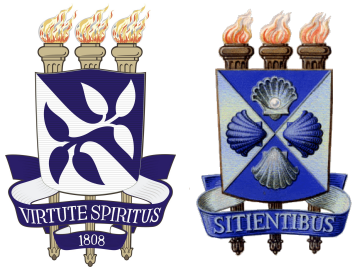

Universidade Federal da Bahia

Universidade Estadual de Feira de Santana

Dissertação de Mestrado

Provisão de computação voluntária e intensiva de dados para suporte de aplicações científicas

Felipe Oliveira Gutierrez

Mestrado Multiinstitucional em Ciência da Computação - MMCC

Salvador - BA
Fevereiro de 2014

Felipe Oliveira Gutierrez

Provisão de computação voluntária e intensiva de dados para suporte de aplicações científicas

Dissertação apresentada ao Instituto de Matemática da Universidade Federal da Bahia, para a obtenção de Título de Mestre em Ciência da Computação, na Área de Sistema Distribuídos.

Orientador: Prof. Dr. Marcos Ennes Barreto

Gutierrez, Felipe Oliveira.
Provisão de computação voluntária e intensiva de dados para suporte de aplicações científicas
89 páginas
Dissertação (Mestrado) - Instituto de Matemática da Universidade Federal da Bahia. Departamento de Ciência da Computação.

1. Sistemas distribuídos
2. Computação intensiva de dados
3. Computação voluntária
4. e-Science
5. Computação em nuvem

I. Universidade Federal da Bahia. Instituto de Matemática. Departamento de Ciência da Computação.

Salvador
Fevereiro de 2014

Comissão Julgadora:

Prof. Dr. Alirio Santos Sá

Prof. Dr. Rodrigo Barban Zucoloto

Prof. Dr. Marcos Ennes Barreto

Dedico esta dissertação de mestrado aos professores que me orientaram durante o curso, aos colegas de mestrada com os quais cursei as disciplinas e principalmente aqueles que pretendem seguir na linha de pesquisa de sistemas distribuídos.

Epígrafe

O que é bonito?
É o que persegue o infinito;
Mas eu não sou
Eu não sou, não...
Eu gosto é do inacabado,
O imperfeito, o estragado, o que dançou
O que dançou...
Eu quero mais erosão
Menos granito.
Namorar o zero e o não,
Escrever tudo o que desprezo
E desprezar tudo o que acredito.
Eu não quero a gravação, não,
Eu quero o grito.
Que a gente vai, a gente vai
E fica a obra,
Mas eu persigo o que falta
Não o que sobra.
Eu quero tudo que dá e passa.
Quero tudo que se despe,
Se despede, e despedaça.
O que é bonito...

Lenine e Bráulio Tavares

Agradecimentos

Agradeço ao meu orientador professor Dr. Marcos Ennes Barreto pelo apoio na pesquisa, pelo comprometimento com o projeto que envolveu a pesquisa e o sucesso na publicação de artigos relacionados com esta dissertação. Agradeço ao professor Dr. Rodrigo Barban Zucoloto pelo apoio nos trabalhos e métricas envolvendo as aplicações científicas utilizadas nesta dissertação, bem como pelas várias horas de trabalho contínuo e desgastante, porém com resultados relevantes para a análise e publicação de artigos em congressos internacionais. Agradeço também ao professor Dr. Alirio Santos Sá pelo apoio a pesquisa, com sugestões de bibliografias, várias vezes com apoio técnico, e a oportunidade de integrar o grupo de pesquisa sobre computação em nuvem no LaSiD (Laboratório de Sistemas Distribuídos) da UFBA. Agradeço a minha família, que apesar de não conhecer muito sobre o tema da minha pesquisa, eles são as pessoas que tenho mais intimidade e liberdade para compartilhar as minhas dificuldades e ajudam a passar por vários obstáculos semeando valores de perseverança, temperança, sabedoria, paciência, ética, comprometimento. Agradeço também aos meus colegas de turma, com os quais cumpri disciplinas no mestrado e realizei trabalhos em grupos, e também pelas participações de cada um nas salas de aula, contribuindo para o compartilhamento de diversos conhecimentos e experiências, tanto profissionais, educacionais, como de grandes valores de amizade. Agradeço aos amigos que não fizeram parte do dia-a-dia na UFBA, mas de alguma forma estiveram presentes na minha vida durante a realização deste mestrado, oferecendo incentivo e apreciação pela vontade de obter conhecimentos relevantes na universidade.

Abstract. *The project mc^2 (My sCientific Cloud), which incorporates the present proposal, aims to provide a platform to support the execution of scientific applications on computational resources from different paradigms: aggregates, peer-to-peer networks, voluntary and large-scale systems. In this scope, this work discusses the inclusion of volunteer computing resources and data intensive processing platform mc^2, along with analysis of performance metrics for some case studies. Satisfactory results were get with the both modules cited of the mc^2 platform in these analyses. Through them it was possible to prove easily that scientists from different areas can get advantage of the platform and gain in performance when they need large processing data volumes.*

Keywords: Distributed system, Big Data, Volunteer computing, e-Science, Cloud computing

Resumo. *O projeto mc^2 (Minha Cloud Científica), no qual se insere a presente proposta, objetiva prover uma plataforma de suporte à execução de aplicações científicas em recursos computacionais de diferentes paradigmas: agregados, redes par-a-par, redes voluntárias e sistemas de larga escala. Neste escopo, este trabalho aborda a inclusão de recursos de computação voluntária e de processamento intensivo de dados na plataforma mc^2, juntamente com a análise de métricas de desempenho para alguns estudos de casos. Nestas análises foram obtidos resultados satisfatórios para os dois módulos do mc^2 citados. Através deles foi possível avaliar a facilidade que os cientistas de diversas áreas podem obter usufruindo da plataforma e o ganho em desempenho quando necessitam de grandes volumes de processamento.*

Palavras-chave: Sistemas distribuídos, Computação intensiva de dados, Computação voluntária, Computação científica, Computação em nuvem

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1. Introdução

As aplicações científicas e seus respectivos fluxos de dados têm se popularizado e crescido na comunidade de cientistas. O avanço da tecnologia também tem proporcionado o crescimento exponencial de um conjunto de dados científicos a cada ano. Este crescimento reflete a grande quantidade de dados e a complexidade dos processos de análise de dados. Classificando melhor as aplicações científicas, elas podem ser divididas em diferentes aspectos: processos científicos complexos, processos de automação de dados derivados; computação de alto desempenho (*HPC - High Performance Computing*) para provisão de vazão e desempenho; e provisão de gerenciamento e consulta [Zhao and Foster 2008].

Atualmente, tem-se produzido dados e informações que já passaram de Exabytes[1] [Hilbert et al. 2010]. A área de tecnologia da informação tem sido motivada por essas demandas de dados para processar e cada vez mais se preocupa em desenvolver arquiteturas e sistemas com o objetivo de resolver estes requisitos. Para o processamento e armazenamento destes dados em sistemas distribuídos foram desenvolvidas diferentes arquiteturas, tais como os agregados (*clusters*), a computação em grades, a computação em nuvem, a computação voluntária e as redes par-a-par.

A plataforma mc^2, um acrônimo para "Minha Cloud Científica", oferece uma abstração de alto nível para fluxos de tarefas de aplicações científicas. Através dos componentes de computação distribuída, o usuário desta plataforma pode usufruir de flexibilidade e o mínimo de dificuldades para a utilização das aplicações científicas implantadas nela.

A necessidade de reduzir o custo de processamento e de disponibilizar uma plataforma de simples acesso a um usuário que não tem conhecimentos avançados em computação foram os principais problemas atacados neste projeto. Pensando nisso, a arquitetura do projeto foi dividida em camadas, de acordo com as funcionalidades e o nível de abstração de cada usuário. A camada de infraestrutura do projeto mc^2 permite que os computadores, utilizados para a provisão de escalabilidade horizontal, possam estar geograficamente distribuídos. Dependendo do módulo utilizado, a escalabilidade é diferente, juntamente com todas as características que o acompanham, como a segurança ou a

[1] Unidade de medida de informação que equivale 1 Exabyte equivale a 1,1529215 x 10^{18} byte.

latência. Os frameworks de computação em grade, computação voluntária e computação intensiva de dados, utilizados na plataforma mc^2, fornecem suporte para implantação das aplicações científicas. A camada de software do projeto mc^2 oferece uma interface Web, na qual o usuário (cientista) não tem mais a necessidade de conhecer as ferramentas de computação distribuída. Ele pode criar uma única tarefa ou um fluxo de tarefas dependentes para seu sistema implantado no projeto mc^2 e assim usufruir da flexibilidade da plataforma de acordo com o tipo de aplicação que ele pretende executar. Entre estas duas camadas existe mais uma que promove a integração e comunicação segura entre elas.

O mc^2 incorpora interfaces que oferecem suporte a aplicações científicas com poder de processamento baseado em clusters, computação em grade, computação voluntária e computação intensiva de dados. Foram escolhidos os temas de computação voluntária e computação intensiva de dados para a elaboração desta dissertação, por haver grande relação entre eles. Os dois frameworks utilizados, o BOINC [Anderson 2004] e o Hadoop [White 2009], são *open-source* e têm grande aceitabilidade na comunidade científica. Mais adiante, na Seção 4.2 serão discutidos outros trabalhos que utilizam os mesmos frameworks utilizados na plataforma mc^2.

1.2. Motivação

O avanço da ciência e o crescimento das complexidades de análises científicas têm motivado pesquisadores e cientistas a trabalharem com sistemas de fluxo de dados para processos coordenados, automatizados, que provêem rastreamento e registro de logs [Zhao and Foster 2008]. Os fluxos de dados científicos têm crescido na computação científica moderna e muitos cientistas e pesquisadores têm conduzido seus trabalhos nestas análises e inovações.

Assim como essa tecnologia vem crescendo, o conjunto de dados para ser processados também cresce exponencialmente a cada ano, tanto em quantidade como em complexidade. Relatos da comunidade científica apontam que os dados analisados de muitas aplicações tendem a crescer cada vez mais. Isso traz a necessidade de sistemas que abordem características de facilidade de expansão ou escalabilidade [Mone 2013].

Portanto, a motivação para esta dissertação é a necessidade de incorporar mais flexibilidade computacional para o processamento de aplicações científicas. A estas flexibilidades incluem-se a coordenação, fluxo de dados, escalabilidade, desempenho, poder computacional e modularização de tarefas ou processos.

1.3. Objetivo

O presente trabalho mostra a pesquisa na área de sistemas distribuídos, abrangendo aspectos de computação voluntária e computação intensiva de dados. O objetivo é especializar estes serviços para serem acoplados na plataforma mc^2, oferecendo suporte a aplicações científicas. Portanto, esta pesquisa se realiza no contexto do projeto mc^2, que pretende oferecer uma plataforma de fácil utilização para usuários sem conhecimento específico na área de computação distribuída. Baseado nesta plataforma, o trabalho desenvolvido inclui a configuração e a contrução de interfaces de comunicação e gerenciamento com os *frameworks* de computação voluntária e computação intensiva de dados, juntamente com um *framework* de computação em grade que é o centralizador e gerenciador da plataforma mc^2.

Pelo fato do objetivo ser disponibilizar diferentes plataformas de computação distribuída no contexto do projeto mc^2, a metodologia de pesquisa e estudo é a comparação destas plataformas. Os aspectos de comparação são o desempenho, a transparência, a eficácia e a usabilidade da plataforma para o usuários final. Estas comparações também têm o objetivo de analisar as aplicações de modo serial e paralelizado, em um mesmo módulo e em diferentes módulos, quando possível. O trabalho foi dividido em duas etapas. A primeira etapa se concentrou no módulo de computação voluntária e a segunda etapa no módulo de computação intensiva de dados. De acordo com os estudos de caso foi possível verificar que os dois módulos de computação voluntária e intensiva de dados apresentaram ganhos significativos no desempenho do processamento dos dados na plataforma mc^2. Foi possível também concuir que o trabalho de engenharia realizado para a integração destes módulos na plataforma mc^2 facilitou o trabalho cotidiano de cientístas que utilizam as aplicações científicas estudadas. Toda a abstração de computação distribuída foi encapsulada nos módulos. Foi possível também verificar a facilidade de acoplamento de mais nós nos módulos, o que aumentou ainda mais o desempenho de execução das tarefas. Relacionado a estes aspectos, a plataforma mc^2 ofereceu a oportunidade de trabalho na integração de diversos *frameworks* de sistemas distribuídos e a análise da melhor coordenação entre eles.

1.4. Estrutura do trabalho

Este trabalho está dividido da seguinte forma. O Capítulo 2 apresenta uma revisão dos paradigmas e ferramentas utilizadas na plataforma mc^2 com os principais conceitos tratados nesta dissertação. No Capítulo 3 é apresentado a plataforma mc^2 com os detalhes da sua arquitetura. No Capítulo 4 são apresentadas as integrações da plataforma mc^2 com uma solução de computação voluntária e uma solução de computação intensiva de dados, juntamente com os trabalhos relacionados à plataforma mc^2. No Capítulo 5 são apresentados estudos de casos com aplicações e-Science que utilizam o modelo de computação voluntária quanto para o modelo de computação intensivo de dados. No Capítulo 6 são relatados as principais contribuições, limitações e ideias para trabalhos futuros relacionado a esta dissertação. Nos apêndices A e B são explicados os detalhes da integração de computação voluntária e intensiva de dados com o plataforma mc^2.

Capítulo 2

Paradigmas e ferramentas de computação distribuída

Neste capítulo será apresentada a evolução dos sistemas distribuídos na área de ciência da computação. Os paradigmas e modelos de computação distribuída serão divididos em Seções para que fique clara a balisa das diferentes características destes sistemas. Para que se inicie o relacionamento dos paradigmas de computação distribuída com a plataforma mc^2 também serão apresentadas as ferramentas utilizadas nesta plataforma.

2.1. Evolução dos paradigmas de sistemas distribuídos

A era de ouro dos supercomputadores foi a década de 1980 [Ceruzzi 1998], quando não existia outra chance para lidar com processamento intensivo de tarefas. O crescimento dos computadores em rede e a queda dos mainframes foram as mais dramáticas mudanças na década de 1980 para a área de tecnologia da informação. Esta mudança colocou mais poder computacional nas mãos do usuário final e distribuiu os recursos de *hardware* através do sistema. O *hardware* foi o primeiro recurso afetado, depois o novo desafio foi desenvolver *softwares* que atendessem o modelo de infraestrutura dos recursos distribuídos. Vários modelos habilitaram esta distribuição, desde o simples compartilhamento de dados até os avançados sistemas de suporte a múltiplos serviços. Na década de 1990 foi observado o aumento da velocidade dos processadores, da capacidade de armazenamento e do aumento da largura de banda de rede. Como consequência, a computação em agregados (*cluster*) e ambientes de computação em grade influenciaram o barateamento do processamento e do armazenamento no hardware disposto para computação científica. A computação em agregados oferece máquinas homogêneas interconectadas por alta velocidade de rede com armazenamento de acesso local e um domínio administrativo. A computação em grade foca na distribuição de recursos compartilhados e coordenação através de múltiplas organizações virtuais espalhadas geograficamente. Com a introdução das arquiteturas de vários processadores, a separação entre computação em grade e supercomputação se tornou menos clara ainda [Zhao and Foster 2008].

As redes par-a-par (*peer-to-peer*) são sistemas distribuídos cooperativos nos quais seus participantes são vistos como entidades anônimas e conseguem compartilhar seus recursos em benefício próprio ou alheio [Anagnostakis et al. 2006]. Suas principais características são o provimento de serviços escaláveis, estáveis, confiáveis e a possibilidade

de prover em larga escala recursos compartilhados entre os participantes. Os termos cliente e servidor têm sido associados a atributos específicos deste modelo de computação distribuída. Logo depois, o termo *peer* veio para classificar os computadores que abrangem estas duas características anteriores. Eles podem ser usados para representar papéis de participantes num protocolo de comunicação, sendo que estes podem mudar constantemente, dependendo da execução atual do computador. Em aplicações ou ambientes distribuídos, o termo servidor é usado para o módulo de *software* ou componente que é responsável por prover um serviço. Além disso, o termo cliente é usado para o módulo de *software* ou componente que atua como consumidor de um determinado serviço. Estes pápeis são idependentes do protocolo de comunicação utilizado no nível de rede. Quando um computador pode oferecer o papel de cliente e servidor ao mesmo tempo, ele é chamado de *peer*.

Diferentemente da arquitetura cliente/servidor, o conceito de *middleware* é entendido como uma classe de tecnologias de *software* projetado para ajudar a gerenciar a complexidade e heterogeneidade inerente aos sistemas distribuídos. Define-se como uma camada de *software* acima do sistema operativo, mas abaixo do programa de aplicação, que proporciona uma captação de programação comum através de um sistema distribuido [Bernstein 1996]. Esta camada deve prover serviços de comunicação como localização, links de resolução, autenticação, transações semânticas e tempo de sincronização entre formato de dados. Esta camada adicional permite que os clientes interajam com uma abstração genérica de um servidor em vez de servidores ou processos específicos.

Na metade da década de 1990, o termo grade foi concebido para descrever tecnologias que iriam permitir consumidores obterem poder computacional por demanda. Ian Foster e outros pesquisadores se dedicaram à padronização de protocolos usados para requisição de poder computacional, que foi o início da computação em grade, uma forma análoga da utilidade de grades de potência elétrica [Foster and Kesselman 1999]. Pesquisadores subsequentes desenvolveram ideias em outros caminhos, produzindo sistemas federados de larga escala (TeraGrid, Open Science Grid, caBIG, EGEE, Earth System Grid), que fornecem não só poder computacional, mas também dados e software por demanda [Ian Foster 2008].

A computação em nuvem foi o próximo desafio da comunidade de sistemas distribuídos. Sua proposta tem como principal característica a elasticidade de recursos computacionais que é a utilidade computacional com potencial de transformar grande parte da indústria de tecnologia. Os novos serviços de Internet não requerem mais grande quantidade de capital gasto com hardware para implantar seus serviços ou gasto humano para operá-los. Um grande processo orientado a tarefas pode ter resultados tão rápidos quanto programas que podem ser escaláveis [Michael Armbrust 2009]. As visões da computação em grade e da computação em nuvem são as mesmas. Estes paradigmas prometem reduzir o custo de computação, aumentar a confiança e a flexibilidade partindo de um modelo em que os computadores são operados por uma terceira parte de usuários e que os usuários não tem contato direto com o hardware [Ian Foster 2008]. Porém, a computação em nuvem oferece um tipo de serviço que atende ao usuário de acordo com sua demanda, muitas vezes chamado de *pay-as-you-go*. Sendo assim, o usuário projetista pode iniciar seu ambiente com o mínimo de recurso financeiro e uma quantidade de tecnologias necessárias para o projeto.

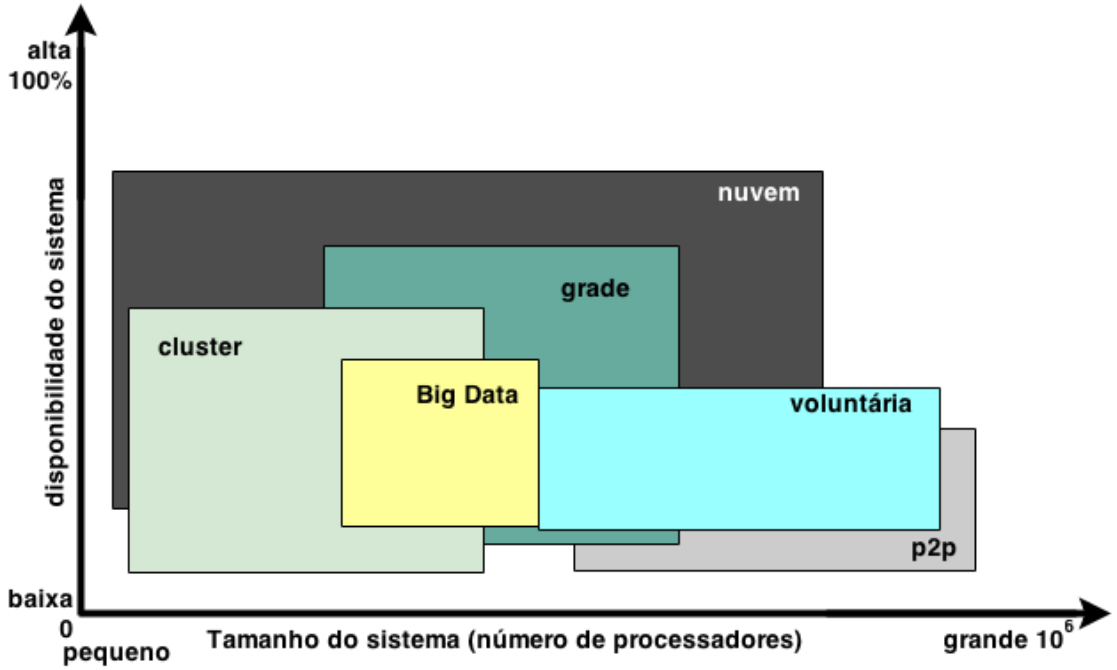

Figura 2.1. Estimativa de disponibilidade pelo tamanho do sistema em configurações comuns - adaptada de [Hwang et al. 2012]

A Figura 2.1 mostra o relacionamento estimado em configurações normais das arquiteturas de sistema distribuído a serem discutidas num gráfico de disponibilidade versus tamanho do sistema [Hwang et al. 2012]. Nela é possível verificar que os sistemas em agregados (*cluster*) e de grade estão mais disponíveis que os outros. Isto ocorre porque eles têm maior confiabilidade de acesso a qualquer momento, por estarem conectados numa rede gerenciada. Porém, os sistemas de par-a-par e de computação voluntária têm maior capacidade de processamento para serem usados. Isto ocorre porque eles são sistemas de baixo acoplamento e, consequentemente, usuários com características diversas facilmente se conectam nele. Em compensação, os sistemas de computação intensiva de dados (*Big Data*) possuem menor tamanho, ou número de processadores, que a maioria dos outros sistemas, devido ao fato de estarem sujeitos a uma rede privada e terem uma característica oposta ao *embarassing parallel*[1]. A sua disponibilidade não chega como a dos sistemas em *cluster*, grade ou nuvem, que podem ter os processadores disponíveis por mais tempo. Porém ela ultrapassa a dos sistemas par-a-par e voluntários, pois está sujeita a um controle maior do administrador de sistemas distribuídos.

2.2. Principais métricas

Em sistemas distribuídos, existem várias métricas que são usadas para qualificar um determinado sistema, dentre as quais podem ser destacadas a escalabilidade, a resiliência, o desempenho, a latência, a confiabilidade. Quando se está apresentando a definição de paradigmas de sistemas distribuídos é comum citar o termo escalabilidade. Escalabilidade, em sistemas distribuídos, indica a habilidade de manipular uma porção crescente de

[1] *Embarassing parallel* é uma característica de sistemas paralelos onde as tarefas não têm relação umas com as outras, ou seja, um paralelismo tímido.

trabalho ou que está preparado para crescer de forma uniforme [Bondi 2000]. Resiliência é a capacidade de um software ou de um sistema em recuperar sua forma ou posição original de acordo com o número e ao grau de severidade das falhas. O desempenho em sistemas distribuídos pode ser avaliado em vários aspectos. Uma métrica relevante é a medição da velocidade que uma mensagem pode ser transmitida entre dois computadores interligados. Esta métrica é dividida em latência e taxa de transferência de dados. A latência é o tempo decorrido entre a solicitação de serviço e entrega do resultado, ou seja, o tempo de envio que um pacote demorou para que o destinatário comece a recebê-lo. A taxa de transferência de dados é a velocidade com que os dados podem ser transferidos entre dois computadores em uma rede, medida em bits por segundo. Portanto, o tempo de transmissão de uma mensagem é igual sua latência somada com a largura da banda dividida pela taxa de transferência de dados, como é apresentada a Equação 2.1. A largura da banda é determinada pela tecnologia de rede empregada, sendo que mensagens maiores precisam ser fragmentadas. A confiabilidade (do inglês, *reliability*) define a capacidade do sistema em prover um serviço operacionalmente correto, mesmo na presença de falhas [Avizienis et al. 2004]. A confiabilidade está ligada ao impacto dos erros de comunicação. Existem aplicativos que são capazes de se recuperarem de falhas de comunicação, e assim, não exigem a garantia da comunicação isenta de erros. Já a segurança está associada a requisitos e técnicas para proteger os sistemas distribuídos.

$$Tempo_{trans} = \frac{latencia + largura\ da\ banda}{Taxa_{transfer}} \tag{2.1}$$

Combinando estas características citadas com a Figura 2.1 é possível perceber que o sistema de computação em agregados fornece menor latência, pois os computadores estão numa mesma rede privada. A resiliência dos sistemas distribuídos está sendo mais desenvolvida nos aspectos de computação em nuvem e computação intensiva de dados. A confiabilidade, assim como todas as outras características, é abrangida por todos os outros tipos de sistemas distribídos. Nos tipos de sistemas de computação voluntária e computação par-a-par esta característica é garantida pela redundância do processamento dos dados. Nos outros tipos de sistemas distribuídos não é necessário o mesmo grau de redundância, pois os computadores estão numa mesma rede privada que acredita-se ser confiável.

2.3. Computação em agregados

O paradigma de computação em agregados (*cluster computing*) foi desenvolvido no início de 1994, baseado em um esboço de sistema de processamento distribuído. Ele é composto de dois ou mais computadores independentes, interligados e que trabalham em conjunto, trocando informações sobre uma mesma tarefa [Tannenbaum et al. 2001]. Os clusters têm como principal características o domínio centralizado, por isso é desnecessário a segurança de seus processos e compartilhamento de recursos. Eles também possuem alta escalabilidade. Isto promove a inclusão e exclusão de nós escravos sem que sejam exigidas modificações no ambiente, ou seja, de forma isolada partindo do nó mestre.

Os *clusters* são classificados geralmente em três tipos, de alto desempenho, de alta disponibilidade e de balanceamento de carga. Os clusters de alto desempenho (*High Performance Computing Cluster*) são direcionados ao processamento de uma quantidade

massiva de dados, na casa de giga FLOPS[2], em computadores comuns e com sistemas operacionais gratuitos. Os *clusters* de alta disponibilidade (*High Availability Computing Cluster*) são direcionados para manter o funcionamento dos seus serviços durante o máximo de tempo possível, através de mecanismos de detecção, redundância, recuperação e mascaramento de falhas. Os *clusters* de balanceamento de carga (*Load Balance Cluster*) têm a função de controlar a distribuição equilibrada do processamento. Eles requerem um monitoramento constante da comunicação e nos mecanismos de redundância, pois se ocorrer alguma falha é necessário que não se interrompa o seu funcionamento. Estes tipos de características dos *clusters* podem ser combinadas entre si, provendo mais funcionalidades de acordo com os requisitos esperados.

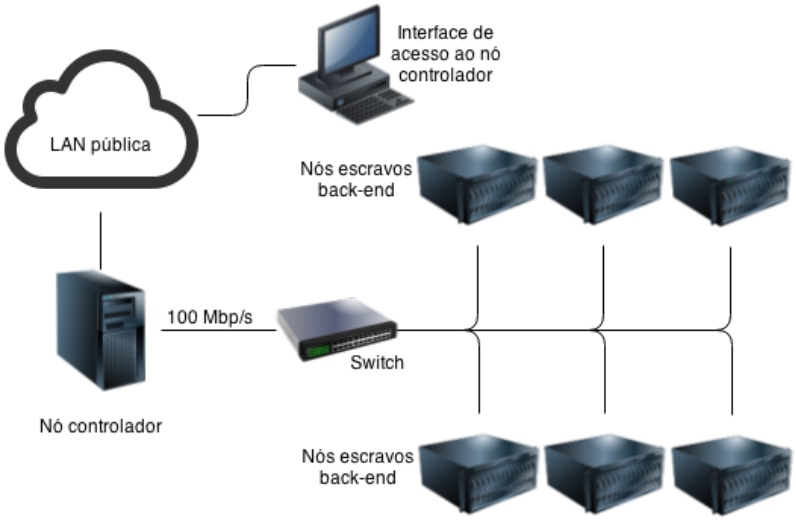

Figura 2.2. Sistema de computação em agregados

A Figura 2.2 representa a arquitetura típica de um sistema em agregados. O nó controlador, também chamado de nó mestre, atua como servidor, monitoranto e distribuindo as tarefas para os outros nós. Os outros computadores são os nós trabalhadores, também chamados de escravos, que executam as tarefas requisitadas pelo nó controlador. Eles se comunicam através de uma rede dedicada com um switch. Por isso, os nós escravos não precisam de periféricos de acesso, tais como monitor, teclado ou mouse. O acesso ao sistema de agregados é pelo nó controlodar através de alguma interface.

