados relacionais e ficheiros de *uploads/downloads* (ver Figura 3.4).

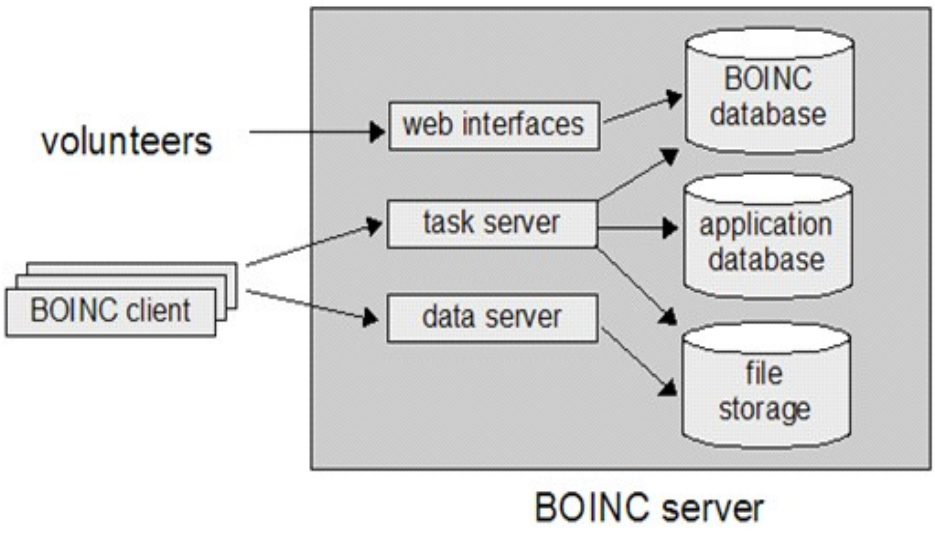

Figura 3.4: Componentes de um servidor BOINC

Fonte: http://boinc.berkeley.edu/boinc_papers/server_perf/server_perf.pdf

O servidor de tarefas pode ter vários programas em execução que utilizam a mesma base de dados. O *work generator* é o responsável por criar trabalhos (*jobs*) no servidor [DER05]. Por exemplo, o *workgenerator* do projecto SETI@home faz a leitura de dispositivos de armazenamento, recebidos por correio do observatório Arecibo [DJE02] e faz a transferência dos dados para ficheiros de *input*, e cria os *jobs* na base de dados BOINC. O programa *validator* faz a validação através da comparação dos resultados recebidos de diferentes clientes para o mesmo *job*. Em seguida atribui os créditos, pelo resultado correcto, ao computador e o seu utilizador. O programa *assimilator* processa os dados validados. Este programa pode mover os resultados para uma pasta específica do servidor, ou então ele pode processar os resultados e guardar ele mesmo os dados na base de dados.

A figura 3.5 demonstra o processo de computação de um sistema baseado no BOINC:

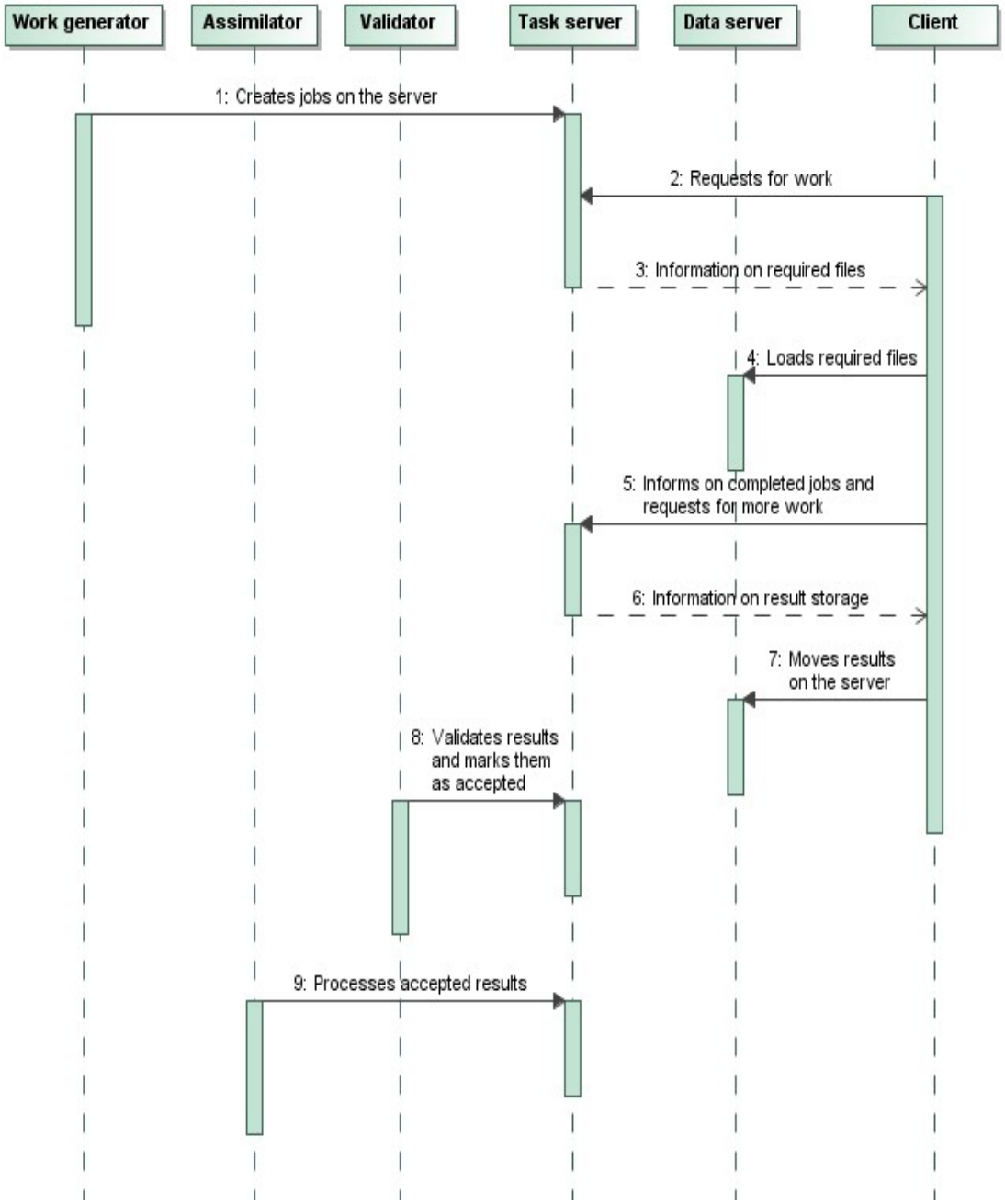

Figura 3.5: Processo de computação no BOINC

Fonte: http://math.tut.fi/en/wp-content/uploads/2009/10/report95.pdf

Cliente

O cliente BOINC que é instalado nos computadores pessoais dos voluntários consiste na realidade em um pacote com vários programas separados (ver Figura 3.5):

- *Core Client* – programa responsável por comunicar com os servidores de dados e *schedulers* dos projectos via protocolos de comunicação HTTP, por reportar e pedir tarefas. Executa e controla as aplicações dos projectos em que está anexo.
- *GUI ou BOINC Manager* – programa que disponibiliza ao utilizador uma interface gráfica que lhe permite controlar o *Core Client*, como por exemplo, suspender as aplicações. A comunicação com o *Core Client* é feito por uma ligação TCP normalmente local, mas é possível controlar o *Core Client* remotamente.
- *Screensaver* – é executado quando o utilizador não está a utilizar o computador, comunica-se com o *Core Client* através de uma conexão TCP, instruindo-lhe para gerar gráficos de protecção de ecrã. Nem todos os projectos disponibilizam este programa.

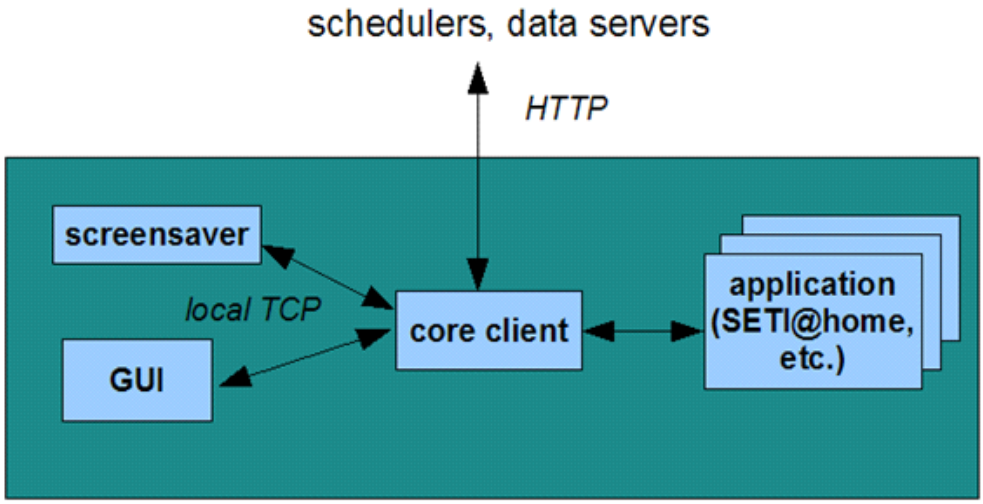

Figura 3.6: Programas que constituem o cliente BOINC

Fonte: http://boinc.berkeley.edu/wiki/How_BOINC_works

3.1.3 Desempenho

A performance dos sistemas de computação voluntária utilizando este *software* está indirectamente ligada a relação entre a performance dos servidores e o número de voluntários por projecto. Utilizando somente um servidor, um projecto BOINC pode despachar cerca de 8.8 milhões de tarefas por dia. Se a cada cliente lhe for atribuído uma tarefa por dia e cada tarefa utilizar 12 horas CPU em um computador de 1 GFLOPS, o projecto pode suportar 8.8 milhões de clientes e obter 4.4 PETA FLOPS de poder de computação. Com mais dois servidores adicionais um projecto BOINC pode despachar cerca de 23.6 milhões de tarefas por dia. Estes

dados foram retirados de um artigo de David P. Anderson, um dos fundadores do projecto SETI@horme, Eric Korpela e Rom Walton, em que fazem um estudo da performance de servidores BOINC [DER05].

O desempenho de um servidor pode limitar a capacidade de computação disponível para um projecto de computação voluntária. Caso se verifique sobrecarga do servidor o pedido de tarefas dos clientes falha e algum poder de computação é desperdiçado.

O mecanismo redundante para impedir falsificação de dados pode também ser considerado uma desvantagem, enviar os mesmos dados para pelos 3 voluntários faz com que algum poder de computação seja perdido, praticamente 1/3 dos resultados são aproveitados. Apesar que em sistemas controlados esta opção de segurança pode ser desactivada no ficheiro de configuração do projecto.

Projectos que se baseiam no paralelismo de dados em que os ficheiros de input não são muito grandes a utilização do BOINC é a melhor alternativa do que a utilização de super computadores, como por exemplo, o projecto Climateprediction.net [Cpred] em que o objectivo é desenvolver um modelo avançado de prognóstico do clima, em que milhares de dados são processados por diferentes clientes de forma independente executando a mesma operação.