O *Cluster Beowulf* foi criado pelos pesquisadores Thomas Sterling e Donald J. Becker da NASA (*National Aeronautics and Space*) em 1994. O objetivo destes pesquisadores foi desenvolver um sistema de computação paralela para o processamento de informações espaciais na ordem de giga FLOPS. Na arquitetura deste sistema é especificado que pelo menos um dos computadores do cluster deve exercer o papel de controlador dos demais. Esses tipos de computadores são chamados de *front-end* e *back-end* respectivamente. A comunicação entre os computadores pode ser feita através de redes

[2]FLOPS é um acrônimo na computação que significa *FLoating-point Operations Per Second*, que em português quer dizer operações de ponto flutuante por segundo. Os FLOPS são calculados pela multiplicação da quantidade de cores do processador, pela frequencia do processador e pelos FLOPs/ciclo (Flops = cores * clock * FLOPs/cycle). Gigaflop é o seu múltiplo que equivale a 10^9.

do tipo *Ethernet* [Tannenbaum et al. 2001]. Esta arquitetura não exige *hardwares* homogêneos, porém isto pode ser relevante para diminuir a complexidade de configuração e manutenção do sistema, e garantir que os procedimentos rotineiros ao *cluster*, como monitorização, distribuição de tarefas e controle de recursos, sejam executados de maneira uniforme. Os nós devem ter dedicação exclusiva ao *cluster*. Deve-se utilizar uma biblioteca de comunicação apropriada para a troca de mensagens entre os nós, como a PVM (*Parallel Virtual Machine*) [Beguelin et al. 1991] e a MPI (*Message Passing Interface*) [Dongarra et al. 1993]. A biblioteca *MPI* é considerada mais avançada que a *PVM*, pois consegue trabalhar com comunicação para todos os computadores ou para apenas um determinado grupo.

2.4. Computação em grade

O paradigma de computação em grade vem sendo estudado desde a segunda metade do século XX [Foster 2002]. Ele surgiu como uma alternativa para a execução de aplicações com necessidade de grande poder computacional, integrando recursos possivelmente heterogêneos e geograficamente distribuídos.

Em 1969, Len Kleinrock sugeriu a possibilidade de existir computação em grade [Kleinrock 2014]. ”Nós iremos provavelmente ver um espalhamento de utilidades computacionais, que, como as utilidades presentes de eletricidade e telefonia, irão servir casas e escritórios individuais pelo país”. Em 1998, Carl Kesselman e Ian Foster [Foster and Kesselman 1999] definiram computação em grade como ”uma infraestrutura de *hardware* e *software* que provê confiabilidade, consistência e acesso de baixo custo para altas capacidades computacionais”. A essência das descrições acima pode ser capturada nas frases a seguir. Um sistema de computação em grade deve ter recursos coordenados que não estão sujeitos a um controle centralizado. Ele se utiliza de padrões, protocolos de propostas gerais e interfaces que são endereçadas para contratos de autorização, autenticação, descoberta de recursos e acesso a recursos, provendo qualidade de serviço como tempo de resposta, disponibilidade, segurança, rendimento e alocação de múltiplos recursos. De acordo com o Centro de Informação sobre computação em grade (GRID Infoware) [Computing 2014], a definição deste paradigma é: “Grade é um tipo de sistema distribuído e paralelo que habilita o compartilhamento, seleção e agregação de recursos dinâmicos, autônomos, geograficamente distribuídos, com tempo de execução dependendo da sua disponibilidade, capacidade, desempenho, custo e requisitos de *QoS* (*Quality of Service*) de usuários.”

A tecnologia de computação em grade [Nadiminti and Buyya 2005] provê soluções efetivas para a necessidade de agregação de recursos distribuídos e priorização de alocação de recursos para diferentes usuários, projetos e aplicações baseadas nos requisitos de *QoS* (*Quality of Service*). Estes benefícios geralmente resultam em larga redução de custos para os negócios. O real e específico problema que envolve a computação em grade é a coordenação de recursos compartilhados e a solução de problemas dinâmicos de organizações virtuais [Foster et al. 2001]. Este compartilhamento envolve computadores, softwares, dados e outros recursos que compõem o universo de estratégias para a solução de problemas na indústria, ciência e engenharia. O compartilhamento é altamente controlado com os provedores e consumidores de recursos, definindo claramente e especificamente o que é compartilhado, quem tem permissão para compartilhar e as políticas em que os compartilhamentos ocorrem. O conjunto de indivíduos ou instituições definidas

para as políticas de compartilhamento são chamadas de *virtual organizations* (*VO*).

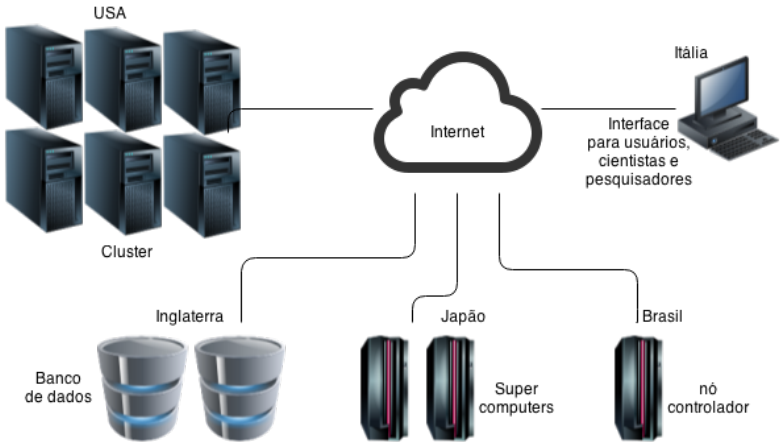

Figura 2.3. Sistema de computação em grade

Na definição de uma arquitetura em grade, o ponto crucial é o estabelecimento de compartilhamento entre qualquer potencial participante na *VO*. Isto é definido como a interoperabilidade da arquitetura em grade. No ambiente de rede a interoperabilidade significa protocolos comuns. Os protocolos definem os mecanismos básicos que os participantes das *VOs* e os recursos negociados irão estabelecer, gerenciar e explorar os relacionamentos compartilhados.

A Figura 2.3 representa a arquitetura típica de um sistema em grade. Da mesma forma que a arquitetura do sistema em agregados, este tipo de sistema distribuído tem um nó controlador. Porém, sua capacidade de processamento é muito maior, devido a associação de federação de *clusters*. Além disso, esse tipo de sistema pode abranger diferentes tipos de computadores e sistemas operacionais, representados pela figura como super computadores e banco de dados. Essas características ocorrem devido ao sistema não se limitar a uma rede privada, como é o caso do sistema em agregados. O sistema em grade, apesar de ter um nó de controle dedicado, os nós escravos podem estar distribuídos geograficamente pela Internet.

2.5. Computação par-a-par

As redes par-a-par (P2P - *peer-to-peer*) têm a característica de serem redes sobrepostas e auto gerenciáveis de forma descentralizada [Karl Aberer 2005] ou híbridas. São características de um sistema par-a-par a eficiência, escalabilidade, auto organização, tolerância à falhas e cooperação. Elas são consideradas redes *overlay* porque usam uma topologia virtual diferente daquela definida pela rede comum. Esta topologia virtual (da rede *overlay*) é usada para viabilizar alguma propriedade da rede par-a-par. Por exemplo, descoberta de nós, localização e distribuição dos dados (ou arquivos). Os nós nestes sistemas compõem uma rede sobreposta. Cada nó mantém apenas informações sobre um pequeno número de outros nós do sistema. Isto limita a quantidade de estado que deve ser mantida, e, consequentemente, aumenta a escalabilidade [Milojicic et al. 2003]. Seguindo os conceitos de redes *overlay*, onde uma topologia lógica pode ser construída sobre uma topologia física. Os nós participantes desse tipo de arquitetura podem ter suas

conexões representadas em uma forma virtual, que corresponde a um caminho ou rota física na rede subjacente.

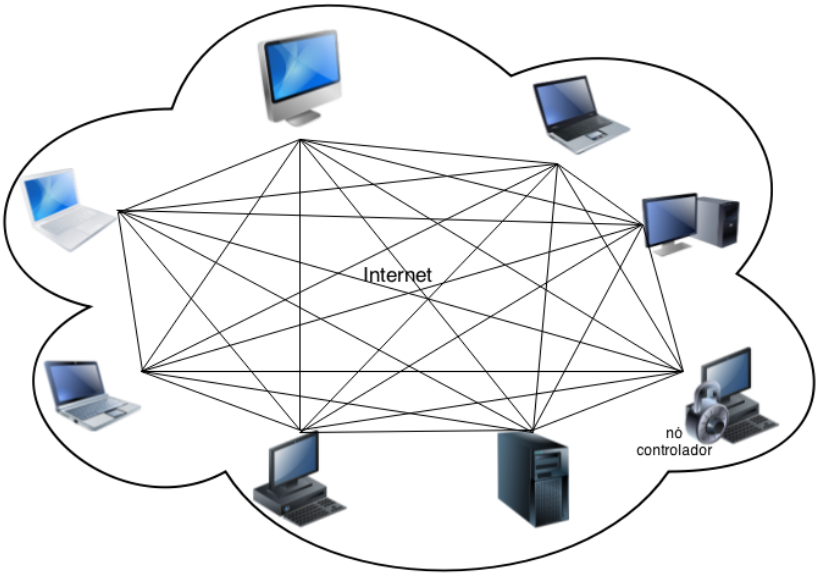

Figura 2.4. Sistema de computação par-a-par

Em termos arquiteturais, as redes par-a-par podem ser descentralizadas (auto-gerenciáveis) ou híbridas (com alguns componentes centralizados). A principal diferença entre os maiores exemplos de cada um desses tipos é a busca pelos recursos e manutenção da comunicação entre os pares. Num sistema auto-gerenciável não há um servidor mestre no sistema que gerencia a comunicação entre os pares e a busca é feita através de técnicas de *flooding*[3] com *TTL*[4] (como em alguns algorítmos de roteamento) para receber respostas com localizações e estados de outros *peers*. Já em arquiteturas híbridas, serviços como busca de arquivo ou regras de negócios são regidas por um nó mestre que utiliza metadados para gerenciar alguns aspectos da rede.

A eficiência nas redes par-a-par é atingida com o mínimo de saltos para o fluxo na rede. A escalabilidade é quando um alto número de participantes se associam ao sistema sem que haja significativa degradação do desempenho. A auto organização é a capacidade do sistema não ter um controle centralizado e conseguir lidar com a frequente mudança na configuração dos seus participantes. A tolerância à falhas é a capacidade do sistema de ter seus recursos acessíveis mesmo com falhas no sistema. Ela é geralmente associada a característica de redundância de recursos. A cooperação é a característica de que os participantes interagem entre si num ambiente confiável e respeitem certas rotinas, resultando em uma qualidade de serviço desejada.

A Figura 2.4 representa o sistema par-a-par. Como é possível notar, esse tipo de sistema está no contexto da Internet. As conexões entre os nós são ambivalentes, e em

[3]Técnica responsável por manter as bases de dados sincronizadas.

[4]*Time-To-Live* - É o valor em um protocolo IP que informa ao roteador o tempo de envio de um pacote e se ele deve ser descartado.

alguns sistemas pode existir um nó controlodos, porém, em outros, alguns nós podem ser escolhidos aleatoriamente como um nó controlador.

2.6. Computação voluntária

Computação voluntária é também conhecida como computação global. Este tipo de paradigma de computação utiliza computadores voluntários de um público geral para realizar processamento distribuído [Anderson and Fedak 2006]. O paradigma de computação voluntária está sendo usado em experimentos físicos de alta energia, biologia molecular, medicina, astrofísica, estudo de climas e outras áreas.

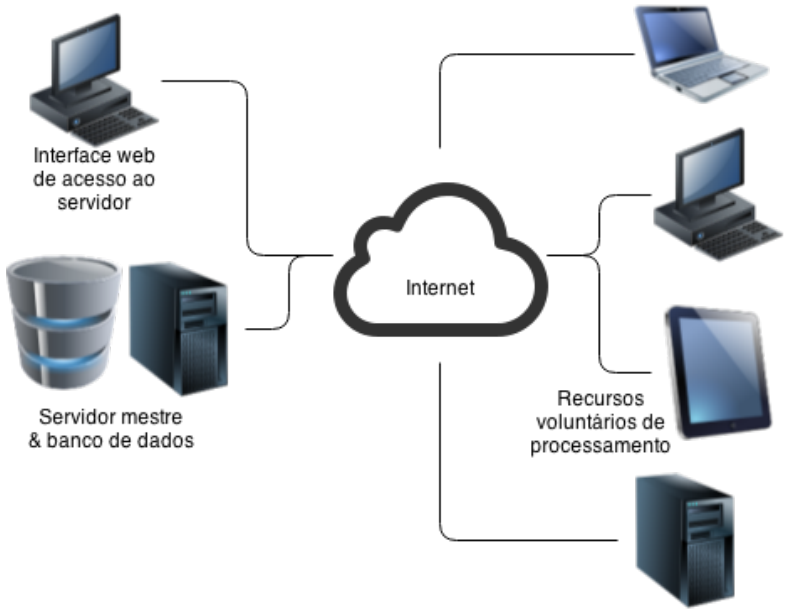

Figura 2.5. Sistema de computação voluntária

O projeto *SETI@home* tem sustentado uma taxa de processamento de 60 Tera FLOPS por ano. Ele utiliza o BOINC (*Berkeley Open Infrastructure for Network Computing*) [Anderson 2004], um *middleware* para computação voluntária. Este projeto utiliza o poder de processamento de computadores ociosos na Internet. Porém, o potencial da computação voluntária se estende além disso. Ela abrange aplicações que requerem significativa memória, espaço de disco e vazão de rede. Os participantes voluntários podem se anexar a um conjunto de projetos, controlar seus recursos compartilhados e também limitar quando e como o BOINC utiliza seus recursos computacionais. A arquitetura do BOINC é dividida em um servidor e vários clientes, chamados respectivamente de *Boinc server* e *Boinc client*. O *Boinc client* contacta periodicamente o *Boinc server* de cada projeto anexado para reportar suas configurações de hardware e a disponibilidade de dados. Também existem outros *middlewares* para computação voluntária disponíveis, tais como o SLINC [Baldassari et al. 2006].

A Figura 2.5 representa o sistema de computação voluntária. Como é possível notar, este tipo de sistema funciona no contexto da Internet. O nó controlador, também chamado de servidor mestre, contém um banco de dados para armazenar as informações

processadas. Ele disponibiliza um *web-service* para facilitar o acesso ao usuário administrador da aplicação. A aplicação implantada no servidor requisita recursos de processamento de diversos tipos de usuários na Internet, representados na figura através dos diversos tipos de computadores.

2.7. Computação em nuvem

Computação em nuvem se refere a entrega de serviços através da Internet. Os *hardwares* e sistemas de *softwares* nos *data centers*[5] proveem estes serviços. Uma nuvem disponível de uma maneira chamada *pay-as-you-go* para um público geral é chamada de nuvem pública e os serviços vendidos são chamados de utilidades computacionais. O termo nuvem privada é usado para *data centers* internos em organizações. Elas oferecem maior segurança e rapidez de processamento. Nos dois tipos de modelos de nuvem os usuários finais podem acessar os serviços a qualquer momento e de qualquer lugar, compartilhar os dados e colaborar entre si com grande facilidade [Armbrust et al. 2010]. Do ponto de vista de *hardware*, três aspectos são novos em computação em nuvem. Estes três fatores cobrem a economia do custo de energia, largura de banda, operações, software e disponibilidade de *hardware*.

1. A ilusão de recursos computacionais infinitos disponíveis por demanda, o que elimina a necessidade de usuários de computação em nuvem terem um planejamento extenso do seu projeto.
2. A eliminação de alto investimento no começo de projetos.
3. A possibilidade de pagar pelo uso de recursos computacionais baseados em um termo curto de responsabilidade e liberação dele como necessidade.

O *NIST* [NIST 2014] (*National Institute of Standards and Technology*) pretende atender as necessidades de organizações comerciais. A definição do NIST sobre computação em nuvem abrange três modelos, o *SaaS* (*Software as a Service*), o *PaaS* (*Plataform as a Service*) e o *IaaS* (*Infrastructure as a Service*). No *SaaS* o cliente usa uma aplicação, mas não controla o sistema operacional, o hardware ou a infraestrutura da rede onde a aplicação está executando. No *PaaS* o cliente usa um ambiente de hospedagem para a sua aplicação, mas também não controla o sistema operacional, o hardware ou a infraestrutura da rede onde a aplicação está executando. No *IaaS* o cliente usa os recursos fundamentais de computação, tais como potência de processamento, armazenamento, componentes de rede ou middleware. Existem também a definição do *EaaS* (*Everything as a Service*) ou *XaaS* (*X as a Service*). Nestes paradigmas são incluídos todos os tipos de recursos disponíveis e possíveis como serviço.

As definições do *NIST* para os modelos de implantação de computacão em nuvem são quatro: Nuvem Pública, Nuvem Privada, Nuvem Comunitária e Nuvem Híbrida. Uma nuvem pública contém serviços disponíveis para clientes de serviços provedores de terceiros pela Internet. Uma nuvem privada contém todos os benefícios do ambiente de computação de uma nuvem pública, porém os serviços, dados e processamentos são gerenciados numa organização sem restrições de banda de rede, segurança e requisitos

[5]Um Data Center é um recurso usado para sistemas de computadores e componentes associados, tais como sistemas de armazenamento e telecomunicações. Geralmente inclui suprimentos redundantes ou de backup de energia, conexões de comunicação de dados redundantes, controles ambientais e dispositivos de segurança.

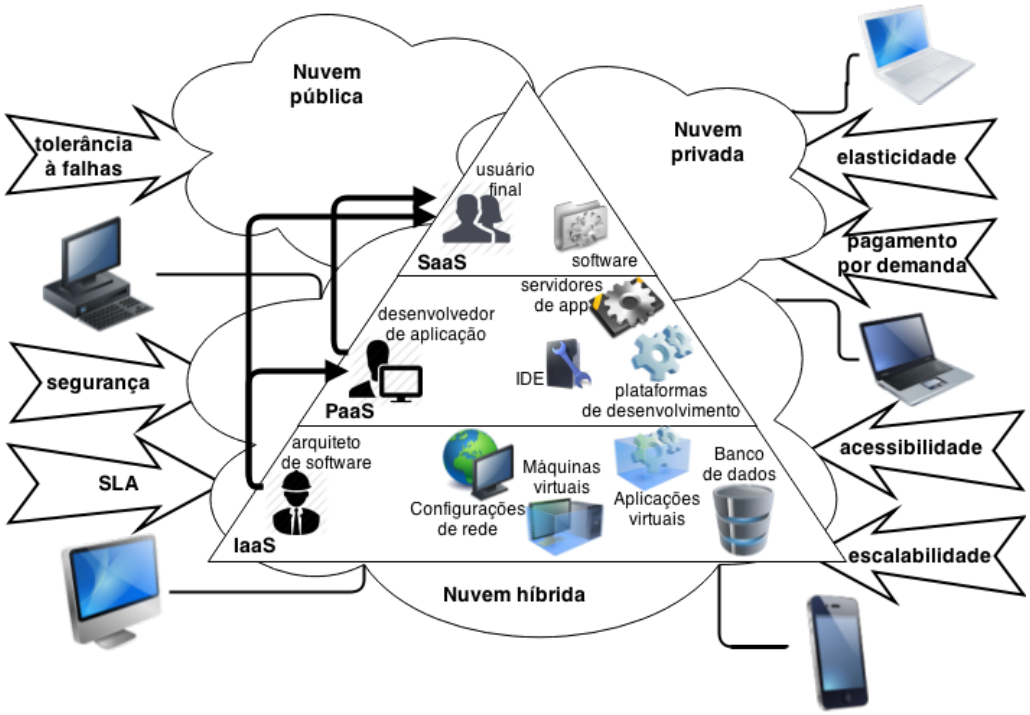

Figura 2.6. Sistema de computação em nuvem

legais. Ela também oferece ao provedor e ao usuário melhor controle da infraestrutura da nuvem, melhorando a segurança e a resiliência. Na nuvem comunitária um grupo de organizações que têm interesses em comum, tais como segurança específicas ou objetivos em comum, usam e controlam a nuvem. A nuvem híbrida é uma combinação da pública e privada. As informações críticas e relevantes são processadas na nuvem privada e as informações que não exigem segurança e sigilo são processadas na nuvem pública.

Vários são os serviços de computação em nuvem disponíveis na Internet que exemplificam estes tipos citados pelo *NIST*. Entre eles podemos citar a Amazon AWS [Amazon 2010], o Eucalyptus [Nurmi et al. 2009], o OpenStack [Sefraoui et al. 2012], o OpenNebula [Milojičić et al. 2011] que fornecem serviços de *IaaS*. O Google AppEngine [Sanderson 2009] e Microsoft Azure [Calder et al. 2011] fornecem serviços de *PaaS*. O Dropbox [Drago et al. 2012] e Google Drive [URL 2014j] fornecem serviços de *SaaS*. Por causa dessa variedade, que mescla serviços pagos e gratuítos, a busca por uma padronização de conceitos se torna cada vez mais necessária. A Figura 2.6 demonstra a relação entre os níveis de plataformas de computação em nuvem, os tipos de usuários, os tipos de acesso, as principais características oferecidas e os tipos de nuvens existentes.

Ao contrário do *NIST*, as definições do grupo *Open Cloud Manifesto* [URL 2014u] de computação em nuvem são baseadas em teorias para atender a comunidade de código aberto e tem o objetivo de apresentar cenários de nuvem que demonstram o desempenho e benefícios econômicos baseados no mais amplo conjunto de costume dos consumidores. O objetivo também abrange evidenciar as capacidades e os requisitos que precisam ser padronizados no ambiente de computação em nuvem para certificar a interoperabilidade, fácil integração e portabilidade. O *Open Cloud Manifesto* propõe alguns princípios para

que a tecnologia de computação em nuvem seja aberta e padronizada.

1. Os provedores de nuvem devem trabalhar juntos para se certificarem que os desafios da adoção de computação em nuvem estão sendo endereçados através de uma colaboração aberta e do uso apropriado de padrões.
2. Os provedores de computação em nuvem devem usar e adotar os padrões já existentes.
3. Deve-se certificar que os padrões promovam inovação e não inibição de novos padrões.
4. Qualquer comunidade ao redor da nuvem aberta deve ser orientada pelas necessidades dos clientes.
5. Organizações de padrões de computação em nuvem, grupos de advogados e comunidades devem trabalhar juntos e coordenados para garantir que não haja esforços conflitantes ou sobrepostos.
6. Provedores de computação em nuvem não devem usar suas posições de marketing para travar os clientes nas suas platafromas particulares e limitar suas escolhas de provedores. Este aspecto é conhecido como *Data Lock-in*.

2.8. Computação intensiva de dados

O crescente volume de dados motivou a criação de um novo paradigma da computação científica. Estes dados chegam na casa de exabytes de informações e os sistemas e protocolos atuais não têm capacidade para o tratamento destes dados. Jim Gray chamou este novo paradigma de Quarto Paradigma da ciência que envolve as áreas de captura, curadoria e análise [Hey et al. 2009]. Na área de genética, o tratamento científico de dados representa um enorme desafio, visto que atualmente o genoma humano foi completamente sequencializado e é necessário analisar estes dados.

A proliferação de serviços do modelo de Computação em Nuvem tem uma forte relação com o modelo de computação intensiva de dados, que utiliza o paradigma *MapReduce* [Jeffrey Dean 2008]. Não há dúvida de que "*everything as a service*"é impulsionado pelo desejo de uma maior eficiência do negócio, mais escalabilidade e elasticidade desempenham papéis importantes também. A nuvem permite a expansão contínua de operações sem a necessidade de um planejamento cuidadoso e suporta escalas que podem ser difíceis ou de custo proibitivo para uma organização alcançar. Serviços em nuvem, como *MapReduce*, representam a busca de um nível adequado de abstração e divisões benéficas de trabalho. *IaaS* é uma abstração sobre a matéria física de hardware de uma organização que pode não ter capital, experiência ou interesse na execução de *data centers* e, portanto, paga um provedor de nuvem para fazê-lo em seu nome. O argumento se aplica igualmente a *PaaS* e *SaaS*. Na mesma linha, o modelo de programação *MapReduce* é uma abstração poderosa que separa o que é oferecido como processamento de dados intensivos.

O *MapReduce* é um modelo de programação e uma implementação associada ao processamento e geração de grande conjunto de dados. Os usuários especificam a computação em funções de mapeamento *Map* e redução *Reduce* através de várias máquinas em *clusters*, com uma comunicação eficiente para fazer o uso de rede e disco. O modelo tem o objetivo de resolver o problema de computação paralela, dados distribuídos e tolerância a falhas. A abstração de *MapReduce* foi inspirada em linguagens funcionais como Lisp. O *MapReduce* é um paradigma de programação, diferente de um *cluster* que

é um modelo de organização de computadores. Portanto, um programa que segue o paradigma de programação *MapReduce* precisa de um *framework* que esteja funcionando sobre um *cluster*.

Existem vários motivos para se usar *MapReduce* como modelo de programação [Lin and Dyer 2010]. Ao escrever um aplicativo, se executamos várias ocorrências ao mesmo tempo e o aplicativo acessa alguns dados comuns, haverá algum conflito. Casos de concorrência, em que *threads* estejam alterando dados enquanto outras *threads* estarão lendo. Mas se a ocorrência está trabalhando em alguns dados que nenhuma outra ocorrência precisará, então estará ocorrendo paralelismo. Obviamente, com paralelismo será possível escalar ainda mais, já que não existem problemas de concorrência.

Com a modelagem de programação *MapReduce* para sistemas distribuídos é possível resolver vários problemas de computação utilizando diversas máquinas ligadas em rede e formando um *cluster*. Em termos gerais é possível diminuir o tempo na obtenção de uma resposta computacional sem precisar aumentar a velocidade de um processador individualmente. O modelo *MapReduce* é uma abstração que cria uma interface entre um programador e um sistema distribuído. Dessa forma, ele faz com que o programador consiga implementar separadamente a solução para seu problema específico e uma solução geral para seu sistema distribuído, a implementação do *MapReduce*. Neste modelo o usuário escreve duas funções: uma *Map(key, value)* e uma *Reduce(key, values)*. Depois, o usuário chama a implementação indicando um conjunto fragmentável ou fragmentado de dados de entrada onde cada fragmento tem o formato (chave, valor). A implementação *MapReduce* é a forma como o sistema distribuído em questão vai executar as funções *Map* e *Reduce*. Sistemas diferentes podem demandar implementações diferentes para um bom funcionamento do *MapReduce*.

De acordo com a Figura 2.7, os dados no modelo MapReduce seguem o seguinte fluxo:

1. A biblioteca de *MapReduce* primeiramente divide os arquivos de entrada em M pedaços de normalmente 16MB até 64MB, dependendo da configuração. Então inicia muitas cópias do programa em um *cluster* de máquinas.
2. Uma das cópias do programa é especial e é enviada para o *master*. O resto dos *workers* são designados ao trabalho pelo *master*. *Workers* e *master* neste conceito de *MapReduce* são os programas nos computadores do *cluster*. Existem *M* tarefas *Map* e *R* tarefas *Reduce* para serem designadas. O *master* requisita os trabalhadores ociosos e designa para cada um uma tarefa *Map* ou *Reduce*.
3. Um *worker* que é designado para tarefas *Map* lê o conteúdo correspondente a entrada dividida. Ele transforma a entrada de *key/value* da entrada para uma saída e cada par *key/value* para a função *Map* definida pelo usuário. Então o par intermediário *key/value* produzido pela função *Map* é alocado na memória.
4. Periodicamente, o *buffer* da memória é gravado em disco local e particionado em regiões *R* pela função de particionamento. As localidades destes *buffers* de pares gravados no disco local são passados para o *master* que é responsável por reencaminhar estas localidades para as funções *Reduce*.
5. Quando um *worker* de *Reduce* é notificado pelo *master* sobre as localizações, ele usa uma chamada remota de processo (*Remote Process Call - RPC*) para ler o dado do *buffer* dos discos locais dos *workers Map*. Quando o *worker Reduce* leu todos

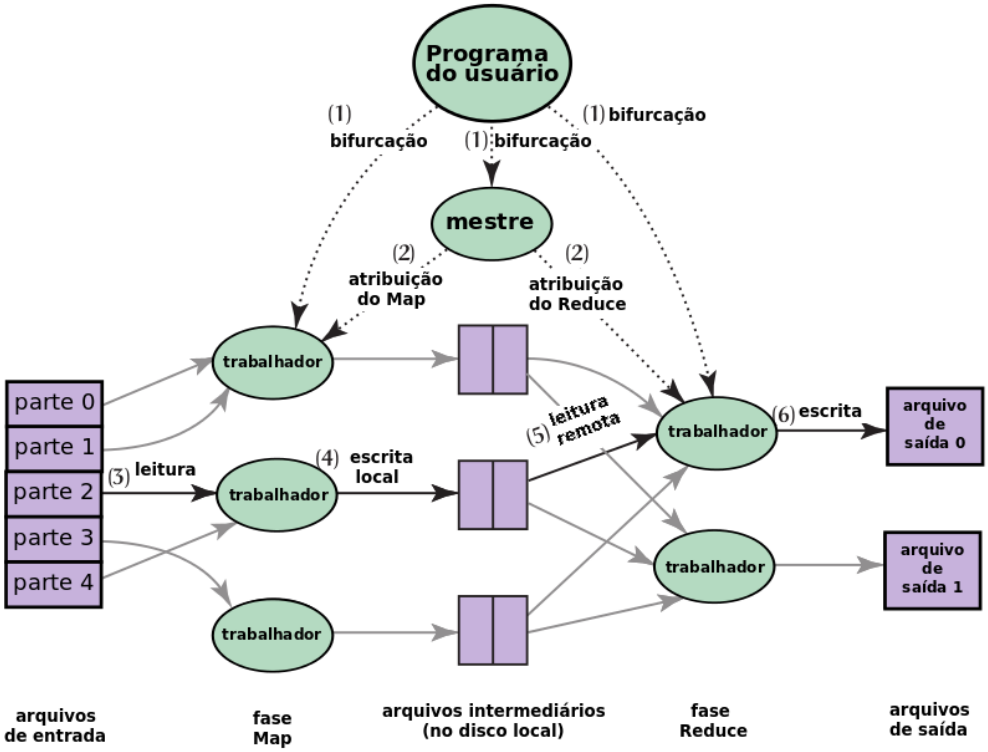

Figura 2.7. Arquitetura *MapReduce* - adaptado de [Jeffrey Dean 2008]

os dados intermediários das suas partições, ele sorteia as chaves intermediárias para que todas as ocorrências das mesmas chaves sejam agrupadas. O sorteio é necessario porque muitas chaves diferentes *Map* são da mesma tarefa *Reduce*. Se um conjunto de dados intermediários é muito grande para ser alocado na memória um sorteio externo é utilizado.

6. Os *workers* de *Reduce* iteram sobre os dados intermediários sorteados e para cada chave única encontrada, ele passa a chave e o correspondente conjunto de valores intermediários para a função de *Reduce* do usuário. A saída da função de *Reduce* é anexada ao final do arquivo de saída para a partição de *Reduce*.
7. Quando todas as tarefas *Map* e *Reduce* são completadas, o *master* reativa o programa. Neste ponto, o *MapReduce* chama o usuário do programa para voltar ao código do usuário.

No sucesso da execução a saída da execução do *MapReduce* fica disponível nos arquivos de saída *R*, um por saída da tarefa de *Reduce*, com os nomes específicos dos usuários. Os usuários não precisam combinar as saídas de arquivos *R* em um arquivo, eles frequentemente passam estes arquivos como entrada para outra chamada *MapReduce* ou usam elas de outra aplicação distribuída que está habilitada para lidar com entradas que estão particionadas em múltiplos arquivos.

As ideias difundidas no modelo *MapReduce* são várias:

1. Escalabilidade *out* e não *up*. Para cargas de trabalho intensivas de dados, um grande número de servidores de *commodities low-end* é preferível a um pequeno número de servidores *high-end*.

2. Assumir que as falhas são comuns. Em escala de *cluster* falhas não são apenas inevitáveis, mas comuns.
3. Mover o processamento dos dados. Em aplicações de computação de alto desempenho tradicionais (*High Performance Computing - HPC*) é comum para um supercomputador ter nós de processamento e nós de armazenamento ligados entre si por uma interconexão de alta capacidade.
4. Processar dados sequencialmente e evitar acesso aleatório. Processamento de dados intensivos, por definição, significa que os conjuntos de dados relevantes são grandes demais para caber na memória e deve ser realizada no disco.
5. Ocultar detalhes no nível do sistema do desenvolvedor do aplicativo. De acordo com muitos guias sobre a prática da engenharia de software escrito por profissionais experientes da indústria, uma das principais razões por que escrever o código é difícil porque o programador deve manter o controle simultâneo de muitos detalhes com curto prazo de memória que vão desde o aspectos triviais aos sofisticados.
6. Alta escalabilidade com baixa coesão.

Algorithm 1 Algoritmo contador de palavras MapReduce

```
1:  function MAP(String chave, String documento)
2:      for all palavra ∈ documento do          ▷ todas as linhas do documento
3:          Emite(palavra, 1)
4:      end for
5:  end function
6:  function REDUCE(String chave, List valores)
7:      soma ← 0
8:      for all v ∈ valores do                  ▷ todos os contadores da lista valores
9:          soma ← soma + v
10:     end for
11:     Emite(chave, soma)
12: end function
```

O exemplo do pseudoalgoritmo 1 representa as funções *Map* e *Reduce* de um contador de ocorrências num grande conjunto de documentos. A função *Map* emite cada palavra, que é a chave, associada com um número um, que é o valor. A função *Reduce* soma todos os valores de mesma chave e emite o resultado associado a chave, que é a palavra. O resultado será a palavra com o número de ocorrências dela no documento.

2.9. Resumo

Neste capítulo foram abordados os paradigmas de sistema distribuído que foram utilizados na plataforma mc^2, bem como os que estão em maior evolução nos dias atuais. Foram também discutidos os principais termos de computação relacionados a sistemas distribuídos. Foi descrito um pequeno histórico sobre estes paradigmas e em alguns casos citados os frameworks que foram usados na plataforma mc^2.

A Tabela 2.1 apresenta as principais diferenças dos paradigmas distribuídos discutidos neste Capítulo. Entre os paradigmas implementados na plataforma mc^2 e que foram descritos nos Apêndices pode-se perceber a diferença do domínio (linha 1), da

Tabela 2.1. Principais diferenças entre os paradigmas de sistemas distribuídos

Características	Cluster	Grade	Par-a-Par	Voluntária	Nuvem	Big Data
1. Domínio	U	M	M	M	M	U/M
2. Nós	K	M	M	M	M	K
3. Segurança	-	+	+	+	+	+/-
4. Custo	++++	+++	+	+	++	+++
5. Granularidade	+	++	+++	+++	++	++
6. S. Operacional	HO	HE	HE	HE	HE	HE
7. Latência	-	+	+	+	+/-	-

quantidade de nós (linha 2) e do sistema operacional (linha 6). Na linha que se refere aos domínios dos computadores destes tipos de sistemas distribuídos, a sigla ”U”representa um domínio único e a sigla ”M”representa domínios múltiplos. Na linha que se refere a quantidade de nós disponíveis para o processamento, a sigla ”K”refere-se a milhares de nós e a sigla ”M”refere-se a milhões de nós. Na linha que representa os tipos de sistemas operacionais, a sigla ”HO”refere-se aos sistemas homogêneos e a sigla ”HE”aos sistemas heterogêneos. O paradigma de computação voluntária e computação par-a-par permitem sistemas heterogêneos e geograficamente distribuidos, o que facilita a adesão de nós no sistema. Porém, deve-se lembrar que na computação voluntária, o que é heterogêneo é o paradigma de computação distribuída, as aplicações que são executadas sobre esse paradigma estão limitadas aos sistemas operacionais a que elas foram desenvolvidas. Por isso, percebe-se que eles pertencem a um domínio múltiplo e com maior quantidade de nós que os outros sistemas. Os computadores dos sistemas de computação intensiva de dados também podem pertencer a um único domínio, assim como a domínios diferentes, como é o caso do BOINC-MR [Costa et al. 2011]. Uma característica importante destes sistemas é a segurança de processamento (linha 3). Este tipo de segurança refere-se a acuracibilidade dos dados processados. Apenas os sistemas de computação em agregados foi dispensado o uso devido ao fato dos computadores pertencerem a um único domínio. No caso dos sistemas de computação intensiva de dados, alguns não precisam de segurança, pois estão numa rede privada, outros que são executados na Internet precisam. O custo dos sistemas (linha 4) que são de domínio único são mais altos que os outros. Todavia, os domínios públicos mais antigos, computação em agregados, têm maior custo que os mais novos, como é o caso da computação em nuvem. Vários estudos sobre o custo de computação em nuvem indicam que o maior gasto das empresas com este tipo de sistema distribuído é devido ao mal preparo dos profissionais para usá-lo. Os custos de sistemas de computação voluntária e computação par-a-par são mais baixos devido ao baixo acoplamento e a facilidade de expansão destes sistemas. A granularidade fina (linha 5) representa a quantidade de pequenas tarefas para serem processadas por estes tipos de sistemas. Portanto, quanto maior a granularidade, mais fina ela será, e maior o número de tarefas um processo pode ser subdividido. A latência (linha 7) é representada pelo sinal -”, que indica baixa latência entre os nós do sistema distribuído, e pelo sinal ”+”, que indica alta latência entre os nós do sistema distribuído. A baixa latência é consequência do domínio de rede ser único para os participantes dos sistema, e o oposto para o outro tipo de sistema. No caso dos sistemas de computação em nuvem, a latência pode variar de acordo com o nível de contrato estabelecido entre o usuário e o fabricante.

Capítulo 3

O projeto mc^2

3.1. Introdução

O projeto mc^2, Minha Cloud Científica, tem como objetivo ser uma plataforma de computação em nuvem para oferecer apoio a aplicações científicas. Ele provê serviços de acesso a grande porte computacional por curtos intervalos de tempo, armazenamento, reprodutibilidade de experimentos e controle de proveniência de dados.

O projeto segue o modelo *PaaS* (*Plataform as a Service*), o que permite o fácil desenvolvimento, implantação e execução de serviços personalizados. Existem dois tipos de perfis de usuários que utilizarão o sistema, o cientista e o desenvolvedor. O usuário cientista tem permissão para utilizar o ambiente de *SaaS* (*Software as a Service*), que visa atender as necessidades específicas de um usuário ou de um grupo de usuários com demanda similares. O usuário desenvolvedor tem um perfil mais voltado para o desenvolvimento e implantação de sistemas distribuídos. Ele estará habilitado para configurar os níveis de *PaaS* e *IaaS* (*Infrastructure as a Service*).

Para que o usuário desenvolvedor possa implantar aplicações *SaaS* na plataforma mc^2 ele deve acessar uma interface Web e configurar os elementos necessários que dão suporte específico a sua aplicação. Após esta implantação, o usuário cientista poderá acessar sua aplicação *SaaS* através do navegador de sua preferência. Desta forma, cada usuário terá acesso único ao seu serviço de computação em nuvem que dá suporte a sua aplicação, dando-lhe a impressão de que ele opera sua própria nuvem particular.

A arquitetura da plataforma mc^2 está dividida em três camadas e é apresentada na Figura 3.1. A camada *IaaS* é responsável por gerenciar os recursos computacionais distribuídos disponíveis, incluindo funcionalidades de monitoramento, escalonamento de tarefas e tolerência a falhas. A camada *PaaS* é responsável por disponibilizar ferramentas Web acessíveis pelos desenvolvedores, para que eles possam configurar portais de aplicações científicas personalizadas, construídas a partir de um conjunto de ferramentas computacionais pré-definidas e acessíveis ou configuráveis por cientistas. A camada *SaaS* é responsável por disponibilizar portais Web acessíveis pelos cientistas, permitindo-os executar aplicações científicas configuradas pelos desenvolvedores e ainda compartilhar os resultados e detalhes destas aplicações entre eles.

No projeto mc^2 foi utilizado o framework CSGrid [Lima et al. 2005] de

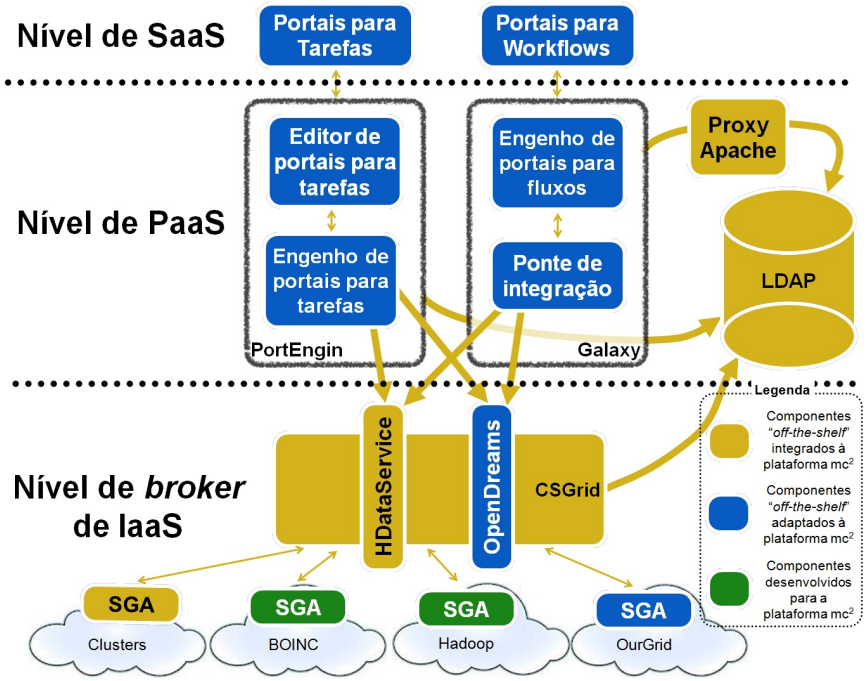

Figura 3.1. Arquitetura mc^2

computação em grade. Existem vários outros frameworks de computação em grade como o NGrid [URL 2014s] escrito em C#, o GrigGain [URL 2014k], o Hazelcast [URL 2014r] ou o Dacframe [URL 2014y]. Porém, foi utilizado o CSGrid porque as outras universidades que participam do projeto mc^2 já tinham uma versão deste framework adaptada para o acoplamento com outros componentes do projeto e o fato de que este framework tenha sido desenvolvido pela PUC-RIO no Brasil.

No projeto mc^2 foi decidido utilizar o *middleware* BOINC de computação voluntária, apesar de existirem outros *middlewares* que oferecem características semelhantes. Outros exemplos de *middlewares* de computação voluntária são o Condor [J. B. M. Litzkow 1997], o Entropia [Chien et al. 2003] e o Xtremweb [Fedak et al. 2001], porém o BOINC tem mais suporte da comunidade científica, mais de 2 milhões de usuários, vários projetos associados, quase 9 milhões de hosts e uma média maior que 8 mil TeraFLOPS[1] de operações por segundo [URL 2014a].

O Galaxy [Goecks et al. 2010] possui duas visualizações na plataforma mc^2. A primeira reflete os fluxos científicos criados pelos usuários desenvolvedores na camada *PaaS*. A segunda reflete os portais criados para os usuários cientistas na camada *SaaS*. Para a execução dos fluxos, o Galaxy divide os fluxos em tarefas para cada ferramenta que os compõe e as submete para a infraestrutura de execução configurada. Novas infraestruturas de execução podem ser adicionadas ao se implementar novos *Job runners*, que têm a responsabilidade de efetuar o escalonamento e despacho da execução das tarefas para o gerenciador de recursos distribuídos. Detalhes deste framework serão explicados na Seção 3.5.1.

A ferramenta representada pelo ”Engenho de portais para tarefas”, que está no

[1] Assim como explicado na Seção 2.3, porém TeraFLOPS é o múltiplo que equivale a 10^{12}.

nível *PaaS*, é implantada uma única vez num servidor web e pode ser instanciada diversas vezes no nível *SaaS* como diferentes portais de propósito específico, dependendo das configurações destas instâncias [Bastos et al. 2013]. Esta ferramenta foi construída sobre o CSGrid, que oferece os serviços básicos de gerenciamento de dados dos cientistas (*API HDataService*), e de submissão e monitoramento de tarefas (*API OpenDreams*) para o SINAPAD. O Engenho de portais permite que os desenvolvedores instanciem portais por meio da configuração de um conjunto de 5 arquivos XML e da carga desses arquivos no contêiner web, onde a ferramenta está hospedada. Para a plataforma mc^2, o Engenho de portais é estendido como uma interface web de configuração e gerenciamento dos portais científicos sob sua responsabilidade.

Para prover o armazenamento permanente de arquivos, a plataforma mc^2 tem um mecanismo de armazenamento de dados. Estes dados podem ter natureza temporária, gerados durante a execução de tarefas e fluxos, ou arquivos de entrada e saída. A *API HDataService*, fornecido pelo próprio gerenciador de recursos CSGrid [Lima et al. 2005], tem a função de armazenamento distribuído que atende a tais requisitos, com a vantagem de ser diretamente integrado como serviço de armazenamento padrão do CSGrid, eliminando a necessidade de uma camada adicional de integração. Assim como já acontece no engenho de portais do SINAPAD [URL 2014x] (SIstema NAcional de Processamento de Alto Desempenho), o Galaxy também usará o HDataService como repositório para os arquivos, que serão acessados diretamente através dos portais.

Para o requisito de autenticação e autorização de usuários na plataforma mc^2 é necessário uma tecnologia que seja integrável ao SINAPAD, Galaxy e ao CSGrid. Com esta base comum de usuários é possível evitar a duplicação de informação. A tecnologia LDAP [Howes and Smith 1997] é usada para atingir este objetivo pois suportada estes três componentes. O engenho de portais do SINAPAD e o CSGrid já possuem suporte nativo a LDAP, enquanto que o Galaxy provê suporte através do uso de um servidor Apache.

O principal componente do nível de infraestrutura no projeto mc^2 é o CSGrid. Ele é um framework de computação em grade explicado na Seção 3.2, que faz a integração com o portal web e os outros componentes que proveem flexibilidade no processamento das aplicações científicas. O componente BOINC que oferece suporte à computação voluntária é explicado na Seção 3.3. O componente Hadoop que oferece suporte à computação intensiva de dados é explicado na Seção 3.4. Com os diversos *frameworks* disponibilizados no nível de infraestrutura o projeto mc^2 pode atingir certa escalabilidade na medida que são acoplados mais computadores de *hardware* com baixo custo. Essa característica também oferece flexibilidade para o processamento de várias tarefas e grande quantidade de dados. A Seção 3.5 abrange o conceito de computação científica. Nela será abordada sua aplicação em sistemas distribuídos e as respectivas ferramentas acopladas a plataforma mc^2.

3.2. CSGrid

O CSGrid [Lima et al. 2005] é um sistema *middleware*[2] utilizado para computação em grade. Ele é desenvolvido sobre o *framework* CSBase [Lima et al. 2006]. O CSBase é

[2]No contexto da plataforma mc^2 o CSGrid funciona como um *middleware* durante a sua utilização. *Middleware* é um *software* que faz a mediação outros *softwares* e demais aplicativos.

um *framework*[3] para gerenciamento de recursos e execução de algoritmos em um ambiente computacional distribuído e heterogêneo. A motivação para sua utilização é a crescente necessidade do uso coordenado de recursos em ambientes computacionais distribuídos e heterogêneos. Estes tipos de sistemas têm diversas plataformas, aplicações e descentralização de dados. Distribuindo geograficamente estes recursos computacionais, o arquiteto de sistema aumenta consideravelmente o poder computacional para os usuários finais. O CSGrid é um sistema utilizado para gerenciar e integrar estas aplicações distribuídas. Ele fornece um serviço de execução remota de algoritmos em máquinas ligadas nele chamado SGA, sigla para *Servidor de Gerência de Algoritmos*.

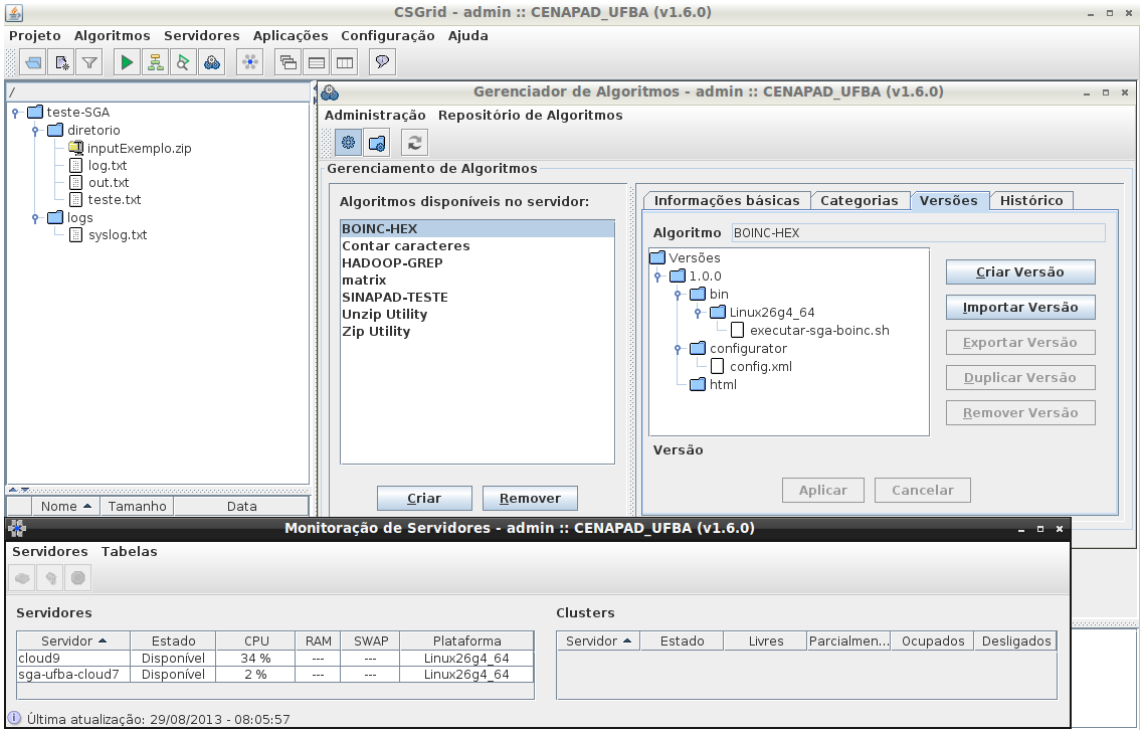

Figura 3.2. CSGrid

Para a interação com o sistema distribuído em grades, o CSGrid fornece uma aplicação *desktop Swing* desenvolvida em Java que é apresentada na Figura 3.2. Através dela os usuários podem monitorar as máquinas da grade, instalar algoritmos, executar aplicações instaladas, comandar a execução remota de algoritmos e acompanhar os processos que estão executando estes algoritmos. Isto é feito de forma transparente para o usuário, ou seja, como se ele estivesse realizando operações na máquina local. O CSGrid é o servidor central. Ele pode ter nenhum ou vários servidores de gerenciamento de algoritmos (SGA's) conectados à ele.

O CSGrid fornece um serviço de controle de acesso de usuários para as áreas de disco, os algoritmos e as máquinas. Para isso, existem classes pré-definidas que permitem a definição de atributos através de interfaces. A parte de gerenciamento de projetos do CSGrid facilita a visualização do sistema de arquivos, suas permissões de leitura e escrita e o envio de notificações de eventos ocorridos.

[3]No contexto do CSGrid sendo desenvolvido, o CBase é um *framework* pois ele é um *software* que segue algumas especificações gerais e deixa disponível para o usuário a especificação de determinadas funções.

3.2.1. SGA

O SGA é um módulo do CSGrid e sua sigla é uma abrevição para *Servidor de Gerência de Algoritmos*. A arquitetura de comunicação do SGA com o CSGrid é apresentada na Figura 3.3.

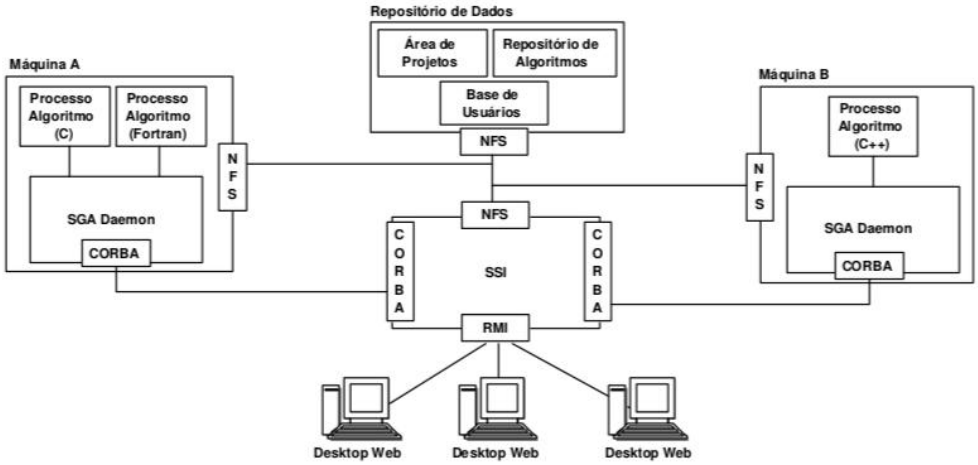

Figura 3.3. SGA - Sistema Gerenciador de Algoritmos [Lima et al. 2005]

Ele é um gerenciador instalado em cada máquina de execução de algoritmos que promove a monitoração do estado de ocupação destas máquinas e dos processos disparados a partir dele. A interação do CSGrid e o SGA é implementada em Lua [URL 2014b] e C++. O SSI (*Server Side Includes*)[4] realiza isto através de CORBA (*Common Object Request Broker Architecture*)[5]. Quando o SGA é ativado, ele registra-se ao SSI e fica disponível aos clientes que se conectam ao SSI. Quando um cliente solicita a execução de um algorítmo através do SSI, o próprio cliente passa a monitorar o processo até a sua conclusão. Se a conexão com o SSI for interrompida, os SGA's em execução permanecem ativos. Quando a conexão se restabelece, o SGA se registra novamente ao SSI.

3.3. BOINC

O BOINC [Anderson 2004] é um *framework* que facilita o trabalho de cientistas que desejam criar e operar recursos públicos em projetos de computação distribuída. Ele foi desenvolvido no Laboratório de Ciências Espaciais na U.C. Berkeley pelo grupo *SETI@home*. O nome BOINC é um acrônimo para *Berkeley Open Infrastructure for Network Computing*. Ele suporta várias aplicações incluindo aquelas de grande armazenamento e alta comunicação. Os proprietários de computadores pessoais podem participar de múltiplos projetos BOINC e também podem especificar como seus recursos computacionais são alocados entre estes projetos.

O objetivo geral do projeto BOINC é avançar no desenvolvimento do paradigma de computação de recursos públicos. Isto implica em encorajar a criação de vários proje-

[4]SSI é uma linguagem de scripting simples interpretado do lado do servidor usado quase exclusivamente para a web.

[5]CORBA é um padrão definido pelo Object Management Group (OMG) que permite que os componentes de software escritos em várias linguagens de computador e em execução em vários computadores para trabalhar em conjunto.

tos e um alto fracionamento de proprietários de computadores na Internet para participar de um ou mais projetos. Os desenvolvedores pretendem reduzir as barreiras de entrada de recursos públicos nos projetos. Para isso o BOINC utiliza-se de softwares *open-source* para a sua configuração (Linux, Apache, PHP, MySQL e Python). Os projetos baseados no BOINC são autônomos. Eles também não têm autorização ou registro centralizado. Cada projeto opera no seu servidor, por isso é possível o compartilhamento de recursos entre projetos. O BOINC suporta diversas aplicações. Isto provê flexibilidade e escalabilidade de mecanismos para dados distribuídos. Aplicações desenvolvidas em linguagens comuns como C, C++, FORTRAN podem ser executadas como aplicações BOINC através de pouca ou nenhuma modificação.

O BOINC permite que seus usuários especifiquem várias preferências que limitam como e quando o BOINC usará seus recursos, assim como as alterem quando for necessário. (i) O BOINC deve ser ativado se não ocorrer algum movimento de mouse ou entrada no teclado nos últimos três minutos. (ii) Um limite de horas ativas que o BOINC deve computar ou se comunicar. (iii) Um limite de horas de comunicação que o BOINC deve se comunicar. (iv) A confirmação que o BOINC deve usar antes de se conectar para pedir comunicação ao usuário. (v) O intervalo mínimo de tempo de conexão na rede. (vi) Um tempo mínimo entre o acesso ao disco, útil para computadores que funcionando com baixa potência desliguem o disco rígido. (vii) O máximo do espaço em disco usado. (viii) O máximo percentual do espaço em disco utilizado. (ix) O mínimo de espaço em disco livre.