3.1.4 Segurança e Fiabilidade

Diversos mecanismos de segurança e fiabilidade foram desenvolvidos e adicionados ao BOINC, entre eles merece destaque:

- O mecanismo redundante para impedir falsificação de resultados.
- A assinatura de código para impedir a distribuição de aplicações forjadas.
- Limite do tamanho máximo do arquivo de saída para impedir ataques do tipo negação de serviço (*Denial of Service*).
- A possibilidade de múltiplos servidores redundantes a falha.
- A codificação de programas e dados com *hash* de chave privada.
- A disponibilização de chave pública para clientes BOINC criadas no acto de criação de uma conta num determinado projecto.

3.2 Condor

Condor é um *software* que permite criar um ambiente, normalmente chamado de HTC (High-Throughput Computing), capaz de disponibilizar um grande poder de computação tolerante a falhas durante um longo período de tempo. Este poder de computação é criado graças a habilidade do Condor em aproveitar de forma eficiente, recursos distribuídos de redes privadas, como por exemplo, redes de *workstations* de instituições públicas e privadas. Faz uso da computação oportunista que é a habilidade de utilizar todos os recursos sempre que estiverem disponíveis, e não precisam estar disponíveis a 100%.

Este *software* foi desenvolvido pelo projecto Condor que começou em 1988 na Universidade de Wisconsin-Madison, foi construído com base nos resultados do projecto Remote-Unix (RU) dirigido pelos professores D. Dewitt, R.Finkel, e M. Solomon e como continuação de estudos feitos, na área de Gestão de Recursos Distribuídos (DRM), pelo grupo dirigido pelo professor M. Livny.

As principais preocupações do projecto tem sido com os consumidores com grande necessidades a nível de computação e ambientes com recursos heterogéneos distribuídos. O projecto Condor "herdou" uma vasta colecção de mecanismos e um *software* com base muito sólida do projecto RU. O núcleo das políticas de gestão de recursos foi providenciado pelas técnicas de escalonamento "preemptivas" e alocação distribuída desenvolvidas como parte da tese de doutoramento do professor M. Mutka do grupo DRM [Cproject].

3.2.1 Funcionamento

Este *software* funciona como um sistema eficiente de gestão de trabalhos e recursos, um utilizador submete uma tarefa numa máquina em que o Condor esteja instalado e ele encontra uma máquina disponível na rede para executar a tarefa. Caso uma máquina não possa terminar uma tarefa por não estar disponível, o Condor consegue migrar a tarefa para uma outra que esteja disponível e continuar a tarefa exactamente onde parou, não há necessidade de iniciar novamente a tarefa, ele guarda o estado das tarefas fazendo pontos de verificação (*checkpoint*) durante a execução [DTM05, DTM03, MLM88].

Esta gestão eficiente de trabalhos e recursos é feita recorrendo aos seguintes mecanismos:

- *ClassAds* : mecanismo que disponibiliza uma plataforma extremamente flexível e expressiva cujo objectivo é conciliar pedidos de recursos (tarefas) com ofertas de recursos (máquinas).
- *Job checkpoint and migration*: mecanismo que permite guardar o estado de uma tarefa em um ficheiro, desta forma é mais fácil migrar uma tarefa para outra máquina que esteja disponível ou que seja mais rápida a executar a tarefa.
- *Remote System Calls*: quando um trabalho faz uma chamada ao sistema, como por exemplo uma função de input ou output, os dados são mantidos na máquina onde se submeteu o trabalho e não na máquina remota onde pode haver imposições de execução.

Com estes mecanismos, o Condor pode fazer mais do que apenas gerir *clusters* dedicados de computação, ele consegue também recolher e gerir poder de computação desperdiçado pelos computadores de trabalho em toda a organização sem o mínimo esforço. Por exemplo, o Condor pode ser configurado para correr tarefas ou trabalhos em *workstations* somente quando o teclado ou o CPU não estiver a ser utilizado, se o utilizador voltar a utilizar o teclado, o trabalho é migrado para outra máquina e o trabalho começa onde ficou antes de ser interrompido [DTM03].

Estes mecanismos também permitem ao Condor suportar escalonamento baseado em prioridade em *clusters* dedicados de computação. Quando um nó do *cluster* não tem tarefas agendadas para serem executadas, o Condor aproveita este nó de forma oportunista, mas quando uma reserva agendada requisitar o nó, qualquer trabalho de computação oportunista que foi colocado no nó é parado pelo Condor.

O Condor é utilizado para combinar todo o poder computacional de uma organização em um só recurso.

Uma outra funcionalidade interessante do Condor é a utilização de um grafo acíclico dirigido (DAG) para casos em que os trabalhos devem ser ordenados de acordo com as dependências entre eles, cada trabalho corresponde a um nó no grafo. Estes trabalhos são submetidos no Condor seguindo as dependências descritas no grafo.

Em sistemas que utilizam este *software*, o administrador tem total controlo sobre a máquina, têm prioridade sobre o uso da máquina. Não precisa realizar qualquer acção para ganhar o controlo da máquina o Condor faz isso automaticamente [Cman].

Apesar de o Condor correr qualquer tipo de processo ainda tem algumas limitações em trabalhos em que ele não pode fazer de forma transparente *checkpoint* e migração [Cman]:

1. Trabalhos de múltiplos processos (Multi-process) não são permitidos. Incluem chamadas ao sistema tais como: *fork( ), exec( )* e *system( ).*
2. Comunicação entre processos (IPC) não é permitida. Incluem *pipes*, semáforos, e memória partilhada.
3. Comunicação na rede deve ser breve. Algum trabalho pode fazer conexões de rede utilizando chamadas ao sistema como *socket( ),* mas uma conexão de rede deixada aberta por longos períodos de tempo vai atrasar o *checkpointing* e a migração.
4. Enviar e receber sinais SIGUSR2 ou SIGTSTP não é permitido. O Condor reserva estes sinais para uso próprio.
5. Não são permitidos chamadas ao sistema como *alarm( ), getitimer ( )* e *sleep( ).*
6. Não são permitidas múltiplas linhas de execução ao nível do núcleo (KLT).
7. Não são permitidos chamadas ao sistema como *mmap( )* e *munmap( ).*
8. Bloqueio de ficheiros é permitido, mas não quando retido entre *checkpoints.*

9. Todos os ficheiros devem ser abertos somente para leitura ou para escrita. Um ficheiro aberto para leitura e escrita irá causar problema caso uma tarefa precise ser revertida para uma imagem antiga de *checkpoint.*
10. Deve haver espaço suficiente no disco da máquina onde se submete os trabalhos para armazenamento das imagens de *checkpoint*. Estas imagens consomem a mesma quantidade de memoria que o trabalho consome quando executado utilizando memória virtual. Caso necessário por falta de espaço no disco, um servidor especial de *checkpoint* pode ser adicionado ao sistema para armazenar todas a imagens de *checkpoint* de um *Condor pool*[3].
11. Escrita e leitura de ficheiros com mais do que 2 GB não é suportado.

Estas limitações somente se aplicam se o *checkpointing* transparente dos trabalhos for exigido, caso contrário as limitações não têm efeito.

3.2.2 Arquitectura

Sistemas que utilizam Condor baseiam-se numa arquitectura [DTM03] de *master/workers* em que o mestre (*master*) distribui tarefas que deverão ser executados pelos trabalhadores (*workers*) e o Condor faz a gestão de recursos e tarefas (ver Figura 3.7).

Cada máquina no *Condor pool* pode desempenhar uma variedade de funções sendo que algumas delas só podem ser executadas por uma única máquina. A lista seguinte descreve estas funções e que recursos são necessários para que este serviço seja providenciado [Cman]:

- Gestor Central (*master*) – Só pode haver uma máquina com esta função, têm a responsabilidade de recolher informação e de ser o negociador entre recursos e pedidos de recursos. Desempenha um papel importante no sistema por esta razão deve ser fiável, se esta máquina avariar toda as negociações de recursos são suspensas. Convêm ela estar preparada para estar ligada sempre, ou se no caso avaria, reiniciar rapidamente. Uma boa conexão de rede as outras máquinas do sistemas também é recomendada, elas enviam os *updates* através da rede para o gestor central.
- Executar (*worker*) – Qualquer máquina no sistema incluindo o Gestor Central pode desempenhar a função de executar ou não trabalhos do Condor. Não requer qualquer recurso a não ser espaço de armazenamento no disco, sendo que se o trabalho remoto descarregar o núcleo, este ficheiro é descarregado primeiro no disco local da máquina que executa o trabalho antes de ser enviado a máquina que submeteu o trabalho. Caso não houver espaço suficiente, o Condor simplesmente vai reduzir o tamanho do núcleo do ficheiro a ser descarregado. Em geral quando maior for os recursos de uma máquina (espaço *swap*, memória real, velocidade do CPU, etc) maior é o número de pedidos de recurso que ele pode satisfazer.
- Submeter – Qualquer máquina no sistema incluindo o Gestor Central pode desempenhar a função de submeter ou não trabalhos do Condor. Requisito de

3 Uma organização de recursos que podem ser partilhados

recursos nestas máquinas são maiores do que as máquinas que executam os trabalhos. Para cada trabalho submetido que é executado numa máquina remota um processo é gerado e mantido na máquina que o submeteu, por esta razão é preciso uma quantidade aceitável de espaço *swap* e/ou memória real no caso de muitos trabalhos serem executados. E ainda há os ficheiros de *checkpoin*t que também são mantidos por esta máquina o que pode exigir uma grande quantidade de espaço no disco, mas pode ser aliviado recorrendo a um servidor de *checkpoint,* apesar de que os ficheiros binários dos trabalhos submetidos continuam a ser armazenados por esta máquina.

- Servidor *checkpoint* - Uma máquina do sistema pode desempenhar esta função, é opcional e não faz parte da configuração padrão da distribuição binária do Condor. O servidor *checkpoint* é uma máquina centralizada que armazena todos os ficheiros de *checkpoint* do sistema. Por esta razão esta máquina deve ter bastante espaço no disco e uma boa conexão de rede com as outras máquinas do sistema, o tráfico de rede pode ser intensivo.