Os participantes voluntários desse tipo de sistema não são confiáveis. Muitos projetos de computação voluntária utilizam computação redundante para minimizar o efeito de dados maliciosos ou *hosts* funcionando incorretamente. Uma expressão do total do poder computacional de pontos flutuantes nos projetos é atingida da seguinte forma [Anderson and Fedak 2006]:

$$X = X_{arrival} * X_{life} * X_{ncpus} * X_{flops} * X_{eff} * X_{onfrac} * X_{active} * X_{redundancy} * X_{share}$$

Onde $X_{arrival}$ é a média da taxa de chegada dos *hosts*. X_{life} é a média do tempo que os *hosts* estão ativos. X_{ncpus} é a média do número de CPUs por *hosts*. X_{flops} é a média de FLOPS por CPUs. X_{eff} é a média da eficiência da CPU. X_{onfrac} é a média em fração. X_{active} é a média de frações ativas. $X_{redundancy}$ é a média de redundância. X_{share} é a média de recursos compartilhados.

Finalmente, o BOINC recompensa os seus participantes. Projetos computacionais baseados em recursos públicos devem promover alguma forma de incentivo com o intuito de atrair e reter participantes. O BOINC promove créditos, uma medição numérica de quantidade de processamento computacional dos recursos que têm contribuído. Existe um sistema de conta que reflete o uso dos tipos de recursos públicos como a CPU, a rede e o disco. Outra forma de retribuição que é descrita nos artigos sobre o BOINC é a possibilidade dos usuários estarem conhecendo mais sobre as pesquisas científicas realizadas mundialmente. O BOINC ainda provê visualizações gráficas e proteção de tela com estas medições.

A Figura 3.4 representa a arquitetura do BOINC. Ela é dividida um esquema de servidor e outro de clientes. O servidor contém os dados para serem processados, o agendador de tarefas, o banco de dados MySQL que grava as informações dos projetos BOINC

e um Web Service para facilitar o acesso do administrador de projetos e dos voluntários que desejam se cadastrar nos projetos. Todas estes componentes fazem referência a determinados projetos implantados no BOINC. No lado do cliente o *software* BOINC é composto apenas de uma interface de comunicação com o usuário e com o servidor. Este componente também é responsável por fazer o *download* da aplicação do servidor BOINC, executá-la e fazer *upload* dos resultados.

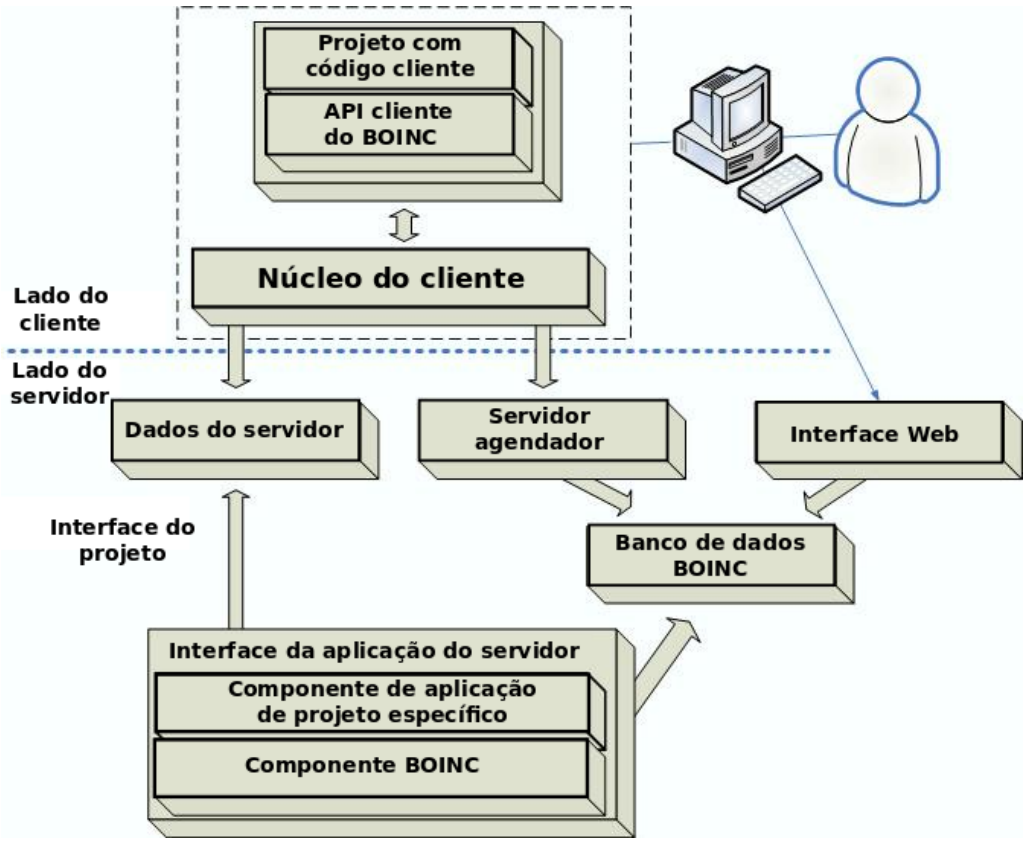

Figura 3.4. Arquitetura do BOINC

3.4. Hadoop

O sistema de arquivos distribuído é um sistema de arquivos no qual os arquivos armazenados estão espalhados em *hardwares* diferentes, interconectados através de uma rede. Eles tem vários aspectos semelhantes aos dos sistemas de arquivos centralizados, além de operações de manipulação de arquivos, preocupações com redundância, consistência e outros atributos desejados de um sistema de arquivos. O Apache Hadoop [White 2009] é uma coleção de subprojetos relacionados sobre uma infraestrutura de computação distribuída. Estes projetos são hospedados pela *Apache Software Fundation* [URL 2014e] que provê suporte para uma comunidade de projetos de software *open source*. O Hadoop é mais conhecido por sua implementação *MapReduce*, seu sistema de arquivos distribuídos *HDFS*, do inglês *Hadoop Distributed FileSystem*, e outros subprojetos que proveem serviços complementares ou oferecem um alto nível de abstração.

O subprojeto *Core* é um conjunto de componentes e interfaces para sistemas de arquivos distribuídos e interfaces de *E/S*. O *Avro* é um sistema de serialização de dados

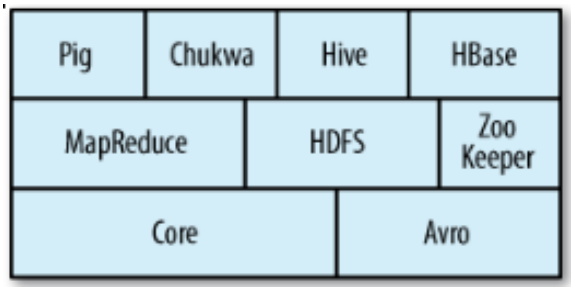

Figura 3.5. Subprojetos do Hadoop [White 2009]

voltado para eficiência e de persistência para armazenamento de dados. O *MapReduce* é um modelo de processamento de dados distribuídos e um ambiente de execução para *clusters* de grande quantidade de computadores de baixo custo. O *HDFS* é um sistema de arquivos distribuídos voltado para *clusters* de grande quantidade de computadores de baixo custo. O *Pig* é uma linguagem para dados de fluxo e um ambiente de execução que explora grande conjunto de dados. Ele é executado no *HDFS* e no modelo *MapReduce*. O *HBase* é uma base de dados distribuída orientada à colunas. O *ZooKeeper* é um serviço de coordenação distribuído de alta disponibilidade. Ele provê funções primitiva como travamento distribuído que pode ser usado para construir aplicações distribuídas. O *Hive* é um *data warehouse* distribuído que gerencia o armazenamento de dados do *HDFS* e provê uma linguagem de consulta baseada em *SQL* para consulta de dados. O *Chukwa* é uma coleção de dados distribuídos e um sistema de análise. Ele executa coletores que armazenam dados no *HDFS* e usa *MapReduce* para produzir relatórios. A Figura 3.5 mostra todos estes subprojetos do Hadoop.

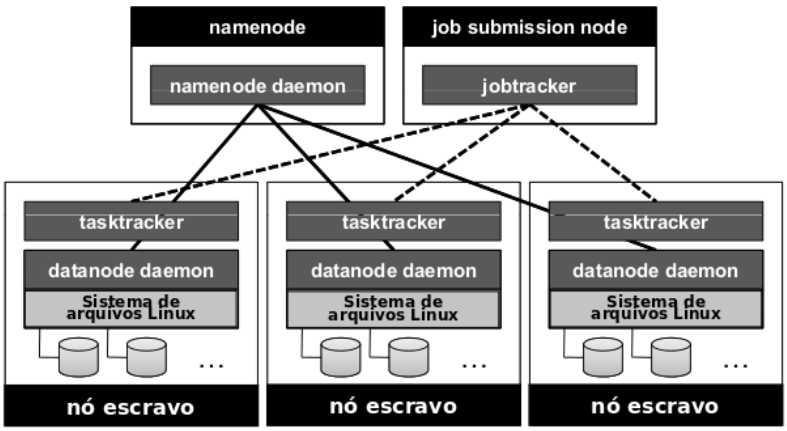

Figura 3.6. Arquitetura do Hadoop - adaptado de [Lin and Dyer 2010]

O *cluster HDFS* tem dois tipos de nós operando no padrão *master-worker*. O *NameNode* que é o *master* e uma quantidade variável de *DataNodes* que são os *workers*. O *NameNode* é que gerencia o sistema de arquivos. Os *DataNodes* são os *workers* do sistema de arquivos *HDFS*. Eles armazenam e redirecionam os blocos quando são requisitados e reportam periodicamente ao *NameNode* uma lista de blocos que estão armazenando. Sem o *NameNode* o sistema de arquivos não pode ser usado. Por esta razão o Hadoop provê dois mecanismos de tolerância à falhas para o *NameNode*. O primeiro mecanismo realiza um *backup* dos arquivos que criam um estado de persistência dos me-

tadados do sistema de arquivos. O segundo mecanismo é a configuração de um segundo *NameNode* que terá o papel de periodicamente copiar a imagem dos metadados. O desenvolvedor submete o job para o nó de submissão de um *cluster*, que no Hadoop é chamado de *JobTracker* e o *framework* cuida de todos os outros aspectos de distribuição de código. Cada nó trabalhador executa um *TaskTracker* para executar tarefas *Map* e *Reduce*. Esta arquitetura pode ser visualizada na Figura 3.6.

3.5. Computação científica

O objetivo geral da computação científica é o processamento de milhares de dados provindos de equipamentos convencionais que não têm essa capacidade acoplada a eles, como microscópios ou telescópios. Estes dados processados por aplicações *e-Science* podem ser melhor analisados na forma de gráficos. De acordo com Jim Gray numa entrevista a NRC-CSTB em janeiro de 2007, *"eScience é o ponto onde "TI encontra cientistas". Os dados atualmente são como um iceberg, as pessoas coletam uma grande quantidade de dados e depois reduzem tudo isto a algum número de colunas em um artigo..."*.

Os sistemas de *workflow* científicos têm a característica de descreverem componentes de computação individuais com suas portas, canais e fluxo entre eles, assim como a coordenação do fluxo entre eles. Os aspectos gerais de um sistema de *workflow* científico são: (i) descrever procedimentos científicos complexos, (ii) automatizar processos de derivação de dados, (iii) computação de alto desempenho (HPC do inglês *High Performance Computing*) para prover vazão e desempenho e (iv) prover gerenciamento e consulta [Zhao and Foster 2008].

O DAGMan [URL 2014n] e o Pegasus [Deelman et al. 2005] são dois sistemas bastante usados em trabalhos de sistemas de *workflow* e têm ampla aplicabilidade em ambientes de computação em grade. O Taverna [Oinn T. 2004] é um sistema de *workflow* de código aberto que tem o foco em aplicações de bioinformática e serviços. Ele é baseado na linguagem XScufl (XML Simple Conceptual Unified Flow). O Triana [Churches D. 2005] é um sistema de *workflow* baseado numa interface gráfica para a coordenação e execução de uma coleção de serviços. O e-Science Gateway [Gannon 2007] é um *framework* interativo que permite a realização de colaborações científicas para recursos em um sistema de computação em grade. Entretanto, este modelo permite que os usuários estejam livres de detalhes complexos relacionados a computação em grade ou qualquer outro *middleware*. Usando estes *frameworks e-Science Gateway* os usuários têm acesso a uma comunidade de dados compartilhados e aplicações que são compostas e programadas em novas aplicações numa linguagem muito natural para cientistas. Geralmente, estes *frameworks* têm sua forma de modelagem num portal Web.

3.5.1. Galaxy

O *framework* Galaxy [Goecks et al. 2010] é um sistema de execução de fluxos científicos com foco em reprodutibilidade, acessibilidade e transparência computacional. Sua plataforma é de configuração de alto nível para fluxos científicos, como demonstrado na Figura 3.7. As ferramentas focam reprodutibilidade com opções de gerenciamento dos passos e compartilhamento de resultados entre usuários do sistema ou de forma pública.

Ele consiste em um conjunto de componentes de *software* reutilizáveis que podem ser integrados em aplicações, encapsulando funcionalidades para descrever interfaces genéricas de ferramentas computacionais, construção de interfaces concretas para que

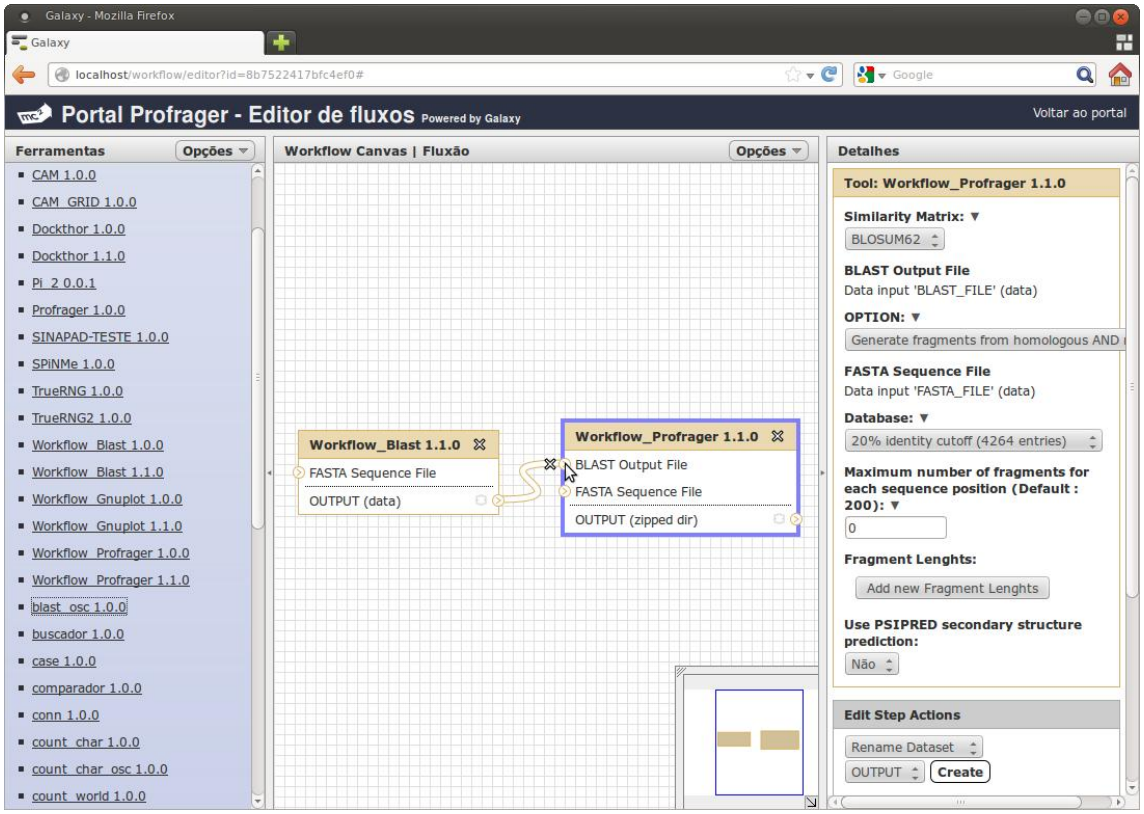

Figura 3.7. Framework Galaxy

os usuários interajam com as ferramentas, invocando essas ferramentas em vários ambientes de execução, lidando com a ferramenta em geral e formatos específicos do conjunto de dados e conversões, e trabalhar com metadados que descrevem conjuntos de dados, ferramentas e suas relações. A aplicação Galaxy é uma aplicação construída usando esta estrutura que oferece acesso às ferramentas através de uma interface (por exemplo, uma interface baseada na web) e fornece recursos para a realização de pesquisa computacional reprodutível, conforme descrito neste artigo. Um servidor Galaxy, ou instância, é uma implementação desta aplicação com um conjunto específico de ferramentas.

3.6. Resumo

No projeto mc^2 foram usados os *frameworks* abordados neste capítulo. Originalmente, o projeto oferece a integração com o OurGrid [Nazareno Andrade 2003] e um *Cluster*, ambos desenvolvidos na Universidade Federal de Campina Grande no LSD - Laboratório de Sistemas Distribuídos [URL 2014q]. A integração com o BOINC foi feita na Universidade Federal da Bahia no LaSiD - Laboratório de Sistemas Distribuídos [URL 2014p]. Estes dois *frameworks* de sistemas distribuídos utilizados têm foco no processamento de grande quantidade de tarefas. A nova implementação integrada no projeto mc^2 oferece uma flexibilidade a mais para o projeto. Com o *framework* de computação intensiva de dados Hadoop é possível implantar aplicações no mc^2 que têm grande quantidade de dados de entrada para processar.

Capítulo 4

Integração de computação voluntária e intensiva de dados

As duas integrações feitas na plataforma mc^2 têm o objetivo de prover maior capacidade de processamento para aplicações científicas, alta abstração para o usuário final, assim como o gerenciamento e compartilhamento de dados processados. Estas integrações foram feitas com dois *frameworks* de computação voluntária e intensiva de dados, respectivamente o BOINC e o Hadoop.

4.1. Introdução

O primeiro *framework* tem com a capacidade de dividir o processamento de uma aplicação para diversos computadores de modo escalável, e o usuário não precisa mais disponibilizar de uma máquina exclusiva para o processamento de uma única aplicação. Os computadores que contém o *Boinc client* se conectam facilmente ao *Boinc server*. Esta solução também oferece a oportunidade da utilização de máquinas de menor poder computacional, que agregadas proveem maior poder de processamento. Para isso foi utilizado o *framework* BOINC que, como explicado na Seção 3.3, fornece suporte à computação voluntária para sistemas distribuídos. O *middleware* BOINC é responsável por criar tarefas para o processamento de arquivos de aplicações implantadas nele.

O segundo *framework* tem a capacidade de suportar programas que seguem o paradigma de programação distribuída *MapReduce*. Este modelo estabelece que os dados a serem processados serão divididos durante a função *Map* e a lógica de processamento será feita na etapa *Reduce*. Após estas duas etapas o resultado é novamente agrupado e é obtido a saída. Para isso foi utilizado o *framework* Hadoop que, como explicado na Seção 3.4, fornece suporte à computação intensiva de dados para sistemas distribuídos.

4.2. Trabalhos relacionados

Existem na comunidade científica diversos frameworks que abordam os paradigmas de computação em grade, computação voluntária, computação em nuvem e aplicações e-sciences. Vários deles são relacionados também com computação científica. Este Capítulo é dedicado à abordagem destes trabalhos com características mais próximas à plataforma mc^2. Estão relatados os trabalhos que focam tanto em computação científica,

quanto aqueles que utilizam os paradigmas de sistemas distribuídos de computação voluntária e computação intensiva dados, sozinhos ou em conjunto. A Tabela 4.1 mostra uma comparação inicial destes trabalhos. Logo depois é possível verificar um pequeno resumo sobre estes trabalhos e sua relação com a plataforma mc^2.

Tabela 4.1. Trabalhos relacionados

Projeto	**Modelos**	**Características**
Attic	Comp. voluntária e p2p	Dinamismo para distribuição de dados.
EDGe	Comp. em grade e voluntária	Arquitetura genérica e de fácil extensão.
BOINC_MR	Comp. voluntária e intensiva de dados	Maior poder computacional.
Globus Toolkit 4	Comp. em grade	Arquitetura orientada à serviços, robusto, desempenho, usabilidade, padronização.
PBJIM	computação em grade	Interoperabilidade de grades heterogêneas.
Gromacs 4	Comp. em nuvem	Suporte à app. científicas.
Science Clouds	Comp. em nuvem	Suporte à app. científicas.
CloudBLAST	Comp. em grade e intensiva de dados	Suporte à app. científicas.
Kepler + Hadoop	Comp. intensiva de dados	Suporte à app. científicas.
MOON	Comp. voluntária e intensiva de dados	Suporte à app. científicas.

O framework Attic [Elwaer et al. 2011] é uma ferramenta para ser usada em conjunto com o BOINC que tem a capacidade de prover um ambiente com maior nível de abstração e dinamismo para a distribuição de dados. A arquitetura do BOINC tem um servidor centralizado e fixo com a cópia dos arquivos do projeto. O Attic faz uso de computação descentralizada e técnicas de enxame para servir dados e o gerenciamento de carga. Este framework mostra que as capacidades de uploading e de tamanho de arquivo podem resultar em largos ganhos para o Attic quando comparado com os dados distribuídos com o BOINC. Portanto, a metodologia de estudo no framework Attic é uma comparação de um sistema centralizado (BOINC) com um descentralizado (peer-to-peer), a plataforma mc^2 se restringe ao uso de modelos cliente/servidor.

O projeto EDGes [Farkas et al. 2010] combina serviços em grade e computação voluntária, integrando estas duas infraestruturas para valorizar as suas vantagens. Com esta característica ele se aproxima muito da plataforma mc^2. Ele faz isso através de uma interface entre estas duas infraestruturas, habilitando a estrutura de computação voluntária para utilizar os recursos livres do serviço em grade. Ele utiliza o framework BOINC para o primeiro modelo de computação distribuída e EGEE (*Enabling Grids for E-Science in Europe*) para o segundo. Sua grande semelhança com a plataforma mc^2 é a disponibilização de um serviço para a comunidade de cientistas e pesquisadores através da EGEE. No

Brasil a plataforma mc^2 utiliza o SINAPAD. No projeto EDGes foi feito uma alteração no *Boinc client* que inicia uma aplicação ao invés de uma unidade de trabalho de um executável.

O projeto BOINC-MR [Costa et al. 2011] combina o paradigma de *MapReduce* [Jeffrey Dean 2008] [Lin and Dyer 2010] com o serviço de computação voluntária oferecido pelo BOINC. Este trabalho tem o objetivo de aumentar o poder computacional de sistemas de computação voluntária permitindo que aplicações mais complexas e paradigmas como o *MapReduce* possam ser executados em recursos dispersos na Internet. No BOINC-MR o modelo *MapReduce* é utilizado nas tarefas enviadas para as máquinas *Boinc client*, ou seja, os modelos de programação de computação voluntária e computação intensiva de dados são utilizados em série. As tarefas são distribuídas pela quantidade de computadores voluntários conectados ao sistema e cada tarefa é processada de acordo com o paradigma *MapReduce* no computador voluntário. Diferente da plataforma mc^2 cujos modelos são utilizados em paralelo.

O Globus Toolkit (GT) [Foster 2005] está sendo desenvolvido desde o final da década de 1990, para apoiar o desenvolvimento de aplicações de computação distribuída orientada a serviços e infraestruturas. O núcleo do GT tem componentes que lidam com questões básicas relacionadas com a segurança, o acesso a recursos, gestão de recursos, movimentação de dados e descoberta de recursos. Estes componentes permitem um amplo conjunto de ferramentas e serviços que constroem o "ecossistema Globus"ou interagem com as principais funcionalidades do GT para fornecer uma ampla gama de funções úteis em nível de aplicativo. Essas ferramentas, por sua vez, foram usadas para desenvolver uma vasta gama de ambas infraestruturas de grade e aplicações distribuídas. Em resumo, as principais características do mais recente lançamento, o Web GT4 baseado em serviços, o que proporciona melhorias significativas em relação às versões anteriores, em termos de robustez, desempenho, usabilidade, documentação, conformidade com padrões e funcionalidade.

O plugin PBJIM [Wu et al. 2007] é baseado na ideia de mecanismos de interoperabilidade para sistemas em grade heterogêneos. A sua motivação foi a grande popularidade e variedade de middlewares para sistemas em grade, junto com a necessidade de comunicação entre estes sistemas e aumento da escalabilidade. Foram usados dois sistemas em grade, o *ChinaGrid* e o *China National Grid*. Eles foram construídos com dois diferentes middlewares, CGPS e VEGA, respectivamente. O PBJIM provê a interoperação do genrenciamento de tarefas destes dois sistemas em grade. Assim como no mc^2, que é projetado para funcionar na rede SINAPAD, o PBJIM foi projetado para funcionar em grandes redes de sistemas em grade na China. Para prover essa interoperabilidade ele utiliza o framework CSGrid. A plataforma mc^2 utiliza o CSGrid para prover comunicação dos diferentes frameworks de sistemas distribuídos e framework de computação científica Galaxy.

O trabalho [Niehörster et al. 2009] é um estudo de aplicações científicas em uma plataforma de nuvem com o cumprimento de acordos de níveis de serviço, conhecidos como SLA (*Service Level Agreement*). Estes níveis de serviço tornam possíveis diferentes configurações usando poucas máquinas de alto desempenho ou múltiplas máquinas de baixo desempenho. As análises também mostram que experiências de análise de desempenho em ambientes físicos não necessariamente são aplicadas em ambientes virtuais. O

comprometimento dos SLAs também dependem significativamente do mapeamento de máquinas virtuais para *hosts* físicos. Este trabalho foca em um tema pouco discutido para aplicações científicas, os SLAs, que também não são abordados na plataforma mc^2.

O projeto *Science Clouds* [URL 2014w] tem o objetivo de executar aplicações científicas em platafromas heterogêneas. Estas plataformas permitem o arrendamento de recursos por períodos de tempo muitos curtos. Os usuários pertencem a uma comunidade de pesquisa que tem acesso a pequenas, médias e largas configurações de máquinas virtuais que podem ser gratuitamente usadas de acordo com a necessidade das aplicações.

O CloudBLAST [Matsunaga et al. 2008] propõe e avalia uma abordagem de paralelização, implantação e gerenciamento de aplicativos de bioinformática que integram tecnologias emergentes de computação distribuída. Ele utiliza o paradigma de *MapReduce* para paralelizar ferramentas e gerenciar sua execução, a virtualização de máquinas para encapsular os ambientes de execução e rede de virtualização para conectar recursos através de *firewall* e NATs, enquanto preserva o desempenho necessário e a comunicação com o cliente. A implementação integra o Hadoop, estações de trabalho virtuais, ViNe como MapReduce, máquinas virtuais e tecnologias de rede virtuais. O CloudBLAST utiliza apenas o modelo de programação intensiva de dados, enquanto o mc^2 trabalha com os dois modelos de programação discutidos nesta dissertação, o de computação voluntária e intensiva de dados, juntamente com *framework* de computação em grade. A organização da platafroma mc^2 segue o modelo de computação em nuvem, dividido em níveis de *SaaS - Software as a Service*, *PaaS - Platform as a Service* e *IaaS - Infrastructure as a Service*.

O projeto Kepler + Hadoop [Wang et al. 2009] é uma arquitetura generalizada que facilita a execução de fluxos científicos de aplicações intensiva de dados. Os cientístas podem utilizar a modelagem MapReduce como seus domínios específicos de problemas e conectá-los através da interface gráfica do Kleper [Altintas et al. 2004], um tipo de workflow científico. A plataforma mc^2 oferece mais flexibilidade que este projeto pois disponibiliza outros modelos de computação distribuída para o usuário cientísta e a integração de workflows entre estes modelos.

O projeto MOON [Lin et al. 2010] propõe uma extensão do modelo de programação *MapReduce* do Hadoop que aborda o conceito de computação voluntária. O Hadoop já oferece um sistema de tolerância à falhas que é baseado na replicação dos dados das tarefas *Map* e *Reduce* para suprir eventuais falhas nos nós do sistema. Porém, o MOON provê um serviço dinâmico e multi dimensional de replicação para tolerar ambientes de computação voluntária altamente voláteis. Para isso, ele classifica os *workers* do Hadoop como nós dedicados ou voláteis, que armazenam arquivos confiáveis e transientes, respectivamente. Os nós voláteis podem sair da rede a qualquer momento, pois seus dados, como ainda não estão completamente processados, serão reprocessados em outras máquinas. Entretanto, é necessário que exista um pequeno grupo de máquinas confiáveis, ou seja, que não sairão do sistema de forma alguma. Estas máquinas irão guardar os arquivos já processados. Com essa estratégia o MOON acelera o processamento em três vezes para o Hadoop. A principal diferença em relação a plataforma mc^2, é que o MOON é uma extensão da característica de tolerância à falhas do Hadoop. O mc^2 utiliza as modelagens de computação intensiva de dados e computação voluntária para compartilharem dados independentes.

4.3. O SGA para o BOINC e para o Hadoop

O SGA BOINC foi desenvolvido para a administração do servidor BOINC. A sua função é criar tarefas no BOINC através do CSGrid. Sendo assim, o CSGrid funciona como uma interface para o gerenciamento das aplicações implantadas no BOINC. A Figura 4.1 demonstra a arquitetura entre o *Boinc server*, o CSGrid, o SGA e as máquinas com *Boinc clients*. Cada *framework* foi instalado em um servidor diferente para compor a topologia de rede da plataforma mc^2. As máquinas com *Boinc clients* podem ser usadas de modo escalável. A partir do nó do CSGrid as requisições enviadas seguem o conceito de *BoT* (*Bag-of-Tasks*) [Cirne et al. 2003]. Todas as tarefas são divididas para os computadores clientes e o usuário que acessa o portal Web receberá o resultado da *BoT*. O SGA Hadoop seque o mesmo esquema de arquitetura, exceto pela substituição so servidor BOINC pelo software Hadoop *master* e os clientes BOINC pelos Hadoop *workers*. Os dados que serão processados chegam através de uma tarefa no servidor Hadoop e são distribuídos nos computadores conectados ao servidor através do *framework* deestensão do Hadoop *HDFS*. O processamento vai então aos dados e executa o que é necessário para colher os resultados. Neste momento o processamento já é em paralelo. Os resultados são novamente agrupados e retornam ao uauário final.

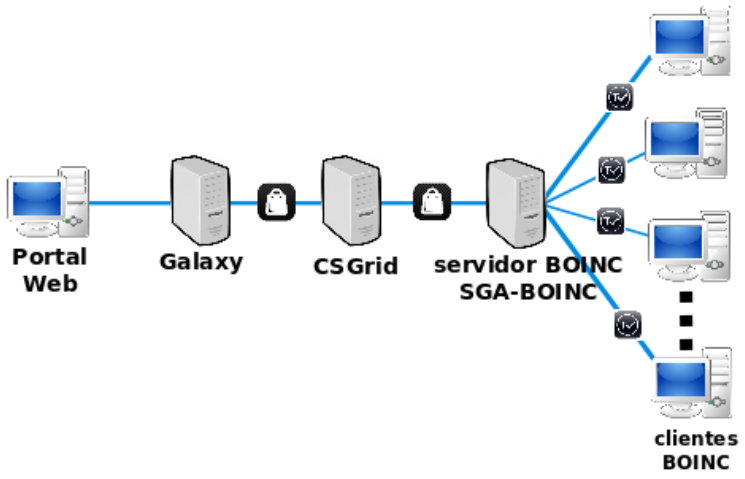

Figura 4.1. Topologia da comunicação do mc^2 com o BOINC

O programa SGA é instalado na máquina do *Boinc server* por um usuário desenvolvedor. Através da interface do SGA é possível configurar quantas tarefas o usuário deseja criar e selecionar um arquivo de dados de entrada para ser processado. A Figura 4.2 apresenta a interface do SGA. Ela é acessível através do CSGrid. Nesta interface é possível configurar quantas tarefas serão criadas no *BOINC server* e quais os arquivos de entrada de dados serão passados para essas tarefas processarem.

Depois que a tarefa é processada completamente, o resultado é transferido da máquina *BOINC client* para a máquina *BOINC server* automaticamente pelo próprio *middleware* BOINC. Ele mesmo se encarrega de verificar se os dados processados não foram adulterados, através de uma configuração de *quorum* especificada na criação das tarefas. A configuração original do BOINC oferece uma redundância de processamento duplo

Figura 4.2. Interface do SGA

para cada tarefa por padrão. Porém, como a plataforma mc^2 está sob uma rede confiável, o BOINC foi configurado para não ter redundância no processamento das tarefas.

A Figura 4.3 representa a arquitetura que foi criada, tanto para a integração de computação voluntária como de computação intensiva de dados, para atender a necessidade de comunicação entre o CSGrid e os SGA's do BOINC e do Hadoop, assim como para alguns exemplos de aplicações científicas. O SGA, tando do BOINC como do Hadoop, são componentes do CSGrid, mas que não estão necessariamente na mesma máquina física. Na implementação da plataforma mc^2 eles foram dispostos nas mesmas máquinas dos *frameworks* BOINC e Hadoop. Os SGA's realizam a comunicação com estes *frameworks* para criar as tarefas e receber o resultado delas. O primeiro processo é realizado pela função *create-task* e após ela é executada a função *query-result* periodicamente, numa frequência de 10 segundos, para verificar se a tarefa terminou. Quando a função *query-result* notifica que o resultado está disponível ele retorna o resultado para o CSGrid. Se o processamento da tarefa terminou e o resultado não está correto, o script *query-result* retorna um código de erro para o CSGrid.

Neste esquema de arquitertura foram descritas as comunicações com as aplicações científicas MEGA-CC para o BOINC. Porém será visto no próximo Capítulo que outras aplicações científicas foram implantadas nestes *frameworks*. Após o resultado das tarefas estarem disponíveis no CSGrid, a integração *PaaS* da plataforma mc^2 realiza a disponibilização dos dados no Galaxy. Neste ponto já é possível verificar que toda parte da implementação realizada na plataforma mc^2 descrita nesta dissertação é focada no nível *IaaS*. O gerenciamento de dados a serem processados, tanto no BOINC como no Hadoop, é implementado pelos próprios *frameworks*, assim como demonstra a simbolização do banco de dados MySQL do BOINC e o sistema de arquivos distribuídos *HDFS* do Hadoop. Os detalhes da implementação do SGA para o BOINC está descrito no Apêndice A.2 e para o Hadoop está descrito no Apêndice B.2.

4.4. Resumo

Neste Capítulo foi visto a integração de dois módulos de computação distribuída na plataforma mc^2. Baseado nestas integrações foi possível analisar diversos trabalhos relaciona-

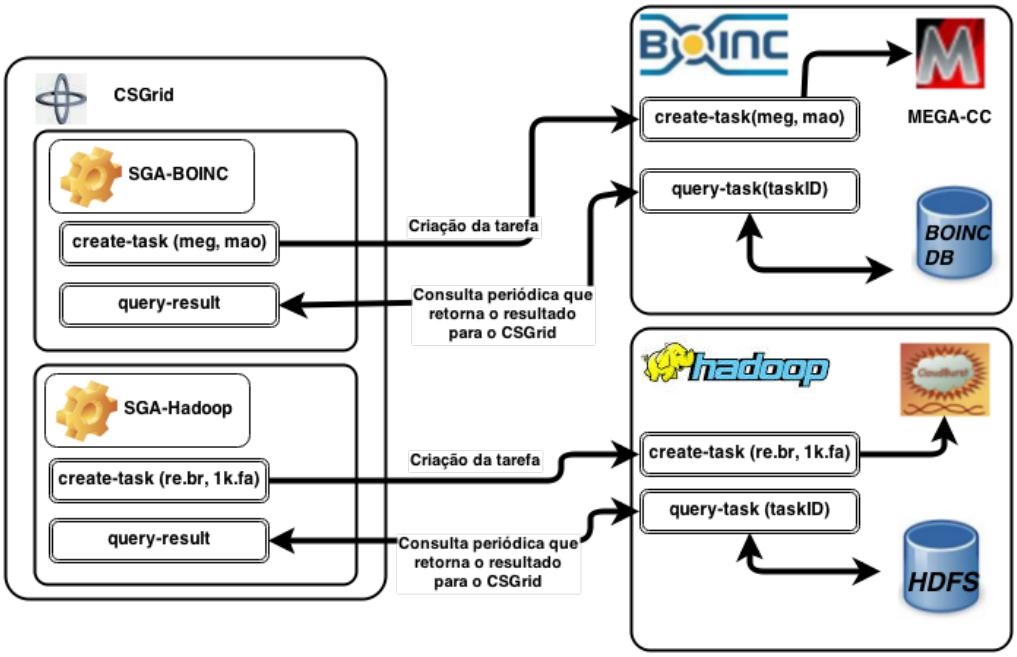

Figura 4.3. Fluxo do SGA para o BOINC e para o Hadoop

dos com a presente proposta da plataforma mc^2 e os tipos de *frameworks* de computação distribuída utilizados nela. O BOINC que fornece suporte à computação voluntária e o Hadoop que fornece suporte à computação intensiva de dados. Foi visto também que estes dois modelos oferecem uma flexibilidade de processamento de tarefas. O primeiro permite que várias tarefas sejam executadas em paralelo e o segundo permite que uma tarefa seja subdividida em várias tarefas menores que também podem ser executadas em paralelo. Além destas características, estes frameworks fornecem segurança e confiabilidade no processamento de dados, garantido a corretudo do resultado. O principal objetivo deste Capítulo foi mostrar as interfaces de acoplamento entre estes dois *frameworks* com o CSGrid, feita através dos SGA's. Usando o SGA é possível o gerenciamento destes *frameworks* remotamente.

Capítulo 5

Estudo de casos

5.1. Introdução

Uma das integrações feita na plataforma mc^2 tem o objetivo de prover maior capacidade de processamento para aplicações científicas. Através de um *framework* com o poder de dividir o processamento de uma aplicação para diversos computadores de modo escalável, o usuário não mais precisa disponibilizar de uma máquina exclusiva para o processamento de uma única aplicação. Esta solução também oferece a oportunidade da utilização de máquinas de menor poder computacional, que agregadas proveem maior poder de processamento. Para isso foi utilizado o *framework* BOINC que, como explicado na Seção 3.3, fornece suporte à computação voluntária para sistemas distribuídos. O *middleware* BOINC é responsável por criar tarefas para o processamento de arquivos de aplicações implantadas nele.

5.2. Estudo de caso em computação voluntária

Foram feitos quatro estudos de caso com o *framework* BOINC na plataforma mc^2. Com estes diferentes estudos foi possível avaliar o desempenho em relação a escalabilidade de processamento da plataforma mc^2 no que se refere a implementação de sistemas de computação voluntária. A escolha das aplicações científicas para a implantação no BOINC e o desenvolvimento do *middleware* para a plataforma mc^2 seguiram vários aspectos relacionados ao estudo de sistemas distribuídos. A primeira aplicação, do estudo de caso 5.2.1 foi escolhida pelo motivo de já estar implantada em outro módulo da plataforma mc^2. A aplicação do estudo de caso 5.2.2 foi desenvolvida para que pudesse atender aos requisitos de métricas de tempo de resposta versus incremento de carga. A aplicação do estudo de caso 5.2.3 foi escolhida por ser um *benchmark* de conhecimento geral da comunidade científica de sistemas distribuídos. A aplicação do estudo de caso 5.2.4 foi escolhida por ser de uso atual e cotidiano no Laboratório de Genética e Populações e Evolução Molecular do Instituto de Biologia da UFBA.

5.2.1. Estudo de caso com a aplicação Hex

O primeiro estudo de caso que foi implantado no projeto mc^2 diz respeito a uma aplicação científica da área de biologia molecular. A aplicação Hex [Ritchie 2003] é um *software* de acoplamento interativo de proteínas e superposição de moléculas. Este programa reconhece estruturas de proteínas e DNA em arquivos de formato PDB [URL 2014o] (*Protein*

Data Bank) e SDF (*Standard Database Format*) que descrevem moléculas pequenas. A aplicação Hex pode ser encontrado no portal INRIA [URL 2014l]. Primeiramente, esta aplicação foi testada fora do plataforma BOINC e foram observadas suas diversas características de funcionamento.

O fluxo de execução da camada *IaaS* do mc^2 começa no CSGrid, passa pelo servidor BOINC através do SGA-BOINC e chega até os clientes BOINC. Quando o processo termina os arquivos voltam para o servidor BOINC e o nível de *PaaS* se encarrega da transferência dos dados do CSGrid para o portal Web. O processo que envolve o servidor BOINC e os clientes BOINC é automático, depois que os clientes estão cadastrados no servidor. Portanto, uma boa medição a ser realizada é a comparação do tempo de execução da aplicação Hex sozinha e acoplada a plataforma mc^2. Neste segundo aspecto, o tempo de processamento envolve todo o processo CSGrid, servidor BOINC e cliente BOINC.

Para medir o tempo de execução da aplicação Hex implantada na plataforma mc^2, foi inserido um comando para criar um arquivo no script de criação de tarefas no servidor BOINC. Quando o cliente BOINC retorna os dados processados para o servidor BOINC é possível comparar o tempo de criação destes dois arquivos. Esta diferença é o tempo total de processamento. A Tabela 5.1 apresenta estes resultados.

Tabela 5.1. Tempo de processamento do Hex

Aplicação	Tempo em segundos
Hex	19,23
Hex na mc^2	22,01

As duas medições foram feitas em uma mesma máquina em que um cliente BOINC estava instalado. A máquina possui um processador Core 2 Duo CPU E4400 2.00GHz x64 bits e memória de 1 Giga. Através desta medição é possível concluir que o tempo extra de três segundos está relacionado a transferência de dados entre as máquinas do CSGrid, do SGA do *boinc server* e a máquina com o *boinc client*. Apesar do aumento da latência do processamento para as tarefas na plataforma mc^2, é possível considerar o resultado satisfatório, já que os dados poderão ser compartilhados com os outros componentes da camada *IaaS* da plataforma mc^2. Os componentes do OurGrid e do Cluster poderão usar estes dados como quizerem através do Galaxy. Os detalhes sobre a implementação desta integração está descrita no Apêndice A.3. Outros tipos de medições foram realizadas nos próximos estudos de caso.

5.2.2. Estudo de caso com uma aplicação C++

O segundo estudo de caso foi com um programa em C++ que usa a técnica de ordenamento por bubblesort[1]. Ele lê um arquivo com números aleatórios e escreve um arquivo com os números ordenados. Para implantar o programa C++ no BOINC foi necessário incluir algumas bibliotecas e funções do BOINC, que inicializam e finalizam o processo. O programa final é maior que o original, pois é necessário utilizar as bibliotecas do BOINC

[1]A técnica de ordenação bubblesort, ou ordenação por flutuação, recorre a ideia é percorrer o vector diversas vezes, a cada passagem fazendo flutuar para o topo o maior elemento da sequência.

para iniciar e parar as tarefas na máquina cliente. Os detalhes de implementação deste programa e sua integração com o BOINC estão descritos no Apêndice A.4. Pensando nisso, as medições que foram feitas com o programa incluiram as bibliotecas do BOINC, tanto quando ele foi executado sozinho quanto na plataforma BOINC.

A Tabela 5.2 apresenta os resultados do programa sendo executado sozinho e na plataforma BOINC com diferentes quantidades de números para ordenar. Pode ser percebido que a diferença de tempo de processamento nas duas ocasiões permanece estável conforme a quantidade de números para ordenar aumenta. Portanto, o processamento não é prejudicado ao se implantar o programa na plataforma BOINC.

Também foi feita uma medição da aceleração do processamento do programa sozinho e implantado na plataforma mc^2. Esta métrica utilizada foi de referência de um artigo que teve o objetivo de interpretar aplicações implantadas na plataforma BOINC [Gonzalez et al. 2008]. Na equação de *Speedup* 5.1 o parâmetro T_{seq} é o tempo de execução do programa em modo sequencial e o parâmetro T_{mc^2} é o tempo de execução do programa na plataforma mc^2. Os resultados são apresentados na última coluna da Tabela 5.2.

$$A = \frac{T_{seq}}{T_{mc^2}} \tag{5.1}$$

Tabela 5.2. Tempo de processamento do programa Bubblesort

Quantidade	Tamanho (Kb)	Standalone (min)	BOINC (min)	Aceleração
100.000	560,539	1,4393	2,1929	0,6563
250.000	1.508,560	5,2126	6,3829	0,8166
500.000	3.181,231	20,4487	21,5930	0,9470
750.000	4.886,164	46,3036	47,1282	0,9825
1.000.000	6.604,130	82,4396	84,0247	0,9811
1.250.000	8.356,177	129,2335	130,0145	0,9939

O artigo [Anderson and Fedak 2006] também faz referência a métricas em computação voluntária. A métrica discutida é a potência computacional total dos pontos flutuantes disponíveis no projeto.

Tabela 5.3. Tempo de processamento paralelo do programa Bubblesort

Tamanho (Kb)	4 arq	8 arq	12 arq	16 arq	20 arq	24 arq
560,539	2,3909	3,4605	4,0975	5,5348	6,2184	8,2448
1.508,560	6,2075	11,2920	17,1688	23,3935	29,2433	34,5128

Para demonstrar que conforme se aumenta o número de tarefas no módulo de infraestrutura do BOINC a velocidade de processamento aumenta, foi proposto um gráfico. Ele tem no eixo das abscissas a quantidade de arquivos processados, que correspondem ao número de tarefas para cada mediação. No eixo das ordenadas temos o tempo de processamento em minutos sobre o número de tarefas processadas. Em cada conjunto de

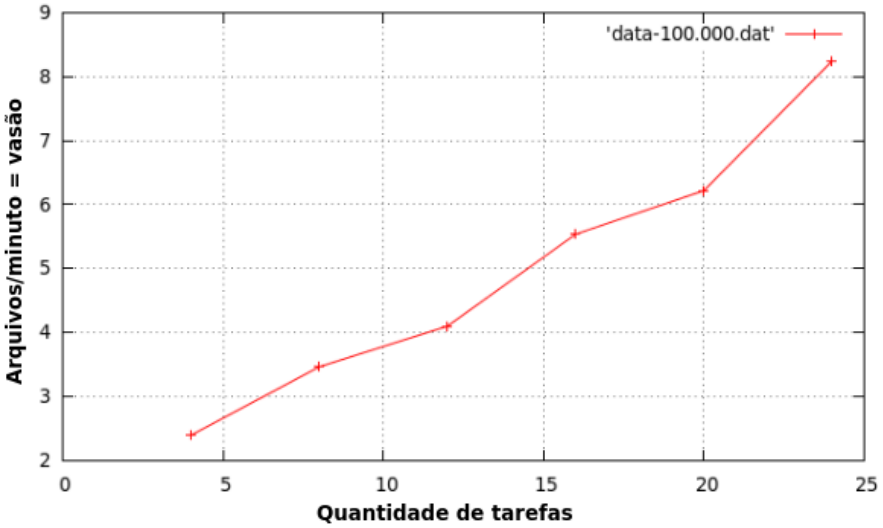

Figura 5.1. Métricas para o programa Bubblesort com 100.000 números na plataforma mc^2

mediação os arquivos processados são identicos, com isso é possível certificar que o processamento de cada tarefa também será o mesmo. A Figura 5.1 demonstra que quando o número de tarefas é maior, a vazão tende a aumentar.

5.2.3. Estudo de caso com o benchmark NAS

O estudo de caso realizado com o *benchmark* consolidado na comunidade científica, o *NPB-NAS Parallel Benchmarks* [URL 2014t], teve o objetivo de utilizar toda a capacidade de processamento dos *BOINC clients*. Este *benchmark* consegue ocupar toda a memória disponível de uma máquina quando executado. Os Benchmarks Paralelos NAS (NPB) são um conjunto de benchmarks paralelos para análise de desempenho de computadores. Estes critérios consistem em cinco núcleos paralelos e três *benchmarks* de aplicação simulados. Mais detalhes da sua integração são descritos no Apêndide A.5.

Foram realizadas medições do processo integral da execução deste *benchmark*, desde o CSGrid até o *BOINC client*. A Tabela 5.4 faz referência a estas medições. Com estes resultados pode ser verificado que o aumento do tempo de execução do *benchmark* foi devido ao tempo de comunicação e a transferência de arquivos entre o CSGrid e o *framework* BOINC.

Tabela 5.4. Tempo de processamento do benchmark NAS

Aplicação	minutos
NPB3.3-OMP	39,12612
NPB3.3-OMP com CSGrid	40,0128

5.2.4. Estudo de caso com o MEGA-CC

O quarto estudo de caso realizado sobre o BOINC na plataforma mc^2 foi com a aplicação científica MEGA-CC [Kumar et al. 2012]. O MEGA-CC é um núcleo com várias subrotinas voltadas para o estudo de questões genéticas. O algorítmo *mode selection* do MEGA-CC, umas das subrotinas do MEGA-CC, que foi utilizado neste estudo de caso, faz um

teste de *goodness of fit* para modelos de substituição nucleotídica ou de aminoácidos baseado no conteúdo de máxima verossimilhança. A saída do MEGA-CC provê um arquivo com dados separados por vírgula com 24 e 48 possibilidades de modelos respectivamente para nucleotídeos e ácido amino. Para medir o tempo de execução, a mesma função que cria tarefas no *BOINC server* também cria um arquivo onde os resultados de saída irão ser transferidos dos *BOINC clients*. Quando os clientes transferem o resultado de volta para o *BOINC server* é possível comparar o tempo que eles levaram para executar. Mais detalhes sobre essa integração pode ser encontrado no Apêndice A.6. Como o objetivo da plataforma mc^2 é permitir que os processos sejam paralelos, o primeiro experimento considerou muitas tarefas em uma mesma máquina. Esta máquina tinha como configuração um processador Intel Core 2 Duo com 2,13 GHz e uma memória de 1GB. Como a máquina tinha um processador com dois núcleos, o primeiro teste foi feito com duas tarefas. Na medida que foram adicionadas mais duas tarefas, estas ficaram na fila atrás das duas primeiras. Nesta segunda medição já foi possível notar um ganho devido a sobreposição do tempo de processamento das tarefas e a transferência de arquivos entre os *frameworks*. Portanto, foi possível verificar que a velocidade do processamento da plataforma mc^2 aumenta de acordo com o número de tarefas no BOINC. A Tabela 5.5 apresenta os resultados em segundos para diferentes quantidades de tarefas e o tempo esperado se a paralelização não fosse possível. De acordo com a Equação de *Speedup* 5.1 é possível verificar a aceleração do processo.

Tabela 5.5. Comparação em segundos da execução sequencial do MEGA-CC sozinho e na plataforma mc^2.

App	1 tarefa	2 tarefas	4 tarefas	6 tarefas	8 tarefas
Mega	244,50	572,00	1144,00	1716,00	2288,00
mc^2 + Mega	286,00	308,00	571,00	825,50	1106,00
Aceleração	0,85	1,85	2,00	2,07	2,06

A Tabela 5.5 apresenta os testes feitos com o MEGA-CC no ambiente local e na plataforma mc^2. O tempo de execução local para o conjunto de dados de proteínas retirado do exemplo 2.1 do livro [Nei and Kumar 2000] com seis taxa e 160 sites foi de 244,5 segundos. O mesmo conjunto de dados foi utilizado neste estudo de caso e o tempo aumentou em 41,5 segundos por causa da transferência de arquivos entre os *frameworks* da plataforma mc^2. A partir da segunda coluna da primeira linha o tempo de execução sequencial foi multiplicado pela quantidade de tarefas, pois o programa executado sequencialmente teria um acréscimo de tempo constante. Na segunda linha é apresentado o tempo do MEGA-CC executado na plataforma mc^2 e o incremento de tarefas. É possível verificar o ganho de tempo com as execuções em paralelo das tarefas. A terceira linha mosta a aceleração de acordo com a Equação 5.1 de Speedup. O primeiro resultado mostra um perda de aceleração, porém depois é observado o ganho de acordo com o número de tarefas.

A Figura 5.2 mostra o grafico que representa a comparação da execução da aplicação MEGA-CC localmente e na plataforma mc^2. Ela demonstra que conforme é incrementado o número de tarefas o desempenho do processamento também aumenta. Ela é um gráfico que relaciona o número de tarefas processadas com o tempo em segun-

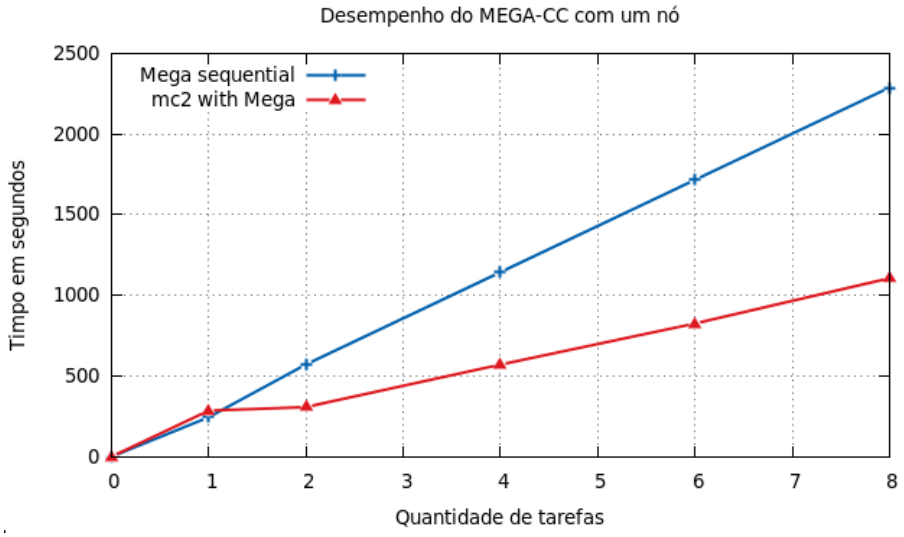

Figura 5.2. Comparação da execução sequencial do MEGA-CC sozinho e na plataforma mc^2.

dos das tarefas processadas. A primeira tarefa processada na plataforma mc^2 levou mais tempo para processar do que localmente, isto por causa da transferência de arquivos de entrada e saída. A próxima tarefa claramente demonstra que quando o número de tarefas aumenta o desempenho do processamento geral também aumenta.

A segunda métrica realizada foi com três nós conectados na plataforma mc^2, o primeiro com um processador Intel Celeron 2.66 GHz e 3 GB de RAM, o segundo com um processados Intel Celeron 2.13 GHz e 2 GB de RAM, e o terceiro com um processados Intel Celeron 2.53 GHz com 1,5 GB de RAM. Todos com o sistema operacional Windows 7 Ultimate. Como é possível verificar, para este teste forma utilizados *commodities hardware*, ou seja, computadores que não tem grande utilizade nos dias atuais devido a sua baixa capacidade de desempenho. A tabela 5.6 apresenta os resultados do MEGA-CC localmente e na plataforma mc^2. Os resultados revelam uma constante aceleração do tempo de execução quando as tarefas são executadas na plataforma mc^2, calculado pela Equação 5.1 de Speedup.

A primeira execução sequêncial do MEGA-CC comparado com a execução na plataforma mc^2 obteve uma aceleração de 2,57. A segunda coluna da primeira linha da Tabela 5.2 e as seguintes representam os tempos esperados baseados no tempo de execução da primeira tarefa executando sequencialmente. A segunda linha apresenta a execução do modelo de do MEGA-CC sobre a plataforma mc^2. Com a apresentação destes valores é possível verificar um ganho na execução do sistema com as tarefas em paralelo. A terceira linha apresenta a aceleração, baseada na Equação de Speedup 5.1.

A Figura 5.3 apresenta um gráfico onde é possível verificar o aumento do número de tarefas do MEGA-CC na plataforma mc^2 e o aumento do desempenho de processamento com um fator constante em torno de 2,6. Enquanto que a Figura 5.2 mostra o ganho de muitas tarefas em um mesmo nó executadas de forma sequencial na plataforma mc^2, a Figura 5.3 mostra o ganho de muitas tarefas executadas em diferentes nós e o ganho de paralelização e sequenciamento das tarefas.

Tabela 5.6. Comparação em segundos da execução paralela do MEGA-CC sozinho e na plataforma mc^2.