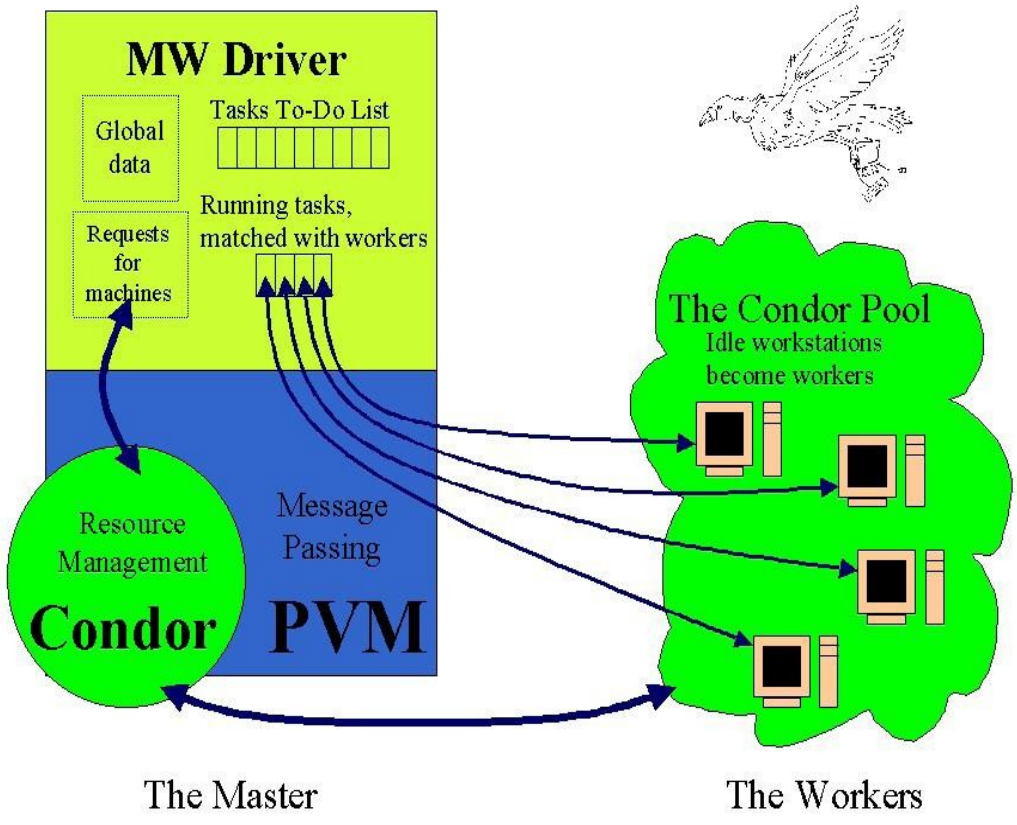

Figura 3.7: Arquitectura do Condor

Fonte: http://www.cs.wisc.edu/condor/slides/API/apis.pdf

A ferramenta MW, apresentada na figura 3.7, é responsável por fazer com que uma aplicação com o estilo *master/workers* funcione num ambiente distribuído e oportunista do Condor.

3.2.3 Desempenho

A performance do Condor está de certa forma dependente dos recursos computacionais disponíveis no sistema e do seu algoritmo de *matchmaking* que é responsável por fazer a conciliação dos trabalhos submetidos com as máquinas do sistema com base em requisitos e preferências.

Todos os trabalhos do Condor são submetidos através de um ficheiro de configuração chamado de *submition description file.* Neste ficheiro é definido o universo em que o programa deve correr e os requisitos mínimos necessários para que uma máquina remota possa executar o trabalho.

Para tirar o melhor partido do Condor é preciso saber como configurar o ficheiro com a descrição da submissão. Por exemplo, convém saber os universos que existem no Condor, eles são os responsáveis por controlar a forma como o Condor lida com os trabalhos submetidos [TTA08].

Universos mais utilizados no Condor:

- Vanilla – permite a execução de quase todos trabalhos a serem executados em série e providência a transferência automática de ficheiros. Pode ser utilizada em quase todas as situações.
- Parallel – permite a execução de trabalhos que utilizam MPI, trabalhos a serem executados em paralelo.
- Standard – permite adicionar tolerância a falhas aos trabalhos submetidos, suporta o processo transparente de *checkpoint* e reiniciar. Suporta também chamadas remotas ao sistema (remote I/O), os trabalhos podem ler e escrever em ficheiros como se fossem locais. Para que este universo funcione o trabalho deve ser conectado as bibliotecas de suporte deste universo. As limitações deste universo incluem as referidas anteriormente quando se faz de forma transparente o checkpoint.
- Grid – permite submeter trabalhos na Grid desde que tenha acesso a estes recursos. Suporta um número de tipos de "back end": Globus (GT2, GT4), NorduGrid, UNICORE, PBS, plataforma LSF, EC2, NQS.
- Scheduler – permite utilizar a funcionalidade do Condor, DAGman, que permite implementar dependências entre os trabalhos submetidos recorrendo a um grafo acíclico dirigido em que cada nó corresponde a um trabalho.
- Java – permite executar trabalhos em java, até saber quais as máquinas que tem uma JVM instalada assim como a localização, a versão instalada e a performance.

No caso dos requisitos do CPU serem muito exigentes e recursos disponíveis estarem sempre a ser requisitados por outros processos com maior prioridade o utilizador pode sempre recorrer a um *cluster* dedicado e também adicionar um gestor central dedicado, o Condor suporta facilmente a adição de mais recursos. Pode adicionar um ou mais *Condor pool,* o Condor disponibiliza o mecanismo *Flocking* que permite fazer isso facilmente.

Como já foi referido nos universos Condor pode-se aumentar a *performance* do sistema se tiver acesso e credenciais para aceder a um sistema Grid ou outros recursos em instituições remotas.

Para não comprometer a execução de um trabalho deve-se ter atenção quando da configuração do ficheiro de submissão, alguns trabalhos podem ficar suspensos (*idle*) se algum trabalho for impedido de executar em determinadas máquinas por não haver memoria RAM suficiente, ou se não houver espaço no disco, ou então um *software* qualquer instalado. Estes factores devem ser definidos no ficheiro de configuração.

3.2.4 Segurança e Fiabilidade

A segurança é um aspecto importante a considerar na implementação de um sistema com o Condor. O principal propósito do Condor é permitir que utilizadores executar código arbitrário no maior número o possível de máquinas, por esta razão é importante limitar o acesso a uma *Condor pool* e também aos seus privilégios quanto da utilização do mesmo [Cman, 3.6 – Security].

Para fazer este controlo de acesso e privilégios o Condor utiliza a biblioteca CEDAR que é uma multi-plataforma C++ TCP e UDP como representação de *socket* de rede que permite a clientes e servidores negociar e usar diferentes protocolos de autenticação tais como: Kerberos, GSI, chaves públicas, autenticação via redes ou nós confiáveis e sistema de ficheiros. Implementa um conjunto de interfaces de autenticação simples e extensível, possui a capacidade de negociar a integridade dos dados e algoritmos de privacidade separado do protocolo de autenticação.

A autenticação no Condor baseia-se num sistema de ficheiros, um processo *daemon* utiliza a propriedade do ficheiro para identificar um utilizador que queira entrar no *Condor pool,* o processo solícita a escrita de um ficheiro em um directório de escrita temporária e comparando o proprietário do ficheiro e nome que o utilizador indicou, ele decide a autorização.

Quando da execução de uma tarefa no Condor algumas informações são conhecidas unicamente pela máquina de execução, tais como quais sistemas de ficheiros, redes e bases de dados podem ser acedidos, enquanto que somente a máquina de submissão conhece em tempo de execução os recursos necessários para a tarefa. Essa cooperação conhecida como *split execution* é realizada no Condor pelos componentes *shadow* e *sandbox*.

O mecanismo *shadow*, utilizado para protecção dos utilizadores, no site de submissão, providencia tudo o que é necessário para a especificação da tarefa em tempo de execução: o executável, os argumentos, o ambiente, arquivos de entrada, etc. Nenhum deles é conhecido fora da máquina de execução até o momento real da execução.

O *sandbox* utilizado na máquina de execução deve proteger os proprietários das máquinas, impedindo acesso indevido aos recursos, é formado por dois componentes distintos:

- *Sand* – responsável por criar o ambiente apropriado à execução da tarefa ou trabalho.
- *Box* – responsável pela protecção dos recursos de possíveis danos que uma tarefa maliciosa possa causar.

Actualmente, as tarefas são executadas sem restrições no sistema de ficheiros, mas com restrições no *login*, ao contrário do que acontecia em versões iniciais em que restringia a execução de tarefas a uma parte sistema de ficheiros através do comando *chroot* do Unix.

Para prevenir que dados sejam lidos e acedidos por intrusos o Condor pode encriptar os dados enviados pela rede, assim como dados do utilizador como ficheiros e executáveis, providenciando a privacidade dos dados. A encriptação é executada no acto de transferência de dados pela rede, os dados são desencriptados e armazenados no disco. A encriptação pode ser feita recorrendo ao 3DES [RD78] e BlowFish [BS94].

Para prevenir ataques como *man-in-the-middle*, o Condor providencia a integridade dos dados, envia informação adicional encriptada para fazer a verificação. Algoritmos como o MD5 [RLR] podem ser utilizados na verificação de integridade.

A fiabilidade dos resultados é garantida pelo mecanismo de *checkpoint*, uma tarefa pode ser resumida no caso da máquina de execução estar ocupada, avariada ou a executar uma tarefa de maior prioridade. Estes ficheiros são criados periodicamente de acordo com as preferências do utilizador, são a garantia que o trabalho sempre termina mesmo em caso de falhas. É um mecanismo muito importante em ambientes de computação oportunista [Cman, 3.8 – The Checkpoint Server].

3.3 Hadoop

Hadoop é a popular implementação *open source* do MapReduce que foi desenvolvido pelo Google, uma ferramenta poderosa concebida para análises e transformações profundas de conjuntos enormes de dados. Esta ferramenta permite explorar dados complexos utilizando uma análise adequada a informação e questões relacionadas com o projecto em estudo.

Este *software* foi desenvolvido por Doug Cutting, o criador de Apache Lucene, a tão conhecida biblioteca de pesquisa de texto. Hadoop é baseado no Apache Nutch, um motor de pesquisa *open source*, também parte do projecto Apache Lucene.

O Hadoop começou a ganhar mais destaque em Janeiro de 2008 quando se tornou um dos principais projectos do Apache, confirmando o seu sucesso, e sua activa e diversa comunidade,

que muito contribui para a sua popularidade. Nesta altura muitas empresas para além da Yahoo já utilizavam o Hadoop como por exemplo [PoweredBy]:

- Last.fm – cluster constituído por 44 nós, com processadores dual quad-core Xeon L5520 (Nehalem) 2.27 Gigahertz, 4 TeraByte disco por nó, utilizado no cálculo de gráficos, análise de ficheiros log e testes A/B (testes de marketing).
- Facebook – o Hadoop é utilizado para armazenar cópias de log interno e dimensão de *data sources*, são usados como fonte informação para análises/relatórios e aprendizagem automática. Actualmente mantém dois clusters, um com 1100 máquinas com 8800 *cores* e cerca de 12 Peta bytes de armazenamento não processado (*raw storage)* e um outro com 300 máquinas com 2400 *cores* e cerca de 3 Peta bytes de armazenamento não processado.
- New York Times – utiliza o EC2 para correr o Hadoop num enorme *cluster* virtual para extrair de 4 Terabytes de ficheiros digitalizados, artigos a serem convertidos em formato pdf para serem colocados na Internet. A correr em 100 máquinas o processo demorou menos de 24 horas.

O Hadoop surgiu como resposta ao problema de armazenamento e análise de dados enfrentado actualmente. Apesar do grande aumento da capacidade de armazenamento dos discos a velocidade de acesso aos dados no disco não alterou muito como por exemplo, um disco de 1Terabyte a velocidade de transferência de dados ronda os 100 Megabyte/s o que demora em média 2 horas e meia a fazer leitura de todos os dados e a escrita é ainda mais lenta. A solução óbvia seria ler múltiplos discos de uma só vez, trabalhando em paralelo com os dados distribuídos por 100 discos a leitura dos dados demoraria cerca de 2 minutos.

O problema de armazenamento é resolvido pelo HDFS e o da análise de dados pelo MapReduce. Estas são as principais capacidades oferecidas pelo Hadoop um sistema fiável de armazenamento partilhado e de análise [TW09].

3.3.1 Funcionamento

Projectos de computação voluntária podem ser ligeiramente semelhantes ao MapReduce quanto a forma como os problemas são abordados (dividir um problema em pequenos pedaços que são analisados em paralelo), mas existem algumas diferenças. MapReduce foi concebido para executar tarefas que duram alguns minutos ou horas em hardware confiável e dedicado de um único *data center* com ligações de rede de elevada largura de banda, enquanto que projectos de computação voluntária executam tarefas em máquinas, que não são confiáveis utilizando a Internet [TW09, JV09].

O MapReduce é um modelo de programação para o processamento de dados em que o Hadoop se baseia, consiste em dividir o processamento em duas fases distintas [ver Figura 3.8]:

1. *Map* – Passo inicial de transformação, em que registos individuais do *input* podem ser processados em paralelo, é responsável de transformar o *input* em pares de registos <chave, valor>.
2. Reduce – Passo que faz o resumo e agregação, todos os registos associados devem ser processados juntos por uma única entidade, com base nos pares de registos gerados pelo Map esta função gera o ficheiro output.

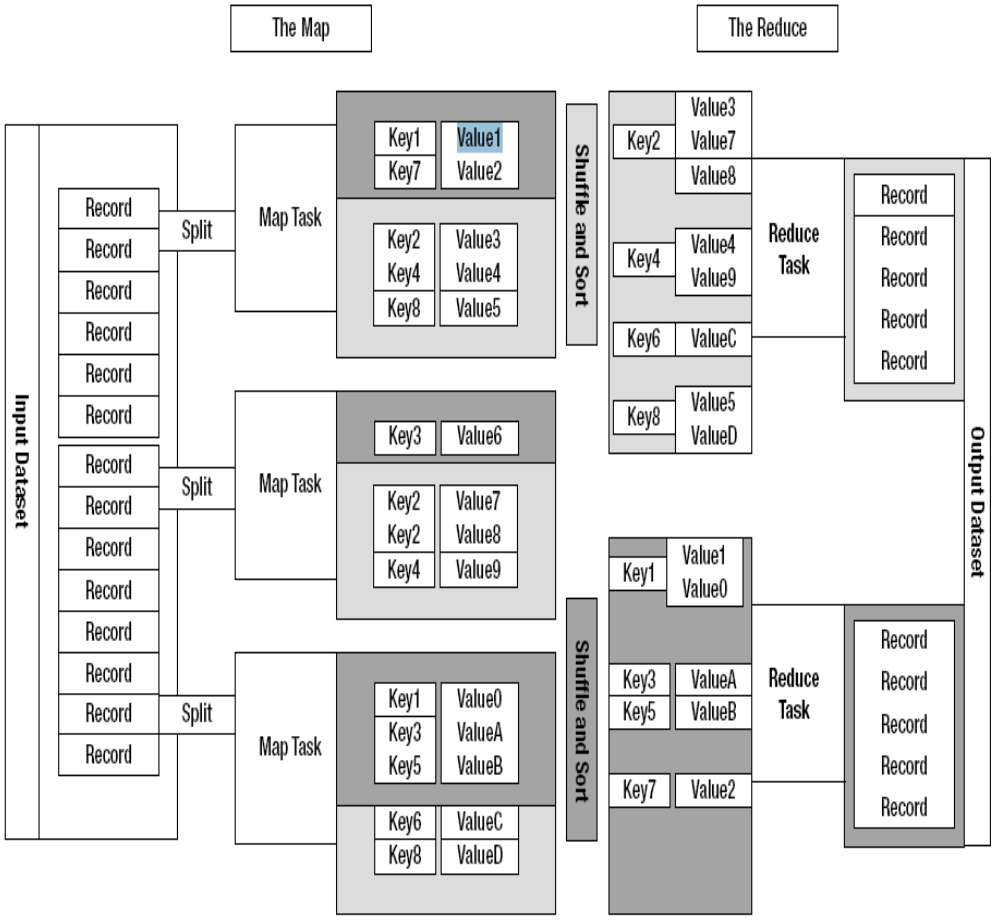

Figura 3.8: Modelo MapReduce

Fonte: Jason Verner, Pro Hadoop, Apress, 2009

3.3.2 Arquitectura

Utiliza uma arquitectura *master/slaves* [ASF10] para implementar o MapReduce. Esta plataforma disponibiliza dois processos responsáveis pela gestão de trabalhos MapReduce:

- TaskTrackers que geram a execução individual das tarefas map e reduce num nó de computação do cluster.

- JobTracker que aceita a submissão de trabalhos disponibiliza monitorização e controlo de um trabalho e gere a distribuição de tarefas para os nós TaskTracker.

Cada trabalho é coordenado por um único JobTracker que é responsável por escalonar as tarefas nos nós *slave* e por acompanhar o progresso destas tarefas em conjunto com o *slave* TaskTrackers que é executado em cada nó *slave*.

A gestão dos ficheiros de input e output é feito pelo sistema de ficheiros distribuídos do Hadoop (HDFS) e os seus serviços são providenciados por dois processos (ver Figura 3.9);

- *NameNode* – responsável pela gestão do espaço de nomes (*metadata*) do sistema de ficheiros, e pela gestão e controlo dos serviços.
- *DataNode* – responsável por providenciar blocos de armazenamento e recuperação de serviços.

Internamente, os arquivos armazenados no HDFS são divididos em blocos de dados, que são atribuídos e mapeados aos *DataNodes* pelo *NameNode*. Operações de abertura, fecho e renomeação de ficheiros e directórios são feitas apenas pelo *NameNode*. Operações de leitura e escrita em ficheiros são feitas directamente pelos *DataNodes*, quando um cliente necessita alterar um bloco de dados de um ficheiro, não é necessário requisitar a operação ao *NameNode*. Além disso, os *DataNodes* são responsáveis por realizar operações de criação, eliminação e replicação de blocos de ficheiros, porém apenas sob a orientação do *NameNode* [BTK09].

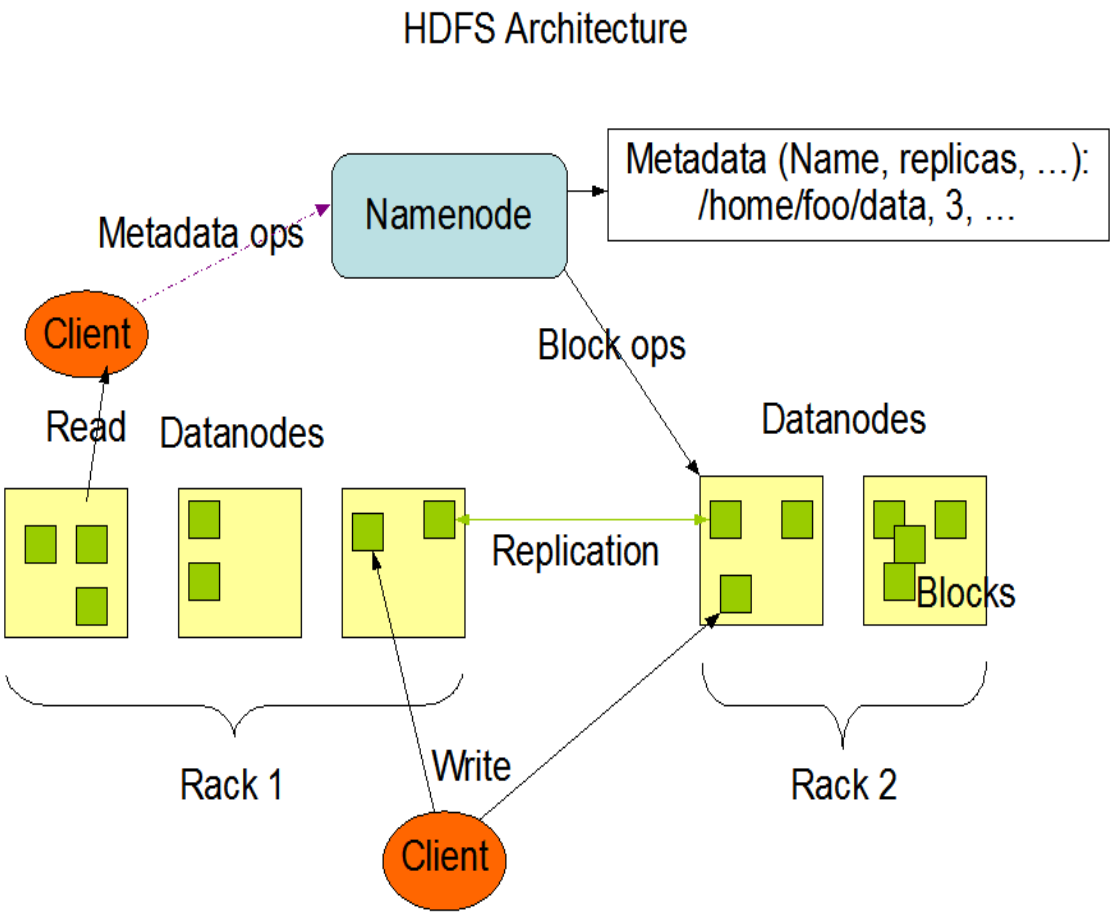

Figura 3.9: Arquitectura do HDFS

Fonte: http://hadoop.apache.org/common/docs/current/hdfs_design.html

Só pode haver um processo *NameNode* por sistema de ficheiros HDFS enquanto que *DataNode* podem haver múltiplos processos no cluster, mas somente um processo por nó.

O JobTracker e o *NameNode* fazem o papel de *master*, enquanto que o TaskTracker e o DataNode são os *slaves* da aplicação.

3.3.3 Desempenho

O Hadoop é o *software* ideal para casos em que é necessário processar grandes volumes de dados por exemplo, na ordem dos Peta bytes. Dispõem de um sistema de ficheiros distribuído (HDFS) que lhe permite armazenar dados em milhares de servidores e executar tarefas (tarefas Map/Reduce) através do cluster. O princípio defendido pelos criadores do Hadoop como forma melhorar a performance do algoritmo MapReduce é de colocar os dados perto de onde se vai executar a tarefa, ou seja, tirando partido de uma das funcionalidades do HDFS que é a replicação eficiente de dados pelos nós do cluster podemos melhorar a performance de execução de uma tarefa se movermos os blocos de dados para o nó que vai processa-los [HW10].

Este *software* foi concebido para trabalhos que duram alguns minutos ou horas a serem executados não é o *software* adequado para problemas computação intensiva que podem durar dias até obter um resultado, ao contrário do Condor. Devidas as suas características é também utilizado em Data Mining, que é o processo de extracção de padrões de grandes quantidades de dados.