App	3 tarefas	6 tarefas	9 tarefas	12 tarefas	15 tarefas	18 tarefas	21 tarefas
Mega	1600,50	3201,00	4801,50	6402,00	8002,50	9603,00	11203,50
mc^2 + Mega	624,00	1231,00	1835,00	2434,50	3048,00	3640,00	4232,00
Aceleração	2,565	2,600	2,616	2,630	2,625	2,638	2,647

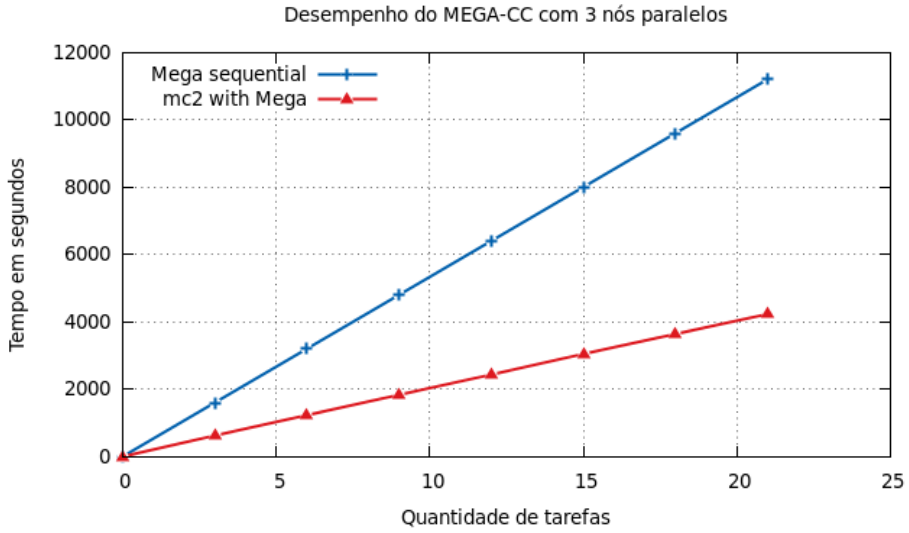

Figura 5.3. Comparação da execução paralela do MEGA-CC sozinho e na plataforma mc^2.

5.3. Estudo de caso em computação intensiva de dados

Foram feitos três estudos de caso com o *framework* Hadoop na plataforma mc^2. Com estes diferentes estudos foi possível avaliar melhor o desempenho em termos da escalabilidade de processamento da plataforma mc^2 no que se refere a implementação de sistemas de computação intensiva de dados. A escolha das aplicações científicas para a implantação no Hadoop e o desenvolvimento do *middleware* para a plataforma mc^2 seguiram vários aspectos relacionados ao estudo de sistemas distribuídos. A aplicação do estudo de caso 5.3.1 foi desenvolvida para que se tomasse conhecimento de como gerenciar aplicações no Hadoop, bem como a primeira integração do *middleware* com o SGA-Hadoop. A aplicação do estudo de caso 5.3.2 foi escolhida devido a sua grande usabilidade na comunidade científica de biologia. A aplicação do estudo de caso 5.3.3 foi escolhida devido a sua necessidade de uso em um projeto no Laboratório de Genética e Populações e Evolução Molecular do Instituto de Biologia da UFBA. Estes estudos de caso são descridos nas Seções a seguir.

5.3.1. Estudo de caso com uma aplicação GREP

O primeiro estudo de caso em computação intensiva de dados realizado no Hadoop através do CSGrid foi de uma aplicação *grep* já conhecida no sistema operacional Linux. A função *grep* no Linux tem o objetivo de procurar *strings* em textos. Em sua saída ela mostra a linha inteira onde achou a *string* passada como parâmetro.

Foram realizados testes com um arquivo de *log* de 3,263Mbytes para a procura de uma *string* qualquer. Apesar de ser um arquivo pequeno, este foi um livro em formato *.txt* com mais de 200 páginas. A máquina em que foi configurado o *NameNode* e o *JobTracker* tem um processador Intel Core2 Duo CPU E7200 2.53GHz. Foram conectadas duas máquinas iMAC para servirem como *DataNode* e *TaskTracker* com um processador Intel Core i5-2400S CPU 2.50GHz de quatro núcleos. O tempo em segundos do processamento está descrito na Tabela 5.7. Com a plataforma mc^2 o tempo de processamento foi superior em 8 segundos, reflexo da transferência de arquivos de entrada e processamentos extras como verificação se o serviço do Hadoop está sendo executado, verificação se o diretório no sistema de arquivos *HDFS* já existe para não sobreescrevê-lo, cópia do arquivo para dentro do sistema de arquivos *HDFS* e cópia do arquivo de resultado para a saída do CSGrid. Mais detalhes da integração da aplicação GREP no Hadoop com o CSGrid está descrito no Apêndice B.3.

Tabela 5.7. Tempo de processamento do Grep *MapReduce*

Aplicação	segundos
MapReduce Grep	19,6
MapReduce Grep com CSGrid	27,0

5.3.2. Estudo de caso com a aplicação CloudBurst

O CloudBurst [Schatz 2009] é um algoritmo de mapeamento paralelo otimizado para ler os dados de sequência do genoma humano e outros genomas de referência, para utilização numa variedade de análises biológicas, incluindo descoberta SNP, genotipagem e genômica pessoal. Ele é modelado para o mapeamento de sequências curtas [URL 2014v] e reporta todos os alinhamentos ou o melhor alinhamento inequívoco para cada leitura com qualquer número de inadequações ou diferenças. Este nível de processamento poderia ser demorado demais, porém o CloudBurst usa a implementação *MapReduce open-source* do Hadoop para paralelizar a execução usando vários nós de computação. Mais detalhes da integração da aplicação CloudBurst no Hadoop com o CSGrid está descrito no Apêndice B.4.

Este estudo de caso foi feito usando as configurações dos mesmos nós do exemplo *grep* da Seção 5.3.1 e com o incremento de mais um computador *worker*. A Tabela 5.8 apresenta o tempo em segundos do processamento do CloudBurst sozinho e na plataforma mc^2, e os respectivos desvios padrão de dez amostras de cada métrica. Ela mostra que a aplicação CloudBurst tem um atraso para processar os dados quando incorporada na plataforma mc^2. Isto acontece pela mesma razão do primeiro estudo de caso, o tempo de transferência de arquivos pela rede. Entretanto, como este programa é uma aplicação científica, ele atende aos objetivos para que foi implantado na plataforma mc^2. Por isso, existe o planejamento de aumentar a carga de dados para este estudo de caso. Foram realizados três testes com 10 amostras cada. No primeiro, apenas com uma máquina servindo como *master* e *worker*. No segundo e no terceiro foram adicionadas máquinas iMAC com papeis de *worker*. Os resultados encontrados estão expressos em segundos.

Depois do cálculo da média dos resultados foi também calculado o desvio padrão, como mostra a Figura 5.4. Com o cálculo do desvio padrão das dez amostras colhidas

Tabela 5.8. Tempo (segundos/desvio) do processamento do CloudBurst

Aplicação	1pc	1pc + 1iMac	1pc + 2iMac	1pc + 3iMac
CloudBurst	101,685/1,35	56,254/1,56	47,724/1,78	42,193/0,97
CloudBurst na mc^2	113,5/1,85	78,1/1,14	58,3/2,41	53,5/1,68

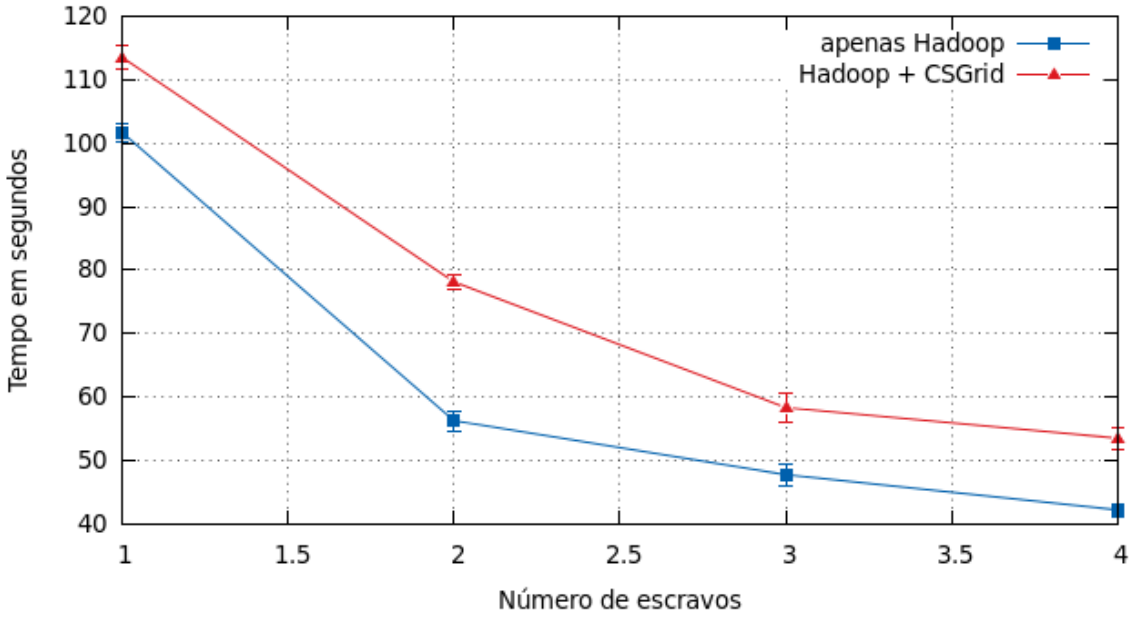

Figura 5.4. Métricas da aplicação CloudBurst no Hadoop na plataforma mc^2

com este experimento é possível verificar que o comportamento da função *MapReduce* no Hadoop e a sua integração com a plataforma mc^2 não varia muito. Existe apenas uma diferença entre 10 e 22 segundos que é devido a transferência de arquivos entre os *frameworks*. Também é possível verificar que com o aumento dos nós *workers* no módulo SGA-Hadoop o desempenho do processamento de uma única tarefa aumenta consideravelmente.

5.3.3. Estudo de caso com a aplicação BLAST

BLAST é um acrônimo para ***B**asic **L**ocal **A**lignment **S**earch **T**ool*. Ele consiste em um algoritmo para comparar informações de sequências biológicas primárias, tais como sequências de aminoácidos de diferentes proteínas ou nucleotídeos de sequências de DNA. A sua implementação no SGA Hadoop é muito similar a do CloudBurst, descrito na Seção 5.3.2, exceto pelos comandos de gerenciamento do *HDFS* que foram específicos para o BLAST. Mais detalhes da sua implementação podem ser verificados no Apêndice B.5. Com esta implementação foi possível medir uma quantidade maior de carga nos computadores com Hadoop, pois os arquivos processados eram de tamanhos maiores. Assim como no estudo de caso com o CloudBurst, foram feitas dez amostras e calculado o desvio padrão para elas, com as configurações de rede iguais ao exemplo anterior. Os resultados podem ser verificados na Tabela 5.9.

A Figura 5.5 mostra os resultados das métricas do BLAST no Hadoop sozinho e seu módulo na plataforma mc^2. Neste acoplamento, chamado de SGA Hadoop, a tarefa

Tabela 5.9. Tempo (segundos/desvio) de processamento do BLAST

Aplicação	1pc	1pc + 1iMac	1pc + 2iMac	1pc + 3iMac
Blast	911,34/4,44	460,43/3,88	289,58/16,0	235,79/7,58
Blast na mc^2	924,6/15,03	469,3/5,0	293,4/6,9	240,96/0,9

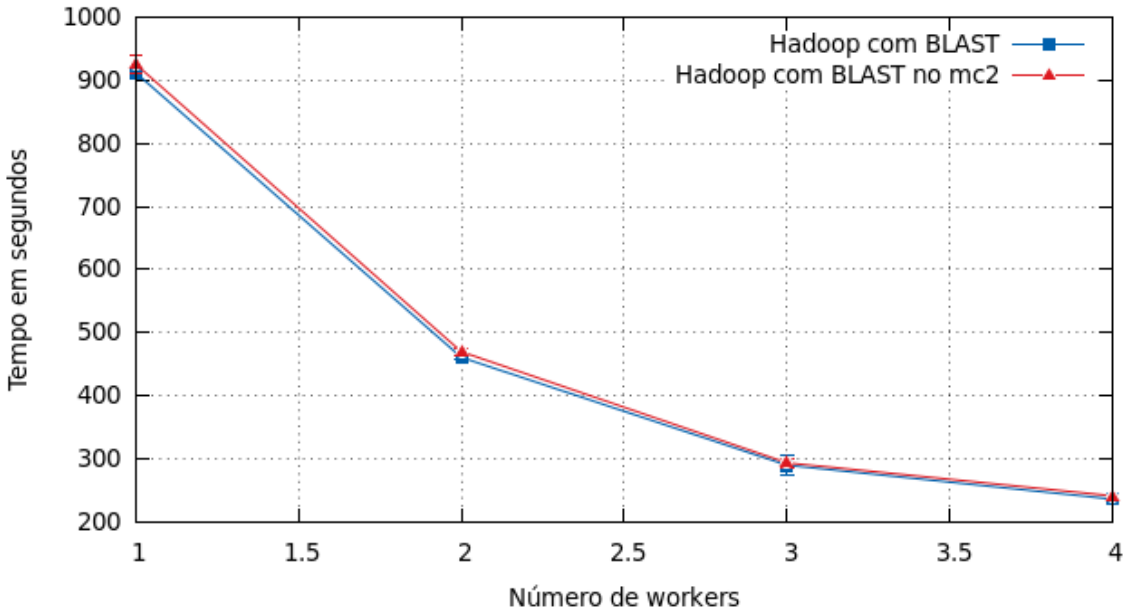

Figura 5.5. Métricas da aplicação BLAST no Hadoop na plataforma mc^2

iniciada pelo cientista percorre o computador do cliente. Iniciada pelo Galaxy, ela vai até o CSGrid, que é o gerenciador da grade computacional do mc^2 e chega até o nó mestre do Hadoop. A partir deste ponto, o controle é feito pelo próprio framework Hadoop. Pode-se observar nas Tabelas 5.8 e 5.9 que o desvio padrão desta métrica com o BLAST é muito menor em relação ao desvio da métrica com o CloudBurst, devido a maior carga de dados. Portanto, é possível concluir que o aumento de carga neste módulo da plataforma mc^2, também melhora as condições de processamento das tarefas, assim como no módulo do BOINC. Além disso, é possível observar que o desvio padrão diminui com o aumento de nós neste módulo. Considerando que a leitura do CSGrid no Hadoop, para descobrir se a tarefa já está concluída, é a cada 10 segundos, este tempo torna-se cada vez mais irrelevante quando se aumenta a quantidade de nós neste módulo da plataforma mc^2.

5.4. Resumo

Este capítulo abordou quatro estudos de casos na plataforma mc^2 utilizando o SGA-BOINC e três utilizando o SGA-Hadoop. Estas implementações oferecem a capacidade de criar várias tarefas ou subtarefas de processamento paralelo e cujos resultados finais são agrupados pelo CSGrid. O seu principal objetivo é tornar o processamento de aplicações científicas escalável. Devido a implantação de quatro tipos diferentes de aplicações científicas no BOINC e três no Hadoop e sua integração com o CSGrid, foi possível provar a flexibilidade, escalabilidade e abstração geral deste módulo da plataforma mc^2.

Por exemplo, com o Hex foi possível provar o aumento do desempenho do proces-

samento com o aumento do número de tarefas em cada nó. Pela sobreposição de tarefas nos processadores e a transferência sobreposta de arquivos pela rede, o desempenho de um conjunto de tarefas sendo processado neste módulo é maior que se fossem processadas sozinhas. Com a aplicação Bubblesort escrita em C++, foi possível perceber o aumento do desempenho com o incremento de carga de tarefas nos processadores. Quando as tarefas eram maiores, a aceleração delas aumentou em relação as tarefas menores. Com a aplicação MEGA-CC foi possível usar as duas metodologias de medição. Tanto o número de tarefas foi incrementado, como o número de nós. Além destes resultados já obtidos com as aplicações anteriores, foi possível colaborar no uso cotidiano do programa no Laboratório de Genética e Populações e Evolução Molecular do Instituto de Biologia da UFBA. O estudo de caso com aplicação GREP foi feito para aprender o paradigma de modelagem *MapReduce* e realizar a primeira integração do *middleware* SGA-Hadoop com a plataforma mc^2. Com o estudo de caso da aplicação CloudBurst foi possível verificar a facilidade da escalabilidade dos nós neste módulo da plataforma mc^2 e o respectivo ganho de desempenho. Com a aplicação BLAST foi possível, além das observações anteriores, ganhar desempenho com uma quantidade de carga relativamente grande para os nós deste módulo. Verificou-se também que o desvio padrão permaneceu baixo em relação ao tamanho da carga.

Portanto, as métricas realizadas puderam demonstrar que com o aumento de tarefas a serem processadas no BOINC, de modo geral, o sistema aumenta a velocidade de processamento, atingindo o objetivo caracterizado como elasticidade oferecida pelos modelos de computação distribuída. A facilidade de escalabilidade oferecida pelo *framework* BOINC garantiu o aumento do poder de processamento das aplicações. Também com o aumento da carga de processamento para o módulo do Hadoop, foi possível verificar que o desvio padrão das métricas diminuiu e a distância entre uma tarefa sendo processada apenas no Hadoop é pequena se ela for processada no módulo da plataforma mc^2.

As maiores dificuldades enfrentadas nesta integração foram com as aplicações que continham vários arquivos, como o Hex e o MEGA-CC. Estas aplicações necessitaram do uso de uma biblioteca *wrapper* de extensão do BOINC, para que as tarefas pudessem iniciar e terminar com sucesso. A aplicação Bubblesort em C++ foi a mais fácil de implantar no BOINC, pois era composta de apenas um arquivo. Com a ajuda das bibliotecas do BOINC em C++, foi possível empacotar todos os arquivos em um úico executável e implantá-lo no BOINC com sucesso. Com a aplicação de *benchmark* NAS foi enfrentado certa dificuldade pois ela é escrita em Fortran, porém foi criado um programa em C++ que pudesse executá-lo.

Os maiores desafios enfrentados na integração do módulo SGA-Hadoop na plataforma mc^2 começaram com o desenvolvimento de uma aplicação que seguia o paradigma *MapReduce*, no caso a aplicação GREP. O paradigma de programação *MapReduce* divide uma tarefa em diversos pedaços pequenos, configurado pelo programador, e processa cada subtarefa em um computador ou espaço do processador diferente. No final do processamento esses resultados das subtarefas são agrupados novamente para formar o resultado final. Compreendido este paradigma de programação, ficou mais claro o entendimento dos tipos de processamento que seriam necessários para as outras aplicações científicas que estavam por vir. Com a aplicação CloudBurst o principal desafio foi saber como executá-la no Hadoop, pois ela apresentava alguns erros mesmo com a ajuda de tutoriais

na Internet. Outro desafio foi estabelecer as regras necessárias para o gerenciamento de arquivos no *HDFS* através do SGA-Hadoop. Com a aplicação BLAST o maior desafio foi em encontrar alguma aplicação relevante para o uso cotidiano no Laboratório de Genética e Populações e Evolução Molecular do Instituto de Biologia da UFBA, assim como a escolha das melhores regras de gerenciamento para os arquivos no *HDFS* e a necessidade de processamento de cargas de arquivos maiores que a aplicação anterior.

O módulo de integração com o BOINC pode ser validado pela comunidade científica devido a publicação *"Providing volunteer computing at the infrastructure level to support e-science applications"* [Gutierrez et al. 2013] e a aprovação de um artigo *"Volunteer computing for bioinformatics data analysis with MEGA Computation Core"* no BICoB 2014 (*6th International Conference on Bioinformatics and Computational Biology*) [URL 2014d]. O módulo de integração com o Hadoop pode ser validado pela comunidade científica devido a publicação do artigo *"Provisão de computação intensiva de dados para suporte a aplicações científicas"* [Felipe Gutierrez 2013] no ERAD-NE 2013, a aprovação do artigo *"Support for e-science applications through volunteer and data-intensive computing frameworks"* [Gutierrez and Barreto 2013] no LatinCloud 2013 e a expectativa de publicação no AlCoB 2014 [URL 2014c].

Capítulo 6

Conclusões

6.1. Considerações finais

Neste trabalho de dissertação foi feita uma pesquisa científica na área de Ciência da Computação, sobre o tema de Sistemas Distribuídos. Todo o período de desenvolvimento deste trabalho foi motivado por indagações de como se faz ciência, especificamente para a área relatada. O trabalho realizado pode ser dividido em quatro grandes etapas, já anteriormente discutidas em outras fontes [URL 2014m]. Estas etapas são muito dinâmicas e envolvem (i) a exploração e o descobrimento, (ii) o teste das ideias, (iii) a análise da comunidade e o retorno dela e (iv) os benefícios e resultados.

As principais contribuições deste trabalho foram a facilidade fornecida para os cientistas de diversas áreas que têm programas que necessitam de alto processamento, o que inclui várias tarefas ou grande quantidade de dados para processar. Também o trabalho de engenharia de *software* que teve o objetivo de integrar os frameworks BOINC e Hadoop com o CSGrid e as diversas aplicações científicas. Através da plataforma mc^2 os cientistas podem utilizar essas aplicações remotamente, assim como tornar os módulos mais poderosos em relação ao desempenho de processamento. Todas essas características preservam a confiabilidade e segurança dos dados processados.

As principais limitações durante a realização deste trabalho foram os tipos de programas *e-Science* encontrados para a implantação no modelo de programação *MapReduce*. Por causa da novidade deste modelo de programação, foi necessário encontrar aplicações *e-Science* que já viessem ou pudessem ser escritos de acordo com este novo modelo. Outro software open-source que se vem estudando é o Contrail [URL 2014f], desenvolvido para solucionar desafios chaves associados com a montagem de genoma em larga escala. Ele já foi implantado na plataforma mc^2, porém ele está sofrendo alterações pela equipe de desenvolvimento oficial para a correção de defeitos. Por isso foi necessário esperar para que se pudesse tirar análises reais deste *software*.

6.2. Publicações

O estudo da ciência não é apenas a coleta de fatos, mas o entendimento do que os testes implicam para uma comunidade científica. A ciência explora por diferentes caminhos, porém, todos eles com espectativas geradas por ideias e observações. A aceitação dessas ideias devem ser submetidas a testes rigorosos e revisados por toda a comuni-

dade científica. A comunidade científica tem o papel de garantir que a ciência se move na direção correta, com alta acuracidade e entendimento. Esse grupo de pessoas facilitam a diversificação da comunidade, abrangendo maiores perspectivas de ideias científicas. Com este objetivo foi publicado um artigo e submetidos, com aprovação, mais dois artigos relacionados a esta dissertação. O primeiro artigo publicado foi no âmbito de Conferências Nacionais da SBC (Sociedade Brasileira de Computação) e envolveu a integração com a plataforma mc^2 através do módulo de computação voluntária, discutida no Capítulo 4 e implementada no Apêndice A. Este artigo foi publicado no XXXIII Congresso da Sociedade Brasileira de Computação de 2013 [Gutierrez et al. 2013]. A mesma integração da plataforma mc^2 com computação voluntária, porém com um software de uso cotidiano no Laboratório de Genética e Populações e Evolução Molecular do Instituto de Biologia da UFBA, obteve um artigo aprovado no BICoB 2014 (*6th International Conference on Bioinformatics and Computational Biology*) [URL 2014d]. A segunda parte da integração com a plataforma mc^2 que aborda a integração com computação intensiva de dados discutida no Capítulo 4 e implementada no Apêndice B.1, foi aprovada no IEEE LatinCloud 2013 [Gutierrez and Barreto 2013] e está se esperando o resultado do AlCoB 2014 (*1st International Conference on Algorithms for Computational Biology*) [URL 2014c].

6.3. Trabalhos futuros

Através da rede SINAPAD [URL 2014x], a platafroma mc^2 pode oferecer suporte a mais aplicações científicas para atender as necessidades de cientistas, independente de suas regiões geográficas. Várias aplicações de computação intensiva de dados, que podem ser usadas com o Hadoop, são encontradas facilmente na Internet. Aplicações não só de Biologia, como também de Matemática, Física, Geologia, etc. As aplicações que não seguem o modelo *MapReduce* podem ser incorporadas no módulo de computação voluntária, através do Boinc, e beneficiar mais cientistas em suas pesquisas.

Uma das aplicações de grande relevância no Laboratório de Genética e Populações e Evolução Molecular do Instituto de Biologia da UFBA é o Contrail, porém foi verificado que este software se encontra em desenvolvimento de uma nova versão que estão corrigindo vários *bugs*. Espera-se que quando a nova versão for liberada ele possa ser implantado na plataforma mc^2 e melhorar o desempenho de análise dos dados no Laboratório de Genética e Populações e Evolução Molecular do Instituto de Biologia da UFBA. Outras aplicações foram avaliadas como sendo muito relevantes para a implantação na plataforma mc^2, tais como o CrossBow [URL 2014g], o GATK [URL 2014i] e o Trinity [URL 2014z]. Para estas aplicações será necessário o estudo, assim como foi feito para as outras, para verificar em qual módulo da plataforma mc^2 elas se adequam melhor.

Referências

[Altintas et al. 2004] Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., and Mock, S. (2004). Kepler: an extensible system for design and execution of scientific workflows. *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*, pages 423–424.

[Amazon 2010] Amazon, A. (2010). Amazon web services. *Available in: http://aws. amazon. com/es/ec2/(November 2012)*.

[Anagnostakis et al. 2006] Anagnostakis, K. G., Charmatzis, F., Ioannidis, S., and Zghaibeh, M. (2006). On the impact of p2p incentive mechanisms on user behavior. In *IN NETECON+IBC*.

[Anderson 2004] Anderson, D. P. (2004). Boinc: A system for public-resource computing and storage. In Buyya, R., editor, *5th International Workshop on Grid Computing (GRID 2004), 8 November 2004, Pittsburgh, PA, USA, Proceedings*, pages 4–10. IEEE Computer Society.

[Anderson and Fedak 2006] Anderson, D. P. and Fedak, G. (2006). The computational and storage potential of volunteer computing. In *CCGRID*, pages 73–80.

[Armbrust et al. 2010] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2010). A view of cloud computing. *Commun. ACM*, 53(4):50–58.

[Avizienis et al. 2004] Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33.

[Baldassari et al. 2006] Baldassari, J., Finkel, D., and Toth, D. (2006). Slinc: A framework for volunteer computing.

[Bastos et al. 2013] Bastos, B. F., Moreira, V. M., and Gomes, A. T. A. (2013). Rapid prototyping of science gateways in the brazilian national hpc network. In *IWSG*.

[Beguelin et al. 1991] Beguelin, A., Dongarra, J., Geist, A., Manchek, R. (Oak Ridge National Lab., T. U. S., Sunderam, V. (Emory Univ., A. G. U. S. D. o. M., and Science), C. (1991). *A users guide to PVM (Parallel Virtual Machine)*.

[Bernstein 1996] Bernstein, P. A. (1996). Middleware: A model for distributed system services. *Commun. ACM*, 39(2):86–98.

[Bondi 2000] Bondi, A. B. (2000). Characteristics of scalability and their impact on performance. In *Proceedings of the 2nd international workshop on Software and performance*, WOSP '00, pages 195–203, New York, NY, USA. ACM.

[Calder et al. 2011] Calder, B., Wang, J., Ogus, A., Nilakantan, N., Skjolsvold, A., McKelvie, S., Xu, Y., Srivastav, S., Wu, J., Simitci, H., Haridas, J., Uddaraju, C., Khatri, H., Edwards, A., Bedekar, V., Mainali, S., Abbasi, R., Agarwal, A., Haq, M. F. u., Haq, M. I. u., Bhardwaj, D., Dayanand, S., Adusumilli, A., McNett, M., Sankaran, S., Manivannan, K., and Rigas, L. (2011). Windows azure storage: A highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 143–157, New York, NY, USA. ACM.

[Ceruzzi 1998] Ceruzzi, P. E. (1998). *A History of Modern Computing*. MIT Press, Cambridge, MA, USA.

[Chien et al. 2003] Chien, A. A., Calder, B., Elbert, S., and Bhatia, K. (2003). Entropia: architecture and performance of an enterprise desktop grid system. *J. Parallel Distrib. Comput.*, 63(5):597–610.

[Churches D. 2005] Churches D., Gombas G., H. A. M. J. R. C. S. M. T. I. W. I. (2005). Programming scientific and distributed workflow with triana services. concurrency and computation: Practice and experience. Special Issue on Scientific Workflows.

[Cirne et al. 2003] Cirne, W., Paranhos, D., Costa, L., Santos-neto, E., Brasileiro, F., Sauvé, J., Grande, C., Alves, F., Silva, B., Santos, C., Barros, C. O., Nacional, L., Científica, C., Silveira, C., and Packard, H. (2003). Running bag-of-tasks applications on computational grids: The mygrid approach. In *In ICPP*, page 407.