Devido ao seu alto-desempenho na análise e armazenamento distribuído de dados o Hadoop tem vindo a ser utilizado por várias empresas com serviços e vasta informação na Internet tais como: Facebook, The New York Times, Google, IBM, Last.fm, Amazon Web Services, Adobe, AOL, Cooliris, Joost, LinkeIn, Twitter, Yahoo, etc. Chegou mesmo a bater recordes em Abril e Novembro de 2008 e em Maio de 2009 como o sistema mais rápido a ordenar um Terabyte de dados [SMR08].

Hadoop pode resolver alguns problemas específicos de algumas empresas mas não é uma solução para todos os problemas, ela não pode substituir ferramentas utilizadas pelas empresas na gestão das suas bases de dados.

3.3.4 Segurança e Fiabilidade

Uma das principais características do Hadoop é o seu sistema de ficheiros distribuído concebido para dividir os dados em grandes blocos e distribui-los pelos nós de um *cluster*, blocos de dado replicados pelas máquinas como forma de prevenir a perda de dados no caso falhas no sistema [JV09].

No caso de um nó falhar os outros nós do *cluster* tem a capacidade de assumir as responsabilidades computacionais deste nó durante o tempo que for necessário, e se o nó recuperar pode-se juntar ao *cluster* sem necessidade de reiniciar todo o sistema. O Hadoop aposta na recuperação individual dos nós.

A autenticação do nó *master* nos nós *slave* é feita recorrendo a chaves públicas SSH, as chaves são geradas no nó master e enviadas aos nós *slave* que devem adiciona-las nas suas chaves autorizadas (*authorized_keys*). Sem esta forma de autenticação o nó master não conseguiria comunicar-se com os nós *slave* do cluster.

4 Resultados e Discussão

Neste capítulo fez-se a discussão dos resultados obtidos com os testes efectuados utilizando os protótipos implementados com base nas tecnologias descritas no capítulo 3, o Hadoop e Condor. No BOINC, por questões técnicas, não foi possível realizar testes apesar de o servidor estar configurado e a funcionar implementar um projecto demora algum tempo e alguns conhecimentos técnicos que devem ser adquiridos como por exemplo, conhecer bem a API do BOINC que tem as bibliotecas necessárias para gerar trabalhos, definir políticas de escalonamento, entre outras configurações.

Os testes apresentados neste tópico consistem na análise do desempenho de algoritmos utilizados na álgebra linear, no caso do Condor, e no processamento distribuído de dados, no caso do Hadoop.

4.1 Testes com o Hadoop

4.1.1 Caso estudo 1

Foi realizado um teste com o modelo de programação MapReduce para três cenários utilizando o Hadoop. O código foi todo desenvolvido em Java na plataforma NetBeans IDE versão 6.8 com base no exemplo WordCount disponibilizado na wiki do Hadoop [HWC], pode ser consultado no Anexo A.1.

Neste caso de estudo o modelo MapReduce foi utilizado para contar validações efectuadas pelos clientes de uma empresa de transporte durante um mês. A estrutura dos ficheiros de input foram baseados numa estrutura real de uma empresa de transporte mas os dados são aleatórios. Os dados de input consistem em 27 ficheiros de texto com um tamanho total de 2 Gigabytes.

Os testes foram realizados num cluster implementado com dois nós, com o Hadoop versão 0.20.1 da *Cloudera Distribution* versão 2 para sistemas Debian [CDH2]. A arquitectura deste

cluster é apresentado na figura 4.1, é constituído por um nó que desempenha o papel de *master* e *slave* e o outro que desempenha somente o papel de *slave*:

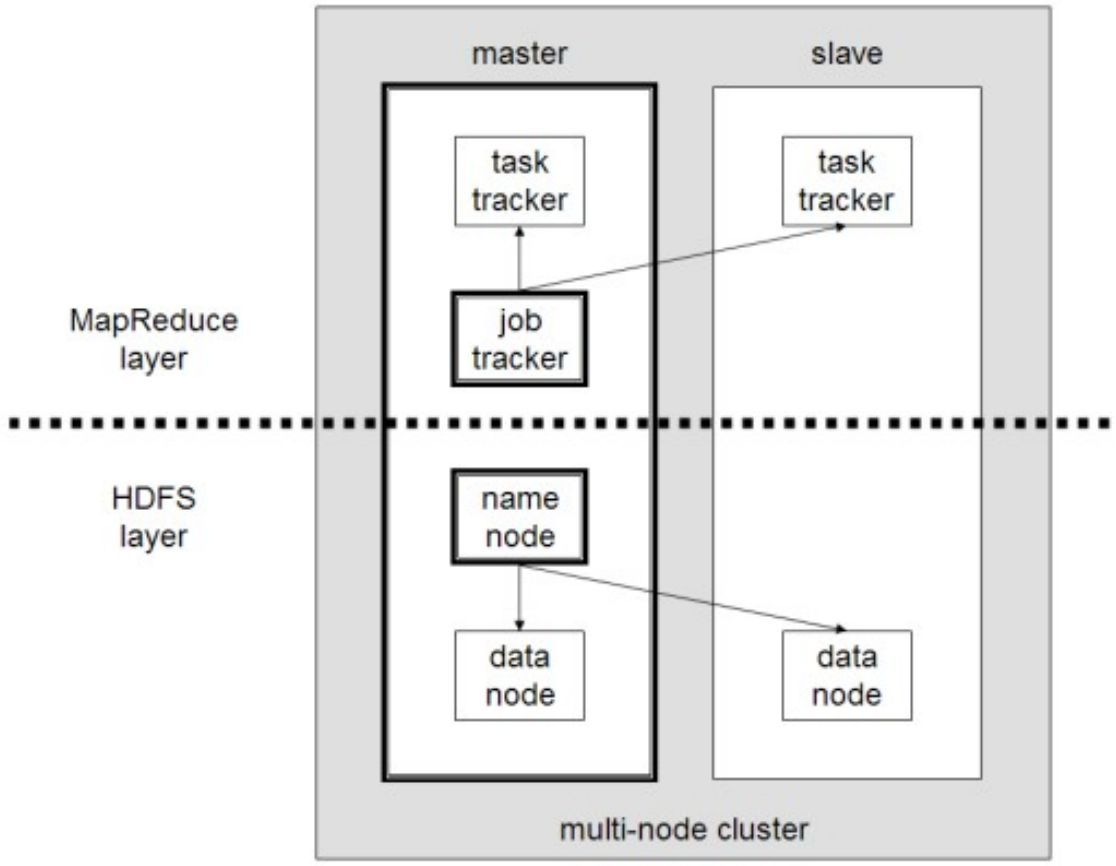

Figura 4.1: Arquitectura do Cluster Hadoop

Fonte: http://www.michael-noll.com/wiki/Running_Hadoop_On_Ubuntu_Linux_%28Multi-Node_Cluster%29

A configuração deste *cluster* foi baseada num tutorial disponibilizado na Internet por Michael G. Noll, um utilizador entusiasta do Hadoop [MGN10]. A tabela seguinte descreve o hardware das máquinas utilizadas neste cluster de 2 nós:

Tabela 4.1: Hardware das máquinas do Cluster Hadoop

Hardware	*Máquinas*
Processador	Intel Pentium 4 - 2.80 GHz
Memória RAM	2 * 512 MB
Rede	NetXtreme BCM5782 Gigabit Ethernet
Disco	160 GB

Foram definidos os seguintes cenários de teste:

- Cenário 1 – Testar o programa utilizando somente o nó *master/slave* do cluster.
- Cenário 2 – Testar o programa no *cluster* com 2 nós, um nó m*aster/slave* e um nó s*lave*.
- Cenário 3 – Testar o programa no modo *standalone* em uma máquina com processador Intel Core 2 duo T5750 2 GHz, 2 GB de memória RAM e 250 GB de disco.

A tabela 4.2 demonstra os resultados obtidos:

Tabela 4.2: Resultados do Caso de estudo 1 - Hadoop

MapReduce	*Cenário 1*	*Cenário 2*	*Cenário 3*
Nº de tarefas Map	45 tarefas / 0 tentativas falhadas	47 tarefas / 2 tentativas falhadas	68 tarefas / 0 tentativas falhadas
Nº de tarefas Reduce	1 tarefa	1 tarefa	1 tarefa
Tempo total de execução	287 segs	154 segs	274 segs
Inputs	2 Gigabytes (27 Ficheiros)	2 Gigabytes (27 Ficheiros)	2 Gigabytes (27 Ficheiros)

As figuras 4.2 e 4.3 apresentam um maior detalhe dos resultados da submissão da tarefa *ValidationCount* no *cluster* no cenário 1 e 2.

	Counter	Map	Reduce	Total
Job Counters	Launched reduce tasks	0	0	1
	Launched map tasks	0	0	45
	Data-local map tasks	0	0	45
FileSystemCounters	FILE_BYTES_READ	4,724,508	143,558,238	148,282,746
	HDFS_BYTES_READ	2,118,190,073	0	2,118,190,073
	FILE_BYTES_WRITTEN	148,284,444	143,558,238	291,842,682
	HDFS_BYTES_WRITTEN	0	5,874,083	5,874,083
Map-Reduce Framework	Reduce input groups	0	407,204	407,204
	Combine output records	0	0	0
	Map input records	7,975,457	0	7,975,457
	Reduce shuffle bytes	0	141,865,608	141,865,608
	Reduce output records	0	407,204	407,204
	Spilled Records	8,237,929	7,975,457	16,213,386
	Map output bytes	127,607,312	0	127,607,312
	Map input bytes	2,118,116,327	0	2,118,116,327
	Map output records	7,975,457	0	7,975,457
	Combine input records	0	0	0
	Reduce input records	0	7,975,457	7,975,457

Figura 4.2: Resultados ValidationCount Cenário 1

	Counter	Map	Reduce	Total
Job Counters	Launched reduce tasks	0	0	1
	Launched map tasks	0	0	47
	Data-local map tasks	0	0	47
FileSystemCounters	FILE_BYTES_READ	4,724,508	143,558,238	148,282,746
	HDFS_BYTES_READ	2,118,190,073	0	2,118,190,073
	FILE_BYTES_WRITTEN	148,284,444	143,558,238	291,842,682
	HDFS_BYTES_WRITTEN	0	5,874,083	5,874,083
Map-Reduce Framework	Reduce input groups	0	407,204	407,204
	Combine output records	0	0	0
	Map input records	7,975,457	0	7,975,457
	Reduce shuffle bytes	0	143,558,496	143,558,496
	Reduce output records	0	407,204	407,204
	Spilled Records	8,237,929	7,975,457	16,213,386
	Map output bytes	127,607,312	0	127,607,312
	Map input bytes	2,118,116,327	0	2,118,116,327
	Map output records	7,975,457	0	7,975,457
	Combine input records	0	0	0
	Reduce input records	0	7,975,457	7,975,457

Figura 4.3: Resultados ValidationCount Cenário 2

Como se pode verificar pelas figuras 4.2 e 4.3 diferenças entre os dois cenários estão no número de tarefas *Map* e no tempo de execução, a transferência de dados através do sistema de ficheiros e mesmo os registos gerados pelo *Map* e *Reduce* são os mesmos nos dois cenários.

A razão é simples, no cenário 1 o nó *slave* deixa de receber tarefas porque o processo *Tasktracker* foi parado e por esta razão todas as tarefas passaram para o nó *master/slave* que passou a ter praticamente o dobro do trabalho, mas no entanto o nó *slave* mesmo não estando a receber tarefas ainda tem o processo *DataNode* a funcionar, ou seja, o fluxo dados corre normalmente com se fossem dois nós a trabalhar, apesar de somente o nó *master/slave* executar tarefas.

```
10/06/29 03:27:26 INFO mapred.Merger: Merging 68 sorted segments (Map Tasks)
                  ...................................
                  ...................................
                  ...................................
10/06/29 03:27:59 INFO mapred.JobClient: Job complete: job_local_0001
10/06/29 03:27:59 INFO mapred.JobClient: Counters: 13
10/06/29 03:27:59 INFO mapred.JobClient:   FileSystemCounters
10/06/29 03:27:59 INFO mapred.JobClient:     FILE_BYTES_READ=75234454745
10/06/29 03:27:59 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=5229293268
10/06/29 03:27:59 INFO mapred.JobClient:   Map-Reduce Framework
10/06/29 03:27:59 INFO mapred.JobClient:     Reduce input groups=407204
10/06/29 03:27:59 INFO mapred.JobClient:     Combine output records=0
10/06/29 03:27:59 INFO mapred.JobClient:     Map input records=7975457
10/06/29 03:27:59 INFO mapred.JobClient:     Reduce shuffle bytes=0
10/06/29 03:27:59 INFO mapred.JobClient:     Reduce output records=407204
10/06/29 03:27:59 INFO mapred.JobClient:     Spilled Records=23531916
10/06/29 03:27:59 INFO mapred.JobClient:     Map output bytes=127607312
10/06/29 03:27:59 INFO mapred.JobClient:     Map input bytes=2118116327
10/06/29 03:27:59 INFO mapred.JobClient:     Combine input records=0
10/06/29 03:27:59 INFO mapred.JobClient:     Map output records=7975457
10/06/29 03:27:59 INFO mapred.JobClient:     Reduce input records=7975457
BUILD SUCCESSFUL (total time: 4 minutes 34 seconds)
```

Figura 4.4: Resultados ValidationCount Cenário 3

Confrontando os resultados obtidos no cenário 1,2 e 3 o *cluster* no cenário 2 que utiliza 2 nós apresentou uma melhor performance a processar 2GB de dados. Apesar da máquina utilizada no cenário 3 apresentar melhores recursos a nível de hardware do que as máquinas do *cluster* (Intel Core 2 duo VS Intel Pentium 4) só conseguiu apresentar melhores resultados em relação ao cenário 1 com uma ligeira diferença de 13 segundos, a taxa de transferência dos discos é quase a mesma, no computador do cenário 1 a taxa média de transferência é 58 MB/s enquanto que no cenário 3 a taxa média é de 46,9 MB/s e a taxa máxima 58,7 MB/s, por esta razão os resultados nestes cenários não apresentam grandes diferenças.

Resumindo, a escrita e leitura para o disco (I/O) fez a diferença, escrever/ler de dois discos (2 x 58 MB/s) ao mesmo tempo, fazendo uso do HDFS, do Hadoop provou ser mais rápido do que um computador utilizando somente um disco.

4.1.2 Caso estudo 2

A abordagem seguida neste caso de estudo é diferente da apresentada anteriormente, na medida que este teste tem como propósito tentar justificar o porquê de *softwares* como Hadoop, concebidos para processar grandes quantidades de dados, não são adequados para problemas que exigem maior cálculo como por exemplo, problemas de álgebra linear como é o caso do produto de matrizes.

O cenário proposto para este teste foi o seguinte: utilizar duas matrizes de pequenas dimensões (2x2) e aplicar o produto de matrizes utilizando um algoritmo já adaptado para o modelo de programação MapReduce e descrito pelo programa MatrixMul.java (Anexo A.2).

O programa desenvolvido recebe dados de um ficheiro input que foi especialmente concebido para ser processado pela estrutura MapReduce, um exemplo deste ficheiro é apresentado abaixo:

Tabela 4.3: Ficheiro de Input

```
1.0 2.0    // C11
2.0 5.0    // C11
;
1.0 3.0    // C12
2.0 6.0    // C12
=
3.0 2.0   //  C21
4.0 5.0   //  C21
;
3.0 3.0   //  C22
4.0 6.0   //  C22
```

As linhas identificadas com C11 e assim por diante representam os termos a serem multiplicados, o primeiro termo representa um elemento da linha da primeira matriz e o segundo termo um elemento da coluna da segunda matriz. Cada identificador representa um elemento da matriz resultante do produto de duas matrizes. Neste caso C11 a C22 representam elementos da matriz como por exemplo, C11 é o elemento da 1ª linha e 1ª coluna da matriz final.

Com base na estrutura do MapReduce foram implementadas duas classes estáticas, *map* e *reduce*.

O *map* faz a leitura linha a linha guardando em memória o produto dos dois termos por linha e os atribui uma chave (key) que é o identificador, os dados são guardados numa estrutura do tipo <*chave, valor*>, como por exemplo <C11, 1.0 * 2.0>, a chave é C11 e o valor é o produto de 1 por 2.

O *reduce* seria o responsável pela última parte do algoritmo que é a soma de todos os valores, guardados na estrutura <*key, value*>, com a mesma chave. Seguindo o exemplo anterior, todos os dados guardados em memória com a chave C11 seriam somados (1.0*2.0 +

2.0*5.0) obtendo assim um elemento da matriz resultante. O ficheiro de saída com os resultados do exemplo apresentado anteriormente:

Tabela 4.4: Ficheiro de *output*

C11	12.0
C12	15.0
C21	26.0
C22	33.0

O processo não é complicado mas preparar o ficheiro input para ser processado pelo Hadoop não é uma tarefa fácil principalmente com matrizes de grandes dimensões. Ainda assim pode-se criar um programa que faz o pré-processamento da matriz para se adaptar a estrutura MapReduce, mas mesmo assim não é o ideal. O Hadoop foi concebido para processamento distribuído de grandes quantidades de dados e não é o mais adequado para álgebra linear, o seu modelo de programação muito rigoroso não permite fugir à sua estrutura inicial em que o *map* prepara os dados para serem processados pelo *reduce*.

4.2 Testes no Condor

Os casos de estudo analisados a seguir foram testados num *cluster* Condor com as seguintes características:

- Número de máquinas - 20
- Número de máquinas virtuais - 74
- Máquinas virtuais com Java - 4
- Sistema Operativo - Linux
- Arquitectura - Intel
- Processadores - Intel Pentium 4 Processor 630 3.0 GHz
- Memória RAM - 512 MB
- Capacidade do disco - 80 GB

4.2.1 Caso de estudo 1

No Condor os testes abordados foram mais direccionados para problemas em que se podia testar o paralelismo de dados.

Neste caso de estudo é calculado o Speedup obtido quando se corre *n* vezes o programa Produto de matrizes quadradas desenvolvido em C (Anexo B.1) no Condor. O programa gera duas matrizes de dimensão 1024 x 1024 com valores aleatórios e calcula o produto de ambos e retorna o resultado.

No *cluster* onde o programa foi testado 17 nós estavam configurados para funcionar como 4 máquinas virtuais e as restantes 3 como duas máquinas virtuais com um total de 74 máquinas virtuais. Por este motivo para podermos utilizar o equivalente a um processador o número de processos tem de ser igual a quatro. Na tabela a seguir é apresentado o ficheiro de submissão da tarefa no Condor:

Tabela 4.5: Ficheiro de submissão no Condor - Produto de Matrizes

```
Executable = matrixm    // nome do executável gerado pelo gcc
Log            = matrixmul.log  //  Ficheiro de log gerado
Output       = out.$(Process)   // Ficheiro output por processo
Error          = error.$(Process)  // Ficheiro de erro por processo
when_to_tranfer_output = ON_EXIT  //  Condição : Ficheiro output é
                                  // transferido ao terminar a execução
Queue  n     // nº de processos gerados
```

O tempo de execução médio do programa utilizando um processador é igual a 51.2 segundos, com base neste resultado é calculado o Speedup (4.1):

$$Sp = \frac{T_1}{T_p} \qquad (4.1)$$

- p é o número de processadores.
- T_1 é o tempo de execução do algoritmo sequencial.
- T_p é o tempo de execução do algoritmo em paralelo com p processadores.

A tabela seguinte apresenta os resultados obtidos para a matriz quadrada 1024 x 1024 em que a “Unidade” representa o número de processadores utilizados:

Tabela 4.6: Resultado do cálculo do Speedup

Unidade	*Nº de processos*	*Speedup*
2	8	1,83
4	16	3,2
8	32	6,5
10	40	6,32
12	48	5,91
14	56	6,96
16	64	7,52

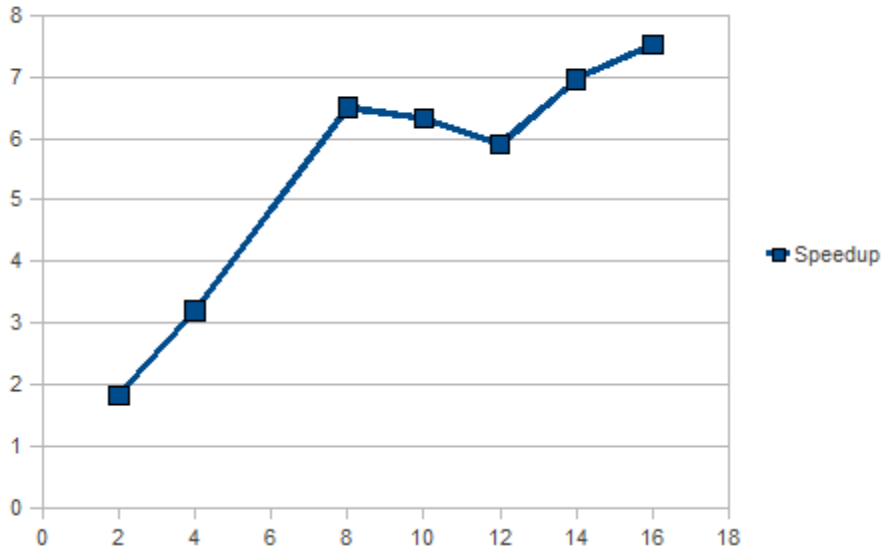

Figura 4.5: Gráfico da Tabela 4.6

Os resultados obtidos demonstram que o *Speedup* não é linear como seria de esperar, há algumas variações entre a unidade 4 e 12. Ao executar os processos entre a unidade 4 e 12 verificou-se algum atraso em alguns deles, à medida que se aumentava o número de processos tornou-se mais visível um ligeiro atraso no *matchmaking* do Condor apesar de haver recursos disponíveis o *daemon* demorava algum tempo a atribuir o processo a uma máquina virtual e como consequência isto atrasava o tempo global da tarefa. O mesmo acontece no caso de estudo seguinte.