[Computing 2014] Computing, G. (2014). Grid computing info centre (grid infoware). January 10, 2014.

[Costa et al. 2011] Costa, F., Silva, L., and Dahlin, M. (2011). Volunteer cloud computing: MapReduce over the Internet. In *Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 1855–1862.

[Deelman et al. 2005] Deelman, E., Singh, G., Su, M. H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G. B., Good, J., Laity, A. C., Jacob, J. C., and Katz, D. S. (2005). Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237.

[Dongarra et al. 1993] Dongarra, J.J. [Tennessee Univ., K. T. U. S. D. o. C. S. R. N. L. T. U. S., Hempel, R. [Gesellschaft fuer Mathematik und Datenverarbeitung mbh Bonn, S. A. G., Hey, A. S. U. U. K. D. o. E., Science], C., and Walker, D.W. [Oak Ridge National Lab., T. U. S. (1993). *A proposal for a user-level, message passing interface in a distributed memory environment*.

[Drago et al. 2012] Drago, I., Mellia, M., M. Munafo, M., Sperotto, A., Sadre, R., and Pras, A. (2012). Inside dropbox: Understanding personal cloud storage services. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, IMC '12, pages 481–494, New York, NY, USA. ACM.

[Elwaer et al. 2011] Elwaer, A., Harrison, A., Kelley, I., and Taylor, I. (2011). Attic: A case study for distributing data in boinc projects. In *IPDPS Workshops*, pages 1863–1870. IEEE.

[Farkas et al. 2010] Farkas, Z., Kacsuk, P., Balaton, Z., and Gombás, G. (2010). Interoperability of boinc and egee. *Future Gener. Comput. Syst.*, 26(8):1092–1103.

[Fedak et al. 2001] Fedak, G., Germain, C., Neri, V., and Cappello, F. (2001). XtremWeb: A generic global computing system. Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID 01).

[Felipe Gutierrez 2013] Felipe Gutierrez, M. B. (2013). Provisão de computação intensiva de dados para suporte a aplicações científicas. In *ERAD-NE 2013*, Salvador, Brazil.

[Foster 2002] Foster, I. (2002). What is the grid. a three point checklist. Argonne National Laboratory and University of Chicago.

[Foster 2005] Foster, I. (2005). Globus toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing*, number 3779 in LNCS, pages 2–13. Springer-Verlag.

[Foster and Kesselman 1999] Foster, I. and Kesselman, C. (1999). *The grid: blueprint for a new computing infrastructure*. Advanced computing. Computer systems design. Morgan Kaufmann Publishers.

[Foster et al. 2001] Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222.

[Gannon 2007] Gannon, D. (2007). Programming e-science gateways. In Danelutto, M., Fragopoulou, P., and Getov, V., editors, *CoreGRID Workshop - Making Grids Work*, pages 191–200. Springer.

[Goecks et al. 2010] Goecks, J., Nekrutenko, A., and Taylor, J. (2010). Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol, 11(8):r86 edition.

[Gonzalez et al. 2008] Gonzalez, D. L., Fernandez de Vega, F., Trujillo, L., Olague, G., Cardenas, M., Araujo, L., Castillo, P., Sharman, K., and Silva, A. (2008). Interpreted applications within BOINC infrastructure. In *IBERGRID 2nd Iberian Grid Infrastructure Conference Proceedings*, pages 261–272. netbiblo.com.

[Gutierrez and Barreto 2013] Gutierrez, F. and Barreto, M. (2013). Support for e-science applications through volunteer and data-intensive computing frameworks. In *LATINCLOUD 2013 ()*, Maceió, Brazil.

[Gutierrez et al. 2013] Gutierrez, F., Barreto, M., and Gomes, A. T. A. (2013). Providing volunteer computing at the infrastructure level to support e-science applications. In *BreSci 2013 (VII Brazilian e-Science workshop)*.

[Hey et al. 2009] Hey, T., Tansley, S., and Tolle, K., editors (2009). *The Fourth Paradigm: Data-Intensive Scientific Discovery*.

[Hilbert et al. 2010] Hilbert, M., López, P., and Vásquez, C. (2010). Information societies or "ict equipment societies?"measuring the digital information-processing capacity of a society in bits and bytes. *Inf. Soc.*, 26(3):157–178.

[Howes and Smith 1997] Howes, T. and Smith, M. (1997). *LDAP: programming directory-enabled applications with lightweight directory access protocol*. Macmillan Publishing Co., Inc., Indianapolis, IN, USA.

[Hwang et al. 2012] Hwang, K., Fox, G. C., and Dongarra, J. J. (2012). *Distributed and cloud computing : from parallel processing to the Internet of things*. Morgan Kaufmann, Watham (Mass).

[Ian Foster 2008] Ian Foster, Yong Zhao, I. R. S. L. (2008). Cloud computing and grid computing 360-degree compared. pages 1–10. Grid Computing Environments Workshop.

[J. B. M. Litzkow 1997] J. B. M. Litzkow, T. Tannenbaum, M. L. (1997). Checkpoint and migration of unix processes in the condor distributed processing system. University of Wisconsin.

[Jeffrey Dean 2008] Jeffrey Dean, S. G. (2008). Mapreduce simplified data processing on large clusters. volume 51, pages 107–113. Magazine Communications of the ACM.

[Karl Aberer 2005] Karl Aberer, Luc Onana Alima, A. G. S. G. S. H. M. H. (2005). The essence of P2P: A reference architecture for overlay networks. Ecole Polytechnique Féderale de Lausanne (EPFL), Université de Mons-Hainaut (UMH), Swedish Institute of Computer Science (KTH).

[Kleinrock 2014] Kleinrock, L. (2014). Leonard kleinrock - distinguished professor - ucla - computer science department. January 10, 2014.

[Kumar et al. 2012] Kumar, S., Stecher, G., Peterson, D., and Tamura, K. (2012). MEGA-CC: computing core of molecular evolutionary genetics analysis program for automated and iterative data analysis. *Bioinformatics (Oxford, England)*, 28(20):2685–2686.

[Lima et al. 2005] Lima, M. J. D., Melcop, T., Cerqueira, R., Cassino, C., Silvestre, B., Nery, M., and Ururahy, C. (2005). CSGrid: um sistema para integração de aplicações em grades computacionais. Anais do 23o. Simpósio Brasileiro de Redes de Computadores - SBC.

[Lima et al. 2006] Lima, M. J. D., Ururahy, C., Moura, A., Melcop, T., Cassino, C., Nery, M., Silvestre, B., Reis, V., and Cerqueira, R. (2006). CSBase: A framework for building customized grid environments. Third International Workshop on Emerging Technologies for Next-generation GRID.

[Lin et al. 2010] Lin, H., Ma, X., Archuleta, J., Feng, W.-c., Gardner, M., and Zhang, Z. (2010). Moon: Mapreduce on opportunistic environments. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 95–106, New York, NY, USA. ACM.

[Lin and Dyer 2010] Lin, J. and Dyer, C. (2010). *Data-Intensive Text Processing with MapReduce*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

[Matsunaga et al. 2008] Matsunaga, A. M., Tsugawa, M. O., and Fortes, J. A. B. (2008). Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications. In *eScience*, pages 222–229.

[Michael Armbrust 2009] Michael Armbrust, Armando Fox, R. G. A. D. J. R. H. K. A. K. G. L. D. A. P. A. R. I. S. M. Z. (2009). Above the Clouds: A berkeley view of cloud computing. Electrical Engineering and Computer Sciences University of California at Berkeley.

[Milojicic et al. 2003] Milojicic, D. S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., and Xu, Z. (2003). Peer-to-peer computing. Technical report.

[Milojičić et al. 2011] Milojičić, D., Llorente, I. M., and Montero, R. S. (2011). Opennebula: A cloud management tool. *IEEE Internet Computing*, 15(2):11 – 14.

[Mone 2013] Mone, G. (2013). Beyond hadoop. *Commun. ACM*, 56(1):22–24.

[Nadiminti and Buyya 2005] Nadiminti, K. and Buyya, R. (2005). Enterprise grid computing: State-of-the-art. Grid Computing and Distributed Systems laboratory - The University of Melbourne.

[Nazareno Andrade 2003] Nazareno Andrade, Walfredo Cirne, F. B. P. R. (2003). OurGrid: An approach to easily assemble grids with equitable resource sharing. pages 61–86. Springer Berlin Heidelberg.

[Nei and Kumar 2000] Nei, M. and Kumar, S. (2000). *Molecular Evolution and Phylogenetics*. Oxford University Press, USA, 1 edition.

[Niehörster et al. 2009] Niehörster, O., Birkenheuer, G., Brinkmann, A., Blunk, D., Elsässer, B., Herres-Pawlis, S., Krüger, J., Niehörster, J., Packschies, L., and Fels, G. (2009). Providing scientific software as a service in consideration of service level agreements. In *Proceedings of the Cracow Grid Workshop (CGW)*, pages 55–63.

[NIST 2014] NIST (2014). National institute of standards and technology. January 10, 2014.

[Nurmi et al. 2009] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. (2009). The eucalyptus open-source cloud-computing system. In Cappello, F., Wang, C.-L., and Buyya, R., editors, *CCGRID*, pages 124–131. IEEE Computer Society.

[Oinn T. 2004] Oinn T., Addis M., F. J. M. D. S. M. G. M. C. T. G. K. P. M. R. W. A. L. P. (2004). Taverna: A tool for the composition and enactment of bioinformatics workflows. pages 3045–3054. Bioinformatics Journal.

[Ritchie 2003] Ritchie, D. (2003). Evaluation of protein docking predictions using hex 3.1 in capri rounds 1 and 2. volume 52, pages 98–106.

[Sanderson 2009] Sanderson, D. (2009). *Programming Google App Engine: Build and Run Scalable Web Apps on Google's Infrastructure*. O'Reilly Media, Inc., 1st edition.

[Schatz 2009] Schatz, M. C. (2009). Cloudburst: highly sensitive read mapping with mapreduce. *Bioinformatics*, 25(11):1363–1369.

[Sefraoui et al. 2012] Sefraoui, O., Aissaoui, M., and Eleuldj, M. (2012). Article: Openstack: Toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3):38–42. Published by Foundation of Computer Science, New York, USA.

[Tamura et al. 2011] Tamura, K., Peterson, D., Peterson, N., Stecher, G., Nei, M., and Kumar, S. (2011). MEGA5: molecular evolutionary genetics analysis using maximum likelihood, evolutionary distance, and maximum parsimony methods. *Molecular biology and evolution*, 28(10):2731–2739.

[Tannenbaum et al. 2001] Tannenbaum, T., Wright, D., Miller, K., and Livny, M. (2001). Condor – a distributed job scheduler. In Sterling, T., editor, *Beowulf Cluster Computing with Linux*. MIT Press.

[URL 2014a] URL (2014a). http://boincstats.com/en/stats/-1/project/detail/overview. January 10, 2014.

[URL 2014b] URL (2014b). http://www.lua.org/. January 10, 2014.

[URL 2014c] URL (2014c). 1st international conference on algorithms for computational biology, alcob 2014. http://grammars.grlmc.com/alcob2014/. January 10, 2014.

[URL 2014d] URL (2014d). 6th international conference on bioinformatics and computational biology - bicob 2014. http://www.cs.umb.edu/bicob/accepted_papers.html. January 10, 2014.

[URL 2014e] URL (2014e). Apache software fundation. http://apache.org/. January 10, 2014.

[URL 2014f] URL (2014f). Contrail. http://sourceforge.net/apps/mediawiki/contrail-bio/index.php?title=Contrail. January 10, 2014.

[URL 2014g] URL (2014g). Crossbow - genotyping from short reads using cloud computing. . January 10, 2014.

[URL 2014h] URL (2014h). Csfs. https://jira.tecgraf.puc-rio.br/confluence/pages/viewpage.action?pageId=30574804. January 10, 2014.

[URL 2014i] URL (2014i). Gatk - genome analysis toolkit. http://www.broadinstitute.org/gatk/. January 10, 2014.

[URL 2014j] URL (2014j). Google drive. http://drive.google.com/. January 10, 2014.

[URL 2014k] URL (2014k). Grid gain in-memory computing platform. http://www.gridgain.com/. January 10, 2014.

[URL 2014l] URL (2014l). Hex. http://hex.loria.fr/dist68/. January 10, 2014.

[URL 2014m] URL (2014m). How science works. http://undsci.berkeley.edu/article/0_0_0/howscienceworks_01. January 10, 2014.

[URL 2014n] URL (2014n). Htcondor - high throughput computing. http://research.cs.wisc.edu/htcondor/dagman/dagman.html. January 10, 2014.

[URL 2014o] URL (2014o). An information portal to biological macromolecular structures. http://www.rcsb.org/pdb/. January 10, 2014.

[URL 2014p] URL (2014p). Laboratório de sistemas distribídos da universidade federal bahia. http://www.lasid.ufba.br/. January 10, 2014.

[URL 2014q] URL (2014q). Laboratório de sistemas distribídos da universidade federal de campina grande. http://www.lsd.ufcg.edu.br/. January 10, 2014.

[URL 2014r] URL (2014r). The leading open source in-memory data grid. http://www.hazelcast.com/. January 10, 2014.

[URL 2014s] URL (2014s). Ngrid - open source grid computing. http://ngrid.sourceforge.net/. January 10, 2014.

[URL 2014t] URL (2014t). Npb- nas parallel benchmarks. https://www.nas.nasa.gov/cgi-bin/software/start. January 10, 2014.

[URL 2014u] URL (2014u). Open cloud manifesto. http://opencloudmanifesto.org/opencloudmanifesto1.htm. January 10, 2014.

[URL 2014v] URL (2014v). The rmap software for short-read mapping. http://rulai.cshl.edu/rmap/. January 10, 2014.

[URL 2014w] URL (2014w). Science clouds. http://scienceclouds.org/. January 10, 2014.

[URL 2014x] URL (2014x). Sistema nacional de processamento de alto desempenho. https://www.lncc.br/sinapad/. January 10, 2014.

[URL 2014y] URL (2014y). Synamic code execution and resource management. http://www.dacframe.org/. January 10, 2014.

[URL 2014z] URL (2014z). Trinity rna-seq assembly. http://sourceforge.net/projects/trinityrnaseq/files/. January 10, 2014.

[Wang et al. 2009] Wang, J., Crawl, D., and Altintas, I. (2009). Kepler + hadoop: a general architecture facilitating data-intensive applications in scientific workflow systems. In *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, WORKS '09, pages 12:1–12:8, New York, NY, USA. ACM.

[White 2009] White, T. (2009). Hadoop the definitive guide. In *Hadoop The Definitive Guide*.

[Wu et al. 2007] Wu, S., Tao, Y., and Xu, Z. (2007). Pbjim: Plugin-based job interoperation mechanism for heterogeneous grid systems. In *GCC*, pages 830–840. IEEE Computer Society.

[Zhao and Foster 2008] Zhao, Y. and Foster, I. (2008). Scientific workflow systems for 21st century, new bottle or new wine. In *IEEE Workshop on Scientific Workflows*.

Apêndice A

Implementação com computação voluntária

A.1. Introdução

Este primeiro apêndice é dedicado a demosntrar como foi feita a implementação do módulo de nuvem que utiliza o *framework* de computação voluntária BOINC. O apêndice A.2 detalha como foi feita a integração do BOINC com o CSGrid, utilizando um SGA. Os apêndices seguintes A.3, A.4 e A.5 detalham a implementação de diferentes estudos de caso com este modelo de programação.

A.2. Implementação do SGA para o BOINC

A integração do BOINC com o CSGrid é feita através de um SGA (Servidor de Gerência de Algoritmos). Este SGA é configurado para que o CSGrid tenha total controle sobre o *BOINC server*. Sendo assim, o *BOINC server* é uma única vez configurado. Sua configuração consiste na implantação de uma aplicação científica, a criação e a verificação de tarefas.

O principal componente desta integração é a configuração de um arquivo na máquina do SGA para que esta seja reconhecida como um nó do CSGrid. Este arquivo é o *sgad-cnf.lua* e sua principal parte é apresentada no script 1. Os dados de entrada para processamento das tarefas no BOINC estarão disponíveis no CSGrid. Este *framework* é que será responsável pela transferência dos dados para o computador onde está instalado o SGA. Essa transferência pode ser feita através da configuração de um diretório NFS (*Network File System*), no caso de uma rede local, ou através de CSFS [URL 2014h] no caso em que os computadores estão em redes diferentes. O atributo *name* é o *localhost* da máquina do SGA. O atributo *platform_id* indica a plataforma do sistema operacional da máquina, que neste caso é um sistema Linux (Ubuntu 12.04 LTS) que usa um processador de 64Mb. Logo depois é possível verificar a quantidade de números de processadores ou núcleos a serem utilizados pelo SGA. O último parâmetro é composto pela configuração do CSFS. Nele é necessário especificar o IP da máquina do SGA, o diretório de compartilhamento de arquivos, o usuário que deve ter acesso via protocolo ssh sem a necessidade de senha e a porta para comunicação, que neste caso é a de número 10000. É digno de nota que os IP's das máquinas utilizadas para esta configuração devem ser fixos e estar

em uma rede privada. Porém, as máquinas com o *BOINC client* podem estar espalhadas pela Internet e sem um IP fixo.

Script 1 Script de configuração do SGA para o Boinc

```
Node{
  name = "sga-ufba-cloud7",
  platform_id = "Linux26g4_64",
  num_processors = 2,
  memory_ram_info_mb = 2048,
  memory_swap_info_mb = 5376,
  csfs = {
    launch_daemon = YES,
    properties = { HOST="192.168.188.7",
    ROOT_DIR="/home/cloud7/sga_v1.6_u020/sgad/csfs/root/",
    USER = "cloud7", PORT = 10000 }
  }
}
```

A.3. Integração do programa HEX na plataforma mc^2

A aplicação Hex utiliza o arquivo *hex.bat* para ler quatro outros arquivos de entrada. Este arquivo é apresentado no script 2 e sua função é a de iniciar a aplicação HEX.

Script 2 Script hex.bat para iniciar o Hex

```
#!/bin/bash
HEX_ROOT=.
HEX_COLOURS=./data
HEX_PDB=.
HEX_MACROS=.
HEX_CACHE=./data

./hex6i.x64 -kill -nogui -ncpu 1 -ngpu 1 -e dock.mac -l
job.log
```

O arquivo *hex61.x64* é o coração do programa. A Figura A.1 apresenta mais dois arquivos de entrada importantes para o processamento da aplicação Hex. Estes serão os arquivos a serem processados pela aplicação Hex. O Hex tem três arquivos de saída e um arquivo de *log*. Eles são muitos similares com o último arquivo de entrada e apresentam os valores processados dos últimos dois arquivos.

Entretanto, para integrar a aplicação Hex com o *BOINC server* é necessário utilizar uma técnica que faça com que a tarefa do BOINC inicie e pare a aplicação desejada. O processamento das tarefas do BOINC é iniciado pela função *boinc_init* e finalizado pela função *boinc_finish*, mas estas funções não estão associadas à aplicação Hex. O BOINC faz isso através de um componente *wrapper*. O componente *wrapper* facilita a execução de aplicações que não estejam em conformidade com a API BOINC, inicialmente usado para compatibilidade com aplicações complexas e de difícil modificação

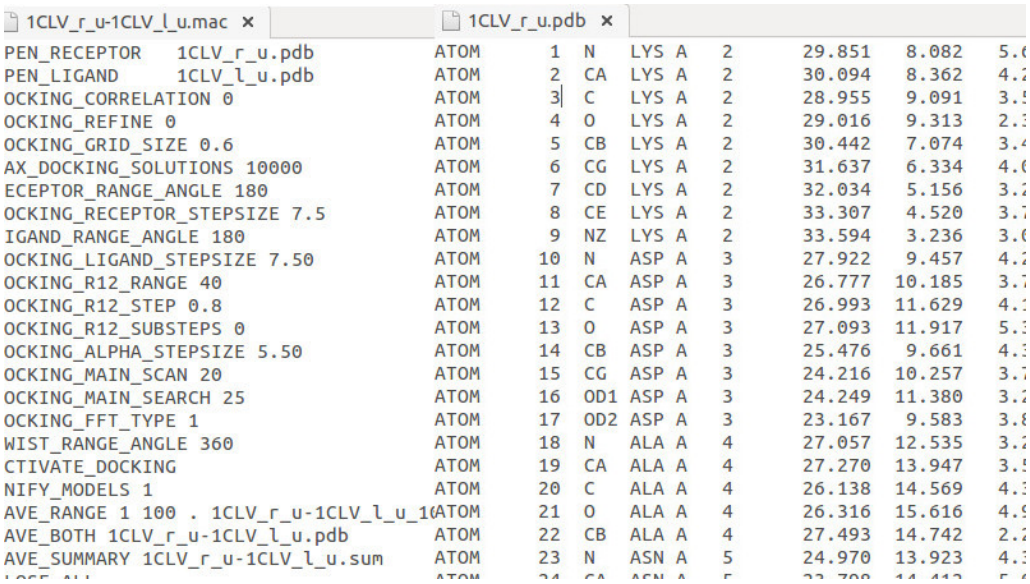

Figura A.1. Arquivos de entrada do Hex

pelo motivo de usarem serviços essenciais. Para isso, é necessário concentrar todos os arquivos da aplicação em apenas três arquivos. O primeiro é o script que executa a aplicação (*hex.bat*). O segundo é um arquivo compactado que contém a aplicação Hex e suas bibliotecas (*hex6i-linux64.zip*). O último é um arquivo compactado que contém os dados de entrada para serem processados (*inputExemplo.zip*). O *Boinc server* envia estes arquivos para as máquinas *Boinc clients* processarem. Os clientes são capazes de descompactar todos os arquivos *zip's* através de um executável *unzip* e executarem a aplicação. Existem diferentes *wrappers* para os diversos tipos de sistemas operacionais. Portanto, a aplicação científica poderia ser executada na quantidade de sistemas operacionais suportados pelos *wrappers*. Para criar uma aplicação *wrapper* no BOINC é necessário seguir um conjunto de regras. A primeira é organização dos diretórios a partir do diretório apps, como mostra a Figura A.2.

```
$ tree apps/
apps/
└── hex
    └── 0.1
        └── x86_64-pc-linux-gnu
            ├── hex.bat
            ├── job.xml
            ├── unzip_1.00_x86_64-pc-linux-gnu
            ├── version.xml
            └── wrapper_25825_x86_64-pc-linux-gnu

3 directories, 5 files
```

Figura A.2. Diretórios app wrapper Hex

Depois de copiados os respectivos arquivos da aplicação para este diretório, deve-se criar um arquivo chamado *job.xml* e um *version.xml*, como mostram os scripts 3 e 4. Estes arquivos dizem a ordem dos arquivos que serão executados, qual o arquivo será o programa principal e quais os executáveis de descompactação e *wrapper* devem ser

utilizados.

Script 3 Arquivo de configuração job.xml

```
<job_desc>
  <task>
    <application>unzip_x86_64-linux-gnu</application>
    <stdout_filename>stdout.txt</stdout_filename>
    <stderr_filename>stderr.txt</stderr_filename>
    <command_line>-n data1.zip</command_line>
    <weight>2</weight>
  </task>
  <task>
    <application>unzip_x86_64-linux-gnu</application>
    <stdout_filename>stdout.txt</stdout_filename>
    <stderr_filename>stderr.txt</stderr_filename>
    <command_line>-n data2.zip</command_line>
    <weight>2</weight>
  </task>
  <task>
    <application>hex.bat</application>
    <stdout_filename>stdout.txt</stdout_filename>
    <stderr_filename>stderr.txt</stderr_filename>
    <weight>96</weight>
  </task>
</job_desc>
```

Para criar tarefas e receber os resultados delas o *framework* BOINC precisa mapear os arquivos de entrada e de saída de cada tarefa. Isto é feito através dos templates *hex_wu.xml* e *hex_re.xml*. O template de arquivo de entrada é o *hex_wu.xml*. Nele são configurados dois arquivos de entrada que estão compactados e serão descompactados no *BOINC client* pelo programa *unzip*. O template de arquivos de saída é o *hex_re.xml*. Nele são mapeados quatro arquivos de saída. Estes arquivos podem ser vistos nos scripts 5 e 6.

Depois da aplicação ser implantada no *BOINC server*, é possível criar tarefas para processar os arquivos de entrada, anexando um arquivo *zip*. É possível também verificar o estados das tarefas criadas no portal BOINC-Hex, assim como descarregar os resultados das tarefas completadas. Quando o *Boinc client* começa a processar as tarefas, toda a aplicação é descarregada do *BOINC server*. O *BOINC server* sabe como utilizar todos os núcleos dos processadores disponíveis nas máquinas *BOINC client*, como mostra a Figura A.3 do software *BOINC client*. Quando o processamento termina os dados são retornados ao *BOINC server* e o usuário pode acessar através do portal BOINC ou do CSGrid.

O script 7 mostra os comandos para a criação de tarefas que são executados pela biblioteca BOINC-SGA.lua. Antes de criar uma tarefa é necessário utilizar o comando *dir_hier_path* para criar um caminho lógico para o arquivo de entrada que irá ser passado para a tarefa.

É possível verificar o tempo gasto para processar uma tarefa verificando o arquivo de *log hex_job_X_0_2*. Este arquivo é o original *job.log* da aplicação Hex. O endereço

Script 4 Arquivo de configuração version.xml

```
<version>
  <file>
    <physical_name>wrapper_x86_64-linux-gnu</physical_name>
    <main_program/>
  </file>
  <file>
    <physical_name>unzip_x86_64-linux-gnu</physical_name>
    <logical_name>unziP_x86_64-linux-gnu</logical_name>
  </file>
  <file>
    <physical_name>hex.bat</physical_name>
    <logical_name>hex.bat</logical_name>
  </file>
  <file>
    <physical_name>job.xml</physical_name>
    <logical_name>job.xml</logical_name>
  </file>
</version>
```

Script 5 Template hex_wu.xml

```
<file_info>
    <number>0</number>
    <sticky/>
    <no_delete/>
</file_info>
<file_info>
    <number>1</number>
    <sticky/>
    <no_delete/>
</file_info>
<workunit>
    <file_ref>
        <file_number>0</file_number>
        <open_name>data1.zip</open_name>
        <copy_file/>
    </file_ref>
    <file_ref>
        <file_number>1</file_number>
        <open_name>data2.zip</open_name>
        <copy_file/>
    </file_ref>
</workunit>
```

Script 6 Template hex_re.xml

```
<file_info>
    <name><OUTFILE_0/></name>
    <generated_locally/>
    <upload_when_present/>
    <max_nbytes>104857600</max_nbytes>
</file_info>
<file_info>
    <name><OUTFILE_1/></name>
    <generated_locally/>
    <upload_when_present/>
    <max_nbytes>104857600</max_nbytes>
</file_info>
<file_info>
    <name><OUTFILE_2/></name>
    <generated_locally/>
    <upload_when_present/>
    <max_nbytes>104857600</max_nbytes>
</file_info>
<file_info>
    <name><OUTFILE_3/></name>
    <generated_locally/>
    <upload_when_present/>
    <max_nbytes>104857600</max_nbytes>
</file_info>
<result>
    <file_ref>
        <file_name><OUTFILE_0/></file_name>
        <open_name>1CLV_r_u-1CLV_l_u.sum</open_name>
        <copy_file/>
    </file_ref>
    <file_ref>
        <file_name><OUTFILE_1/></file_name>
        <open_name>1CLV_r_u-1CLV_l_u.pdb</open_name>
        <copy_file/>
    </file_ref>
    <file_ref>
        <file_name><OUTFILE_2/></file_name>
        <open_name>job.log</open_name>
        <copy_file/>
    </file_ref>
    <file_ref>
        <file_name><OUTFILE_3/></file_name>
        <open_name>1CLV_r_u-1CLV_l_u_100.pdb</open_name>
        <copy_file/>
    </file_ref>
</result>
```

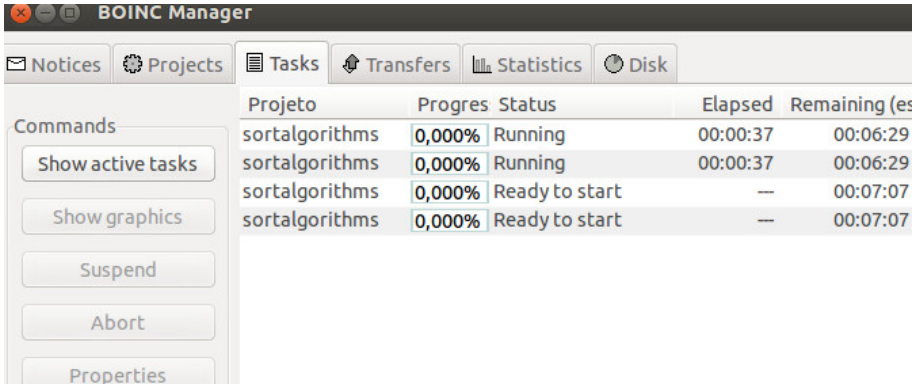

Figura A.3. Cliente BOINC

Script 7 Comando para criar tarefas Hex no BOINC

```
# cp hex6i-linux64.zip `bin/dir_hier_path hex6i-linux64.zip`
# cp inputExemplo.zip `bin/dir_hier_path inputExemplo.zip`

# bin/create_work --appname hex
--wu_template templates/hex_wu.xml
--result_template templates/hex_re.xml
--wu_name hex_job_1 hex6i-linux64.zip inputExemplo.zip
```

do portal BOINC-Hex é http://BOINC-IP/hex_ops, onde o IP é o endereço da máquina do *BOINC server* e o prefixo hex é o nome da aplicação implantada no *BOINC server*. Também é possível verificar através deste endereço o diretório onde estão os resultados processados da aplicação.