4.2.2 Caso de estudo 2

Neste teste é calculado o *Speedup* do programa Eliminação de Gauss desenvolvido em C (Anexo B.2). O programa calcula aleatoriamente uma matriz aumentada (1024 x 1025) e retorna a solução do sistema de equações lineares por eliminação de Gauss.

Na tabela a seguir é apresentado o ficheiro de submissão da tarefa no Condor:

Tabela 4.7: Ficheiro de submissão no Condor - Eliminação de Gauss

```
Executable = gauss    // nome do executável gerado pelo gcc
Log          = gauss.log  //  Ficheiro de log gerado
Output       = out.$(Process)   // Ficheiro output por processo
Error        = error.$(Process)  // Ficheiro de erro por processo
when_to_tranfer_output = ON_EXIT  //  Condição : Ficheiro output é
                                  // transferido ao terminar a execução
Queue  n     // nº de processos gerados
```

A tabela seguinte apresenta os resultados obtidos:

Tabela 4.8: Resultado do cálculo do Speedup

Unidade	*Nº de processos*	*Speedup*
2	8	1,6
4	16	2,29
8	32	3,2
10	40	3,2
12	48	3,43

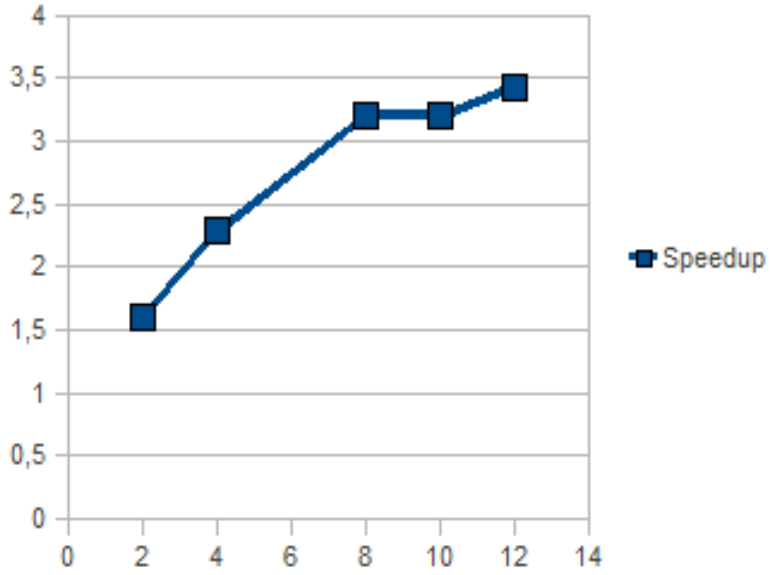

Figura 4.6: Gráfico da Tabela 4.8

Como se pode verificar pelo gráfico aconteceu o mesmo que o caso de estudo anterior o Speedup também não é linear neste caso. Ao analisar o ficheiro log gerado a cada execução pode-se constatar que nem todos os processos começam ao mesmo tempo alguns começam mais tarde e há casos em que alguns começam a ser executados primeiro e terminam em último. Os testes foram sempre feitos tendo em conta os recursos disponíveis estando 69 máquinas virtuais disponíveis foram gerados nos dois casos de estudo com um máximo de 64 processos.

4.3 Análise Comparativa dos softwares

A análise comparativa das tecnologias descritas no capítulo 3 foi feita recorrendo a uma matriz de comparação. A matriz compara os *softwares* BOINC, Condor e Hadoop com base em características que definem um sistema de distribuído, a análise feita neste tópico é um resumo de tudo o que já foi referido no capitulo 3 e nos testes já realizados.

Tabela 4.9: Matriz de comparação

	Tecnologias		
Características	*BOINC*	*Condor*	*Hadoop*
Arquitectura	Cliente / Servidor	*Master / Worker*	*Master / Slave*
Escalabilidade	Muito Boa	Fraca	Boa
Tipo de Paralelismo	Paralelismo de dados	Paralelismo de dados	Paralelismo de dados
Fiabilidade	Boa	Boa	Muito Boa
Segurança	Muito Boa	Boa	Boa

Escalabilidade

A escalabilidade é uma das características mais importantes de um sistema distribuído e pode ser medida segundo três dimensões diferentes (Neuman, 1994):

1. Um sistema pode ser escalável segundo o seu tamanho, é fácil adicionar mais utilizadores e recursos ao sistema.
2. Pode ser escalável em termos geográficos na medida que os recursos e os utilizadores podem estar longe uns dos outros.
3. Pode ser escalável em termos administrativos, pode ser fácil de gerir, mesmo que abranja muitas organizações administrativas diferentes.

Com base nestas três dimensões fez-se uma análise da escalabilidade das três tecnologias apresentadas no decorrer deste estudo.

O BOINC apresenta uma escalabilidade muito boa a nível de tamanho é fácil adicionar mais um voluntário a um projecto BOINC, o voluntário só tem de instalar um programa cliente e conectar-se ao projecto pretendido, um servidor BOINC também é facilmente escalável segundo David P. Anderson [DER05]. Em termos geográficos e administrativos o BOINC também é escalável facilmente os recursos podem estar longe uns dos outros e em organizações administrativas diferentes, deste que tenha o cliente instalado ligado a Internet, ainda continua a ser contactado para executar trabalhos e um servidor pode ser facilmente replicado para uma localidade mais próxima do utilizador.

O Condor têm a desvantagem de estar “preso” a redes privadas, a escalabilidade em termos geográficos e administrativos é difícil de conseguir mas em termos de tamanho é fácil adicionar

mais recursos, o Condor dispõem do mecanismo *Flocking* que permite adicionar um *Condor pool* facilmente.

O Hadoop apresenta uma escalabilidade boa em termos de tamanho um nó *slave* pode ser facilmente adicionado ao *cluster* mas a dimensão do cluster está dependente da capacidade do nó *master* em aceitar mais nós *slave,* em sistemas implementados com o Hadoop só pode haver um nó *master*. Em termos geográficos é difícil de conseguir escalabilidade; o HDFS utilizado pelo Hadoop requer proximidade entre os recursos e uma rede rápida para facilitar o fluxo de dados entre os nós. As máquinas que partilham o mesmo HDFS não podem estar em domínios diferentes, é mais complicado administrar o HDFS em casos de diferentes domínios administrativos.

5 Conclusões e Trabalho Futuro

5.1 Conclusões

No decurso da dissertação foi realizado um estudo sobre o paradigma da computação distribuída e as tecnologias BOINC, Condor e Hadoop, foram implementados *clusters* para o Hadoop e Condor e configurado um servidor para o BOINC.

Das tecnologias analisadas a plataforma BOINC é a que permite implementar verdadeiros sistemas de computação voluntária, possui ferramentas que permitem implementar um servidor para um projecto e gerar milhões de trabalhos que podem ser executados por voluntários do projecto durante o período de tempo em que os seus computadores pessoais não estejam a ser utilizados.

O Condor é um bom candidato para sistemas privados de computação voluntária, dispõem das ferramentas necessárias para implementar computação voluntária em sistemas controlados ao contrário da Internet, é o mais indicado para redes privadas como as de instituições universitárias.

O MapReduce, utilizado pelo Hadoop, pode ser ligeiramente semelhante aos projectos de computação voluntária quanto a forma como os problemas são abordados, paralelismo de dados, mas existem diferenças. Os sistemas que utilizam o Hadoop foram concebidos para executar tarefas que duram alguns minutos ou horas em hardware confiável e dedicado de um *cluster* com ligações de rede de elevada largura de banda entre os nós. Este *software* é o mais indicado para problemas que requerem o processamento distribuído de conjuntos enormes de dados, por esta razão é muito utilizada na análise de dados, prospecção de dados, extracção de conhecimento.

As instituições universitárias podem recorrer a estes *softwares* para implementar sistemas de computação distribuída e voluntária para aproveitar os seus recursos computacionais para trabalhos de investigação em diversas áreas científicas, tanto o BOINC e o Condor podem ser

utilizados para casos em que tipo de processamento é o paralelismo de dados e os dados de input, que podem estar em ficheiros ou ser gerados pelo programa, não são muito extensos. O Hadoop por outro lado pode ser utilizado para casos em que é necessário processar uma grande quantidade de dados, nestas situações justifica-se ter um *cluster* que possa receber um conjunto enorme de dados e processa-los em horas ou minutos.

O BOINC pode apresentar o mesmo desempenho que o Condor em redes privadas, uma vez que há um maior controlo da segurança em sistemas privados o mecanismo redundante de verificação de resultados pode ser retirado da configuração dos projectos BOINC, ou seja, dados para serem computados podem ser enviados para um único voluntário não há necessidade de verificar os resultados.

A solução proposta consiste em ter um cluster para o Condor e um servidor BOINC para casos que envolvam paralelismo de dados e computação intensiva como por exemplo, produto de matrizes de grandes dimensões, simulações de *Monte Carlo*, algoritmos de encriptação, etc, e ter um cluster dedicado para o Hadoop em casos que é necessário processar conjuntos enormes de dados de forma eficiente.

5.2 Trabalho Futuro

A realização desta dissertação foi um desafio, na medida que o tema computação voluntária era desconhecido pelo autor, era algo novo, uma forma diferente de definir a computação distribuída.

Esta dissertação serviu para assimilar um novo conceito, que irá permitir as instituições universitárias com as ferramentas adequadas conciliar a suas necessidades computacionais com os recursos que têm disponível.

Finalmente tem-se como objectivo introduzir estes serviços, disponibilizados pelas tecnologias testadas ao longo desta dissertação, nas instituições universitárias para serem utilizados por alunos, docentes e investigadores. Posteriormente seria interessante verificar o impacto que a introdução destes serviços teria no desenvolvimento dos seus projectos fazendo um levantamento estatístico da utilização e aplicabilidade.

Referências

[APstats] All Project Stats. *Estatisticas de todos os projectos BOINC*, Junho 2010. http://www.allprojectstats.com/po.php?projekt=0

[ASF10] The Apache Software Foundation, *Hadoop*, Janeiro 2010. http://lucene.apache.org/hadoop.

[ASM] Andrew S. Tanenbaum and Maarten Van Stenn, *Distributed systems: principles and paradigms (2nd Edition)*, Prentice Hall, 2006.

[Bstats] BOINCstats. *Estatísticas da plataforma BOINC*, Junho 2010. http://br.boincstats.com/stats/project_graph.php?pr=sah

[BS94] B. Schneier, *Description of a new variable-length key, 64-bit block cipher (blowfish). In Fast Software Encryption*, Cambridge Security Workshop Proceedings, pp. 191–204. Springer-Verlag, 1994.

[BTK09] D.Borthakur, *Hdfs architecture*, Dezembro 2009. http://hadoop.apache.org/core/docs/current/hdfs_design.html.

[CDH2] Cloudera, *Installing CDH2 on Debian Systems*, Março 2010. https://docs.cloudera.com/display/DOC/Installing+CDH2+on+Debian+Systems.

[Condor] Condor, Dezembro 2009. http://www.cs.wisc.edu/condor/

[Cpred] Climateprediction.net, *The world's largest climate forecasting experiment for the 21st century*, Dezembro 2009. http://climateprediction.net/

[Cproject] Condor Project, *How did the Condor project start?*, Dezembro 2009. http://www.cs.wisc.edu/condor/background.html

[Cman] Condor Project, *CondorVersion 7.5.2 Manual*, Janeiro 2010, http://www.cs.wisc.edu/condor/manual/v7.5/1_3Exceptional_Features.html

[DER05] David P. Anderson, Eric Korpela, Rom Walton, *High-Performance Task Distribution for Volunteer Computing*, First IEEE International Conference on e-Science and Grid Technologies. Melbourne, 2005.

[DJE02] David P. Anderson, Jeff Cobb,Eric Korpela, Matt Lebofsky, Dan Werthimer, *SETI@home: An Experiment in Public-Resource Computing*, Communications of the ACM, vol.45, nº 11, 2002.

Referências

[DJM07] David P. Anderson, John McLeod VII, *Local Scheduling for Volunteer Computing, Workshop on Large-Scale*, Volatile Desktop Grids (PCGrid 2007) held in conjunction with the IEEE International Parallel & Distributed Processing Symposium (IPDPS), Long Beach., USA, 2007.

[Dpa03] David P. Anderson, *Public Computing: Reconnecting People to Science*, Conference on Shared Knowledge and the Web. Residencia de Estudiantes, Madrid, Spain, 2003.

[Dpa04] David P. Anderson, *BOINC: A System for Public-Resource Computing and Storage*, 5th IEEE/ACM International Workshop on Grid Computing4, Pittsburgh, USA, 2004.

[DTM03] Douglas Thain, Todd Tannenbaum, and Miron Livny, *Condor and the Grid*, in Fran Berman, Anthony J.G. Hey, Geoffrey Fox, editors, *Grid Computing: Making The Global Infrastructure a Reality*, John Wiley, 2003. ISBN: 0-470-85319-0

[DTM05] Douglas Thain, Todd Tannenbaum, and Miron Livny, *Distributed Computing in Practice: The Condor Experience*, Concurrency and Computation: Practice and Experience, vol. 17, nº 2-4, 2005.

[FGT03] F. Berman, G. Fox, T. Hey, *Grid Computing: Making the Global Infrastructure a Reality*, John Willey, 2003.

[HS04] Salim Hariri and Manish Parashar, *Tools and environments for parallel and distributed*, John Willey, 2004.

[HWC] Hadoop Wiki, *WorkCount Example*, Novembro 2009 http://wiki.apache.org/hadoop/WordCount.

[HW10] Hadoop Wiki, *ProjecDescription*, Março 2010. http://wiki.apache.org/hadoop/ProjectDescription

[ICK03] I. Foster, C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure (2nd Edition)*, Morgan Kaufmann, 2003.

[ICS01] I.Foster, C. Kesselman, Steven Tuecke, *The anatomy of Grid: Enabling Scalable Virtual Organizations*, International Journal of Supercomputer Applications, vol. 15, 2001.

[IFT02] I. Foster, *What is the grid? - a three point checklist*, GRIDtoday, vol. 1, nº 6, 2002.

[JV09] Jason Venner, *Pro Hadoop*, Apress, 2009.

[LFS01] Luis F. G. Sarmenta, *Volunteer Computing*, Ph.D. thesis. Dept. of Electrical Engineering and Computer Science, MIT, 2001.

[MGN10] Michael G. Noll, *Running Hadoop On Ubuntu Linux (Multi-Node Cluster)*, Março 2010. http://www.michael-noll.com/wiki/Running_Hadoop_On_Ubuntu_Linux_%28Multi-Node_Cluster%29

[MLM88] Michael Litzkow, Miron Livny, and Matt Mutka, *Condor - A Hunter of Idle Workstations*, Proceedings of the 8th International Conference of Distributed Computing Systems, 1988.

[PCS04] Beth Plale, C. Jacobs, S. Jensen, Y. Liu, C. Moad, R. Parab, P. Vaidya, *Understanding Grid Resource Information Management through a Synthetic Database Benchmark/Workload*, Fourth IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2004), 2004.

[Pwby] Hadoop Wiki, *PoweredBy*, Fevereiro 2010. http://wiki.apache.org/hadoop/PoweredBy

[RD78] R. Davis, *The data encryption standard in perspective*, Communications Magazine, IEEE, Vol. 16, nº 6, pp. 5-9. 1978.

[RLR] Ronald L. Rivest. *The MD5 message-digest algorithm (RFC 1321)*. http://www.ietf.org/rfc/rfc1321.txt

[SMR08] *Sorting 1PB with MapReduce*, 21 Novembro 2008. http://googleblog.blogspot.com/2008/11/sorting-1pb-with-mapreduce.html

[TOP500] TOP500. *TOP500 List - June 2010*, Junho 2010. http://www.top500.org/list/2010/06/100

[TTA08] Todd Tannenbaum, *Tutorial: Basic Introduction to Condor*, Paradyn/Condor Week, 2008.

[TW09] TomWhite, *Hadoop – The Definitive Guide*, O'Reilly, 2009.

[ZST10] Zach Miller, Dan Bradley, Todd Tannenbaum, Igor Sfiligio, *Flexible Session Management in a Distributed Environment*, Journal of Physics: Conference Series, Vol. 219, Issue 4. 2010.

Referências

Anexo A: Testes efectuados no Hadoop

A.1 Código fonte ValidationCount,java

```
import java.io.IOException;
import java.util.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class ValidationCount {

    public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
            String line = value.toString();
            String id_val = null;
            StringTokenizer tokenizer = new StringTokenizer(line);
            int i = 0;
```

```
            tokenizer.nextToken();
            tokenizer.nextToken();
            word.set(tokenizer.nextToken());
            output.collect(word, one);
        }
    }
     public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterator<IntWritable> values,
            OutputCollector<Text, IntWritable> output, Reporter reporter)
            throws IOException {
            int sum = 0;
            while (values.hasNext()) {
                sum += values.next().get();
            }
            output.collect(key, new IntWritable(sum));
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf conf = new JobConf(ValidationCount.class);
        conf.setJobName("validationcount");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setMapperClass(Map.class);
        conf.setReducerClass(Reduce.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));
        try {
            JobClient.runJob(conf);
```

```
            } catch (IOException e) {
                System.err.println(e.getMessage());
            }
        }
    }
```

A.2 Código fonte MatrixMul.java

```
/**
 *
 * @author Girson
 */
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class MatrixMul {

    public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, DoubleWritable> {

        private final static DoubleWritable one = new DoubleWritable(1);
        private Text word = new Text();
        int i = 0;

        public void map(LongWritable key, Text value, OutputCollector<Text, DoubleWritable> output, Reporter reporter) throws IOException {
            String line = value.toString();
```

```
            StringTokenizer tokenizer = new StringTokenizer(line);
            int ch = 0;
            double ft = 0.0;
            String token = null;

            while (tokenizer.hasMoreTokens()) {
               token = tokenizer.nextToken();

               if (token.compareTo(";") == 0) {
                  i++;
               } else {
                  if (ch == 0) {
                     ft = Double.parseDouble(token);
                     ch = 1;
                  } else {
                     one.set(ft * Double.parseDouble(token));
                     word.set("C" + i);
                     output.collect(word, one);
                  }
               }
            }
         }
      }

         public static class Reduce extends MapReduceBase implements Reducer<Text, DoubleWritable, Text, DoubleWritable> {

                  public void reduce(Text key, Iterator<DoubleWritable> values, OutputCollector<Text, DoubleWritable> output, Reporter reporter) throws IOException {
            int sum = 0;
            while (values.hasNext()) {
```

```
            sum += values.next().get();
        }
        output.collect(key, new DoubleWritable(sum));
    }
  }

  public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(MatrixMul.class);
    conf.setJobName("matrixmul");

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(DoubleWritable.class);
    conf.setMapperClass(Map.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);
    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);
    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));
    JobClient.runJob(conf);
  }
}
```

Anexo B: Testes Efectuados no Condor

B.1 Código fonte MatrizMul.c

```
#include <stdio.h>

#define DIM 1024
unsigned int matrix_1[DIM][DIM];
unsigned int matrix_2[DIM][DIM];
unsigned int matrix_3[DIM][DIM];
int main () {
    int i, j, k;
    /* inicializacao da primeira matriz */
    for (i=0; i<DIM; i++) {
            for (j=0; j<DIM; j++) {
                    matrix_1[i][j] = 12+rand();
            }
    }
    /* inicializacao da segunda matriz */
    for (i=0; i<DIM; i++) {
            for (j=0; j<DIM; j++) {
                    matrix_2[i][j] = 12+rand();
            }
    }
    /* inicializacao da terceira matriz */
    for (i=0; i<DIM; i++) {
            for (j=0; j<DIM; j++) {
```

```
                matrix_3[i][j] = 0;
            }
        }
        /* cálculo do produto de duas matrizes quadradas */
        for (i=0; i<DIM; i++) {
            for (j=0; j<DIM; j++) {
                for (k=0; k<DIM; k++) {
                matrix_3[i][j] = matrix_3[i][j] + (matrix_1[k][j] * matrix_2[i][k]);

                }
                printf("%.2d\t", matrix_3[i][j]);

            }
            printf("\n");
        }
        return 0;
}
```

B.2 Código fonte EliminacaoGauss.c

```
#include <stdio.h>

#define DIM 1024
float a[DIM][DIM+1];

void forwardSubstitution() {
    int i, j, k, max;
    float t;
    for (i = 0; i < DIM; ++i) {
        max = i;
        for (j = i + 1; j < DIM; ++j)
            if (a[j][i] > a[max][i])
                max = j;

        for (j = 0; j < DIM + 1; ++j) {
            t = a[max][j];
            a[max][j] = a[i][j];
            a[i][j] = t;
        }

        for (j = DIM; j >= i; --j)
            for (k = i + 1; k < DIM; ++k)
```

```
                    a[k][j] -= a[k][i]/a[i][i] * a[i][j];
        }
    }
    void reverseElimination() {
        int i, j;
        for (i = DIM - 1; i >= 0; --i) {
                a[i][DIM] = a[i][DIM] / a[i][i];
                a[i][i] = 1;
                for (j = i - 1; j >= 0; --j) {
                        a[j][DIM] -= a[j][i] * a[i][DIM];
                        a[j][i] = 0;
                }
        }
    }
    void gauss() {
        int i, j;
        forwardSubstitution();
        reverseElimination();
        for (i = 0; i < DIM; ++i) {
                for (j = 0; j < DIM + 1; ++j)
                        printf("%.2f\t", a[i][j]);
                printf("\n");
        }
    }
    int main(int argc, char *argv[]) {
        int i, j;
        for (i = 0; i < DIM; ++i)
                for (j = 0; j < DIM + 1; ++j)
                        a[i][j] = rand()+rand();
        gauss();

        return 0;
    }
```