A.4. Integração do programa Bubblesort na plataforma mc^2

O programa Bubblesort foi desenvolvido em C++. Ele recebe um arquivo de entrada com números em cada umas das suas linhas e escreve um arquivo de saída com estes números ordenados em ordem crescente utlizando o modelo bubblesort. Esta integração com o BOINC, e consequentemente com a plataforma mc^2, foi feita para exemplificar um programa que não necessitasse do componente *wrapper* utilizado na integração anterior.

O script 9 mostra as principais linhas do programa bubblesort, que tem as funções referentes as bibliotecas do BOINC. Estas funções precisam existir no programa para que o BOINC reconheça como iniciar e parar o programa. Elas são específicas para programas C++. Caso o programa fosse escrito em outra linguagem de programação, as bibliotecas e funções seriam outras. As funções referentes a biblioteca do BOINC são *boinc_init_diagnostics()*, *boinc_init*, *boinc_resolve_filename()*, *boinc_fopen()* e *boinc_finish()*. Este programa foi compilado através de um script *Makefile* 8 que faz referência a todas as bibliotecas do BOINC localizadas no sistema operacional. O script *Makefile* cria um binário executável do programa bubblesort com as bibliotecas do BOINC para ser implantado no *BOINC server*.

Depois que o programa é compilado o executável gerado é maior do que um mesmo programa sem estas bibliotecas do BOINC. Neste novo tipo de aplicação existirá apenas um arquivo que foi implantado no *BOINC server*. Não há mais a necessidade

Script 8 Makefile do bubblesort em C++

```
BOINC_DIR = ../..
BOINC_API_DIR = $(BOINC_DIR)/api
BOINC_LIB_DIR = $(BOINC_DIR)/lib

CXXFLAGS = -g \
    -I$(BOINC_DIR) \
    -I$(BOINC_LIB_DIR) \
    -I$(BOINC_API_DIR) \
    -L /usr/X11R6/lib \
    -L.

ifeq ($(wildcard /usr/local/lib/libglut.a),)
LIBGLUT = /usr/lib/libglut.a
LIBGLU = /usr/lib/libGLU.a
LIBJPEG = /usr/lib/libjpeg.a
else
LIBGLUT = /usr/local/lib/libglut.a
LIBGLU = /usr/local/lib/libGLU.a
LIBJPEG = /usr/local/lib/libjpeg.a
endif

PROGS = bubblesort \

all: $(PROGS)

libstdc++.a:
ln -s `gcc -print-file-name=libstdc++.a`

clean:
/bin/rm -f $(PROGS) *.o libstdc++.a

distclean:
/bin/rm -f $(PROGS) *.o libstdc++.a

install: bubblesort

bubblesort: bubblesort.o libstdc++.a $(BOINC_API_DIR)/
libboinc_api.a $(BOINC_LIB_DIR)/libboinc.a
$(CXX) $(CXXFLAGS) -o bubblesort bubblesort.o
libstdc++.a -pthread \
$(BOINC_API_DIR)/libboinc_api.a \
$(BOINC_LIB_DIR)/libboinc.a
```

Script 9 Programa Bubblesort em C++

```
int main(int argc, char **argv) {
  boinc_init_diagnostics(BOINC_DIAG_REDIRECTSTDERR|
          BOINC_DIAG_MEMORYLEAKCHECKENABLED|
          BOINC_DIAG_DUMPCALLSTACKENABLED|
          BOINC_DIAG_TRACETOSTDERR);

  fprintf(stderr,"Hello, stderr!\n");
  rc = boinc_init();
  rc = boinc_resolve_filename("out.txt",
        resolved_name, sizeof(resolved_name));
  f = boinc_fopen(resolved_name, "a");

  for(i=0;i<(totalNums-1);i++) {
    for(j=0;j<(totalNums-(i+1));j++) {
      if(numbers[j] > numbers[j+1]) {
        int temp;
        temp = numbers[j];
        numbers[j] = numbers[j+1];
        numbers[j+1] = temp;
      }
    }
  }
  for(i=0;i<totalNums;i++) {
    fprintf(fpout, "%d\t", numbers[i]);
  }
  fprintf(fpout, "\n");
  fclose(file);
  fclose(f);
  boinc_finish(0);
}
```

de existir os arquivos *job.xml* e *version.xml*. Os arquivos de templates para as workunits e para os resultados são configurados como mostram os scripts 10 e 11.

Estes templates são necessários para que as tarefas do projeto saibam como serão as entradas e saídas da aplicação que será executada. No template *workunit* é configurado o nome do arquivo para a criação de tarefas como *in.txt*. Os parâmetros *min_quorum* e *target_results* são responsáveis pela redundância na verificação do correto processamento da tarefa. Como dito anteriormente, este parâmetro foi configurado para a plataforma mc^2 pois todos os componentes estão numa rede confiável. No arquivo *workunit.xml* é possível configurar o nome e a quantidade de arquivos de saída da aplicação. Nesta aplicação o nome escolhido foi *out.txt*.

Script 10 Template workunit.xml

```
<file_info>
  <number>0</number>
</file_info>
<workunit>
  <file_ref>
    <file_number>0</file_number>
    <open_name>in</open_name>
    <copy_file/>
  </file_ref>
  <rsc_fpops_est>5e13</rsc_fpops_est>
  <rsc_fpops_bound>5e15</rsc_fpops_bound>
  <rsc_memory_bound>1e8</rsc_memory_bound>
  <rsc_disk_bound>1e8</rsc_disk_bound>
  <delay_bound>250000</delay_bound>
  <min_quorum>1</min_quorum>
  <target_nresults>1</target_nresults>
  <max_error_results>5</max_error_results>
</workunit>
```

A.5. Integração do benchmark NAS na plataforma mc^2

O terceiro estudo de caso foi realizado com o *benchmark NAS*, um *benchmark* muito utilizado na comunidade de sistemas distribuídos. Este *benchmark* consegue ocupar por completo a capacidade do processador da máquina e seu objetivo é medir a evolução de desempenho de supercomputadores em paralelo. A Figura A.4 apresenta a saída da execução deste *benchmark* em um computador que possui um processador Core 2 Duo 2.00GHz e 1 GB de RAM. A medição do tempo foi retirada através do comando *time* do Linux e não do tempo do próprio *benchmark*.

Para executar este programa no BOINC foi necessário ter um programa escrito em C++ que pudesse executar o *benchmark* escrito em Fortran. Foi usado o mesmo template de programa do subapêndice A.4, com as alterações para chamar o binário em Fortran. A principal função inserida no código C++ é a *boinc_resolve_filename()* que chama o programa Fortran.

Script 11 Template result.xml

```
<file_info>
  <name><OUTFILE_0/></name>
  <generated_locally/>
  <upload_when_present/>
  <max_nbytes>99999999999999999999999999999</max_nbytes>
  <url><UPLOAD_URL/></url>
</file_info>
<result>
  <file_ref>
    <file_name><OUTFILE_0/></file_name>
    <open_name>out</open_name>
    <copy_file/>
  </file_ref>
</result>
```

O *BOINC client* também teve que ser configurado para que fosse possível que suas tarefas consumissem mais do que o padrão quando o programa é instalado. Na instalação padrão do *BOINC client*, suas tarefas consomem apenas 50% do processador do computador e o processamento só inicia quando o computador está em modo de descanso (na proteção de tela). Portanto, foram alteradas essas configurações de acordo com a Figura A.5.

A.6. Integração do MEGA-CC na plataforma mc^2

O último estudo de caso com o módulo do BOINC foi com o programa MEGA-CC [Tamura et al. 2011]. Ele é uma ferramenta integrada para a realização de alinhamento de sequências, inferir árvores filogenéticas, mineração de bases de dados baseados na web, taxas de estimativa de evolução molecular, inferir sequências ancestrais e testar hipóteses evolutivas. O MEGA-CC é usado pelos biólogos em um grande número de laboratórios para reconstruir a história evolutiva das espécies e inferir a extensão e a natureza das forças seletivas que moldam a evolução dos genes e espécies.

Para a configuração do MEGA-CC no BOINC foi necessário utilizar os componentes *wrapper* para plataformas Windows intel x86, visto que o MEGA-CC só está disponível para Windows e MacOS. A estrutura de diretórios da aplicação MEGA-CC no BOINC é determinada pelo nome da aplicação, a versão e a plataforma (apps/m52cc/0.1/windows_intelx86). Neste diretório temos o script *executar.bat*, os arquivos de configuração *job.xml* e *version.xml* e o programa MEGA-CC com seu arquivo de configuração.

Para iniciar a execução do MEGA-CC através de uma linha de comando basta passar os parâmetros de arquivos de configuração e de entrada. Este arquivo é o *executar.bat* descrito no script 12.

Script 12 Script executar.bat de inicialização do MEGA-CC no BOINC

```
M52CC.exe -a protmodel.mao -d data.meg -o out.csv
```

```
$ time ./bt.C.x
 NAS Parallel Benchmarks (NPB3.3-OMP) - BT Benchmark
 No input file inputbt.data. Using compiled defaults
 Size:  162x 162x 162
 Iterations:  200       dt:    .0001000
 Time step    1
 Time step   20
 Time step   40
 Time step   60
 Time step   80
 Time step  100
 Time step  120
 Time step  140
 Time step  160
 Time step  180
 Time step  200
 Verification being performed for class C
 accuracy setting for epsilon =   .1000000000000E-07
 Comparison of RMS-norms of residual
           1  .6239811655176E+04  .6239811655176E+04  .1107751341780E-13
           2  .5079323919042E+03  .5079323919042E+03  .1242216404017E-13
           3  .1542353009301E+04  .1542353009301E+04  .5734637869153E-13
           4  .1330238792929E+04  .1330238792929E+04  .1247766821685E-13
           5  .1160408742844E+05  .1160408742844E+05  .6113413672190E-14
 Comparison of RMS-norms of solution error
           1  .1646200836909E+03  .1646200836909E+03  .1191287178775E-13
           2  .1149710790382E+02  .1149710790382E+02  .2549327021655E-13
           3  .4120744620746E+02  .4120744620746E+02  .8104241260813E-14
           4  .3708765105969E+02  .3708765105969E+02  .1915847230703E-13
           5  .3621105305184E+03  .3621105305184E+03  .3045374914452E-13
 Verification Successful
 BT Benchmark Completed.
real    39m12.612s
user    38m44.581s
sys     0m2.516s
```

Figura A.4. Execução do benchmark NAS

O arquivo *XML job.xml*, apresentado no script 13, é necessário para dizer qual é a aplicação que será executada no *BOINC client*. A tag <*weight*> é usada para dividir o peso de processamento da aplicação. Se existisse mais de uma aplicação o peso iria ser dividido entre elas e a porcentagem de execução no *BOINC client* iria seguir estes pesos.

Script 13 Arquivo job.xml do MEGA-CC no BOINC

```
<job_desc>
  <task>
    <application>executar.bat</application>
    <weight>1</weight>
    <append_cmdline_args/>
  </task>
</job_desc>
```

O arquivo *XML version.xml*, apresentado no script 14, é necessário para mapear os arquivos que serão enviados para o *BOINC client*. Como o *software* MEGA-CC e legado, ou seja, não é *open-source*, é necessário usar o componente *wrapper* para iniciar e parar as tarefas no *BOINC client*. Estes componentes podem ser baixados no site do BOINC. Os outros arquivos mapeados se referem a aplicação MEGA-CC e ao *job.xml* explicado anteriormente. A tag <*copy_file/*> é necessária para copiar os arquivos para o cliente. Caso ela seja omitida o *BOINC client* não conseguirá achar os arquivos.

Script 14 Arquivo version.xml do MEGA-CC no BOINC

```
<version>
  <file>
    <physical_name>wrapper_26009_windows_intelx86.exe
    </physical_name>
    <main_program/>
  </file>
  <file>
    <physical_name>executar.bat</physical_name>
    <logical_name>executar.bat</logical_name>
  </file>
  <file>
    <physical_name>M52CC.exe</physical_name>
    <copy_file/>
    <logical_name>M52CC.exe</logical_name>
  </file>
  <file>
    <physical_name>protmodel.mao</physical_name>
    <copy_file/>
    <logical_name>protmodel.mao</logical_name>
  </file>
  <file>
    <physical_name>job.xml</physical_name>
    <logical_name>job.xml</logical_name>
  </file>
</version>
```

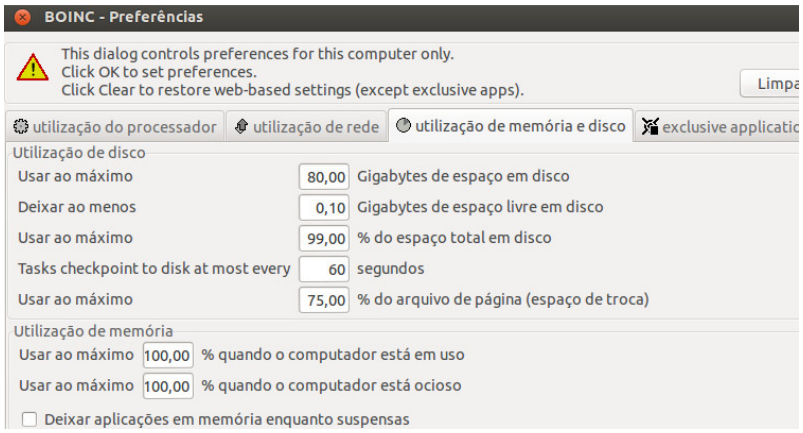

Figura A.5. Preferências do BOINC client

Os arquivos *m52cc_wu.xml* e *m52cc_re.xml*, apresentados respectivamente nos scripts 15 e 16, são os templates para a criação de *workunits* e os resultados das tarefas do MEGA-CC. O template *wu* mapeia os arquivos que serão enviados como parâmetro de entrada para o MEGA-CC. O *data.meg* será o único arquivo de entrada. O *protmodel.mao* é um arquivo de configuração e como ele é sempre o mesmo podemos passar ele no script 12. O template *re* mapeia o único arquivo de saída do MEGA-CC, o *out.csv*.

Script 15 Arquivo m52cc_wu.xml do MEGA-CC no BOINC

```
<input_template>
  <file_info>
    <number>0</number>
    <sticky/>
    <no_delete/>
  </file_info>
  <workunit>
    <file_ref>
      <file_number>0</file_number>
      <open_name>data.meg</open_name>
      <copy_file/>
    </file_ref>
  </workunit>
</input_template>
```

Para criar *workunits* (tarefas) no projeto BOINC-MEGA-CC é usado o comando *create_work*, descrito no script 17. Como não é necessário utilizar do recurso de redundância, foi passado o parâmetro *-min_quorum 1*. O nome da aplicação é especificado através do parâmetro *-appname m52cc*, os templates utilizados *-wu_template templates/m52cc_wu.xml* e *-result_template templates/m52cc_re.xml*, o nome da *workunit* *-wu_name m52cc_job_001* e o arquivo de parâmetro de entrada *data.meg*. Para que este arquivo de entrada seja achado pelo BOINC é necessários criar um link para ele. Isso é feito utilizando o comando *stage_file*.

Script 16 Arquivo m52cc_re.xml do MEGA-CC no BOINC

```
<output_template>
  <file_info>
    <name><OUTFILE_0/></name>
    <generated_locally/>
    <upload_when_present/>
    <max_nbytes>104857600</max_nbytes>
    <url><UPLOAD_URL/></url>
  </file_info>
  <result>
    <file_ref>
      <file_name><OUTFILE_0/></file_name>
      <open_name>out.csv</open_name>
      <copy_file/>
    </file_ref>
  </result>
</output_template>
```

Script 17 Script para criar tarefas do MEGA-CC no BOINC

```
#! /bin/sh
bin/stage_file inputs/data.meg
bin/create_work -target_nresults 1 -min_quorum 1
  -appname m52cc -wu_template templates/m52cc_wu.xml
  -result_template templates/m52cc_re.xml
  -wu_name m52cc_job_001 data.meg
```

Apêndice B

Implementação com computação intensiva de dados

B.1. Introdução

A implementação de computação intensiva de dados na plataforma mc^2 foi realizada com o *framework open-source* Hadoop da Apache. Este *framework* tem uma biblioteca de *MapReduce* e de um sistema arquivos distribuídos (*HDFS*).

A parte principal do arquivo *sgad-cnf.lua* é apresentado no script 18. Ele segue o mesmo padrão do arquivo de configuração do BOINC apresentado no script 4.1, exceto o fato de ser uma outra máquina e, portanto, com um IP diferente. Neste caso, o CSGrid também é responsável pela transferência de arquivos para serem processados no Hadoop. Também foi feita esta transferência utilizando o protocolo CSFS, pois as máquinas podem estar em redes diferentes, apesar da recomendação do Hadoop ser para redes privadas.

B.2. Implementação do SGA para o Hadoop

A integração do Hadoop com o CSGrid foi feita através de um SGA (Servidor de Gerência de Algoritmos). Este SGA foi configurado da mesma forma da configuração com o BOINC, para que o CSGrid tenha total controle sobre o Hadoop. Sendo assim, o Hadoop é uma única vez configurado. Sua configuração consiste na implantação de uma aplicação científica, a criação e verificação de tarefas. Tudo isso acontece pela manipulação de arquivos no sistems de arquivos distribuido do Hadoop, o *HDFS*.

O principal componente desta integração é a configuração de um arquivo na máquina do SGA para que esta seja reconhecida como um nó do CSGrid. Este arquivo é o sgad-cnf.lua e sua principal parte é apresentada no script 18. Os dados de entrada para processamento das tarefas no Hadoop estarão disponíveis no CSGrid. Este *framework* é que será responsável pela transferência dos dados para o computador onde está instalado o SGA. Essa transferência pode ser feita através da configuração de um diretório NFS (*Network File System*), no caso de uma rede local, ou através de CSFS no caso em que os computadores estão em redes diferentes. O atributo *name* é o *localhost* da máquina do SGA. O atributo *platform_id* indica a plataforma do sistema operacional da máquina, no nosso caso é um sistema Linux que usa um processador de 64Mb. Logo depois temos a quantidade de números de processadores ou núcleos a serem utilizados pelo SGA. O

último parâmetro é composto pela configuração do CSFS. Nele precisamos especificar o IP da máquina do SGA, o diretório de compartilhamento de arquivos, o usuário que deve ter acesso via protocolo ssh sem a necessidade de senha e a porta para comunicação, que no nosso caso é a 10000.

Script 18 Script de configuração do SGA para o Hadoop

```
Node{
  name = "sga-ufba-cloud6",
  platform_id = "Linux26g4_64",
  num_processors = 2,
  memory_ram_info_mb = 2048,
  memory_swap_info_mb = 5376,
  csfs = {
    launch_daemon = YES,
    properties = { HOST="192.168.188.6",
    ROOT_DIR="/home/cloud6/sga_v1.6_u020/sgad/csfs/root/",
    USER = "cloud6", PORT = 10000 }
  }
}
```

B.3. Integração do programa Grep MapReduce na plataforma mc^2

Para criar uma aplicação para ser implantada no Hadoop foi necessário seguir o paradigma de computação distribuída conhecida como *MapReduce*. O pseudoalgoritmo 2 detalha o programa *Map* e o programa *Reduce*. A classe *Mapper* recebe um documento com um ID. Cada linha do documento é extraída num laço iterativo e a cada conjunto de 10 linhas, opção configurável, uma nova chave *k* é criada. Esta chave é passada como parâmetro na função *Emit* da biblioteca *Map*. A função *Reduce* irá receber um conjunto de linhas com a mesma chave, transmitidas pela função *Map*. Como cada conjunto de 10 linhas uma nova chave é criada na função *Map*, cada computador que processar a função *Reduce* irá processar um conjunto de 10 linhas. O método *getStringToGrep()* retorna a *string* que será procurada nos arquivos. Se a *string* for encontrada na linha processada a execução irá entrar na condição e a função *Emit* da biblioteca *Reduce* irá escrever no arquivo de saída.

Este programa foi implementado usando a linguagem de programação Java juntamente com as bibliotecas do Hadoop. Depois de criadas as classes *Map* e *Reduce* foi necessário criar uma classe principal que é responsável por iniciar o programa e fazer referência as bibliotecas *MapReduce*. Depois que elas são compiladas foi necessário criar um pacote *jar* que será a biblioteca a ser chamada pelo *framework* Hadoop. O executável *hadoop*, localizado dentro do diretório bin na instalação do Hadoop, é responsável por executar a aplicação *MapReduce*. No CSGrid foi necessário criar um novo algoritmo que chamamos de HADOOP-GREP. A estrutura de diretórios que foi criada segue como está demonstrado na Figura B.1. Nesta configuração é possível perceber que existe no mesmo CSGrid o algoritmo de integração com o BOINC para a aplicação HEX, além de outros já disponibilizados na instalação do CSGrid. Abaixo da estrutura do diretório *bin* é possível adicionar as plataformas habilitadas para executar o nosso SGA e o executável

Algorithm 2 Algoritmo Grep MapReduce

```
1: contador ← 0
2: key ← 0
3: function MAP(chave, documento)
4:     linha ← empty
5:     for all linha ∈ documento do                ▷ todas as linhas do documento
6:         if (contador%10 = 0) then
7:             key ← key + 1
8:         end if
9:         contador ← contador + 1
10:        Emit(key, linha)
11:    end for
12: end function
13: function REDUCE(chave, lista)
14:    pattern ← getStringToGrep()
15:    for all linha ∈ lista do                    ▷ todas as linhas da lista
16:        if pattern ⊂ linha then
17:            Emit(k, linha)
18:        end if
19:    end for
20: end function
```

responsável por executar o programa no SGA. Este executável é responsável por verificar se o *framework* Hadoop está iniciado na máquina do SGA, executar a função *grep* e retornar os resultados para o arquivo *out.txt*. Se algumas destas operações falharem o SGA retornará o inteiro 1 na saída.

Script 19 Comando Grep para o Hadoop

```
# /usr/local/hadoop/bin/hadoop jar grep.jar Main
/user/hduser/grep /user/hduser/grep-out regex
```

O principal comando no algoritmo *executar-sga-hadoop-grep.sh* é o de execução da aplicação *grep*. Ele é apresentado no script 19. O arquivo *config.xml* é o configurador responsável por criar a interface de parâmetros do SGA-Hadoop. Ela tem o objetivo de captar os parâmetros utilizados no programa *grep*, o arquivo de entrada, a *string* que será procurada e o arquivo de saída. Ela é apresentada na Figura B.2. No texto de saída é possível verificar a porcentagem da execução das funções *Map* e *Reduce*. Algo como *map 0% reduce 0%*, *map 100% reduce 0%*, *map 100% reduce 33%* e *map 100% reduce 100%*.

B.4. Integração do programa CloudBurst na plataforma mc^2

Para processar alguns genomas no CloudBurst é necessário gerar alguns arquivos de entrada disponíveis no site do CloudBurst. Estes arquivos são binários e precisam ser processados e convertidos para um tipo compatível. No site do CloudBurst também é possível fazer o *download* desta biblioteca. Neste segundo link também está contido o programa *MapReduce* que foi implantado no Hadoop. Para utilizar os arquivos de entrada

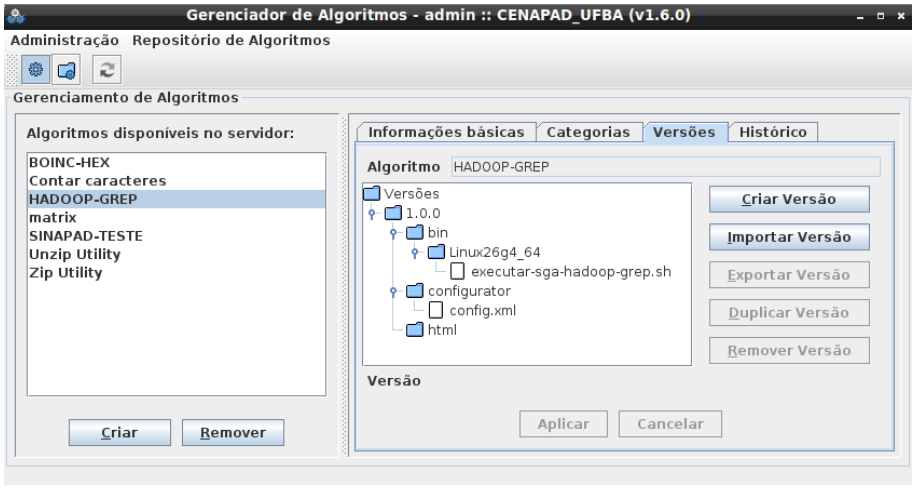

Figura B.1. Gerenciador de algoritmos CSGrid

Figura B.2. Interface do SGA-Hadoop no CSGrid

é necessário convertê-los da extensão *.fa* para a *.br* utilizando o pacote *ConvertFastaForCloud.jar*, conforme descreve o script 20.

Feito o *donwload* dos arquivos, é possível começar a administrar o sistema de arquivos distribuído do Hadoop e copiar os dados de entrada para o *HDFS*. A implementação foi feita da seguinte forma. Foram copiados os arquivos de entrada para um diretório novo no *HDFS* e depois executado o processamento destes arquivos com o programa *CloudBurst.jar*. Como este programa foi implementado utilizando o modelo de programação *MapReduce*, é necessário executá-lo através do Hadoop. Além dos dois arquivos de entrada, existem mais 12 parâmetros que precisam ser passados na linha de comando de execução do CloudBurst. Portanto, no total são necessários 14 parâmetros para a execução do CloudBurst. A descrição detalhada destes parâmetros pode ser verificada quando é executado o programa com uma quantidade de parâmetros menor que o programa espera. O script 21 demonstra as linhas de comando para as execuções descritas.

No CSGrid foi criado um projeto chamado HADOOP-CLOUDBURST através do menu "*Gerenciador de Algoritmos*". Neste projeto foi criado uma estrutura de diretórios

Script 20 Convertendo arquivos para o CloudBurst

```
$ java -jar ConvertFastaForCloud.jar 100k.fa re.br
Converting 100k.fa into re.br
13/09/09 12:37:41 WARN util.NativeCodeLoader: Unable to
load native-hadoop library for your platform... using
builtin-java classes where applicable
Processed 100000 sequences
min_seq_len: 36
max_seq_len: 36
Using DNAString version: 2.0

$ java -jar ConvertFastaForCloud.jar s_suis.fa qry.br
Converting s_suis.fa into qry.br
13/09/09 13:19:35 WARN util.NativeCodeLoader: Unable to
load native-hadoop library for your platform... using
builtin-java classes where applicable
In 1... 2007491bp
  32 chunks
Processed 1 sequences
min_seq_len: 2007491
max_seq_len: 2007491
Using DNAString version: 2.0
```

Script 21 Executando o CloudBurst no Hadoop

```
$ bin/hadoop dfs -mkdir /user/hduser/cloudburst/
$ bin/hadoop dfs -put re.br /user/hduser/cloudburst/
$ bin/hadoop dfs -put qry.br /user/hduser/cloudburst/
$ bin/hadoop jar CloudBurst.jar
/user/hduser/cloudburst/qry.br /user/hduser/cloudburst/re.br
/user/hduser/cloudburst-results 36 38 3 0 1 20 4 2 2 128 16
```

como mostra a Figura B.3 para que seja alocado o script do SGA-Hadoop e o arquivo *config.xml* que determina quais são os parâmetros necessários para a aplicação CloudBurst. Nesta Figura também é possível observar que os arquivos de entrada para a aplicação CloudBurst estão incluídos no projeto do CSGrid. O arquivo *config.xml* contém os dados apresentados no script 22.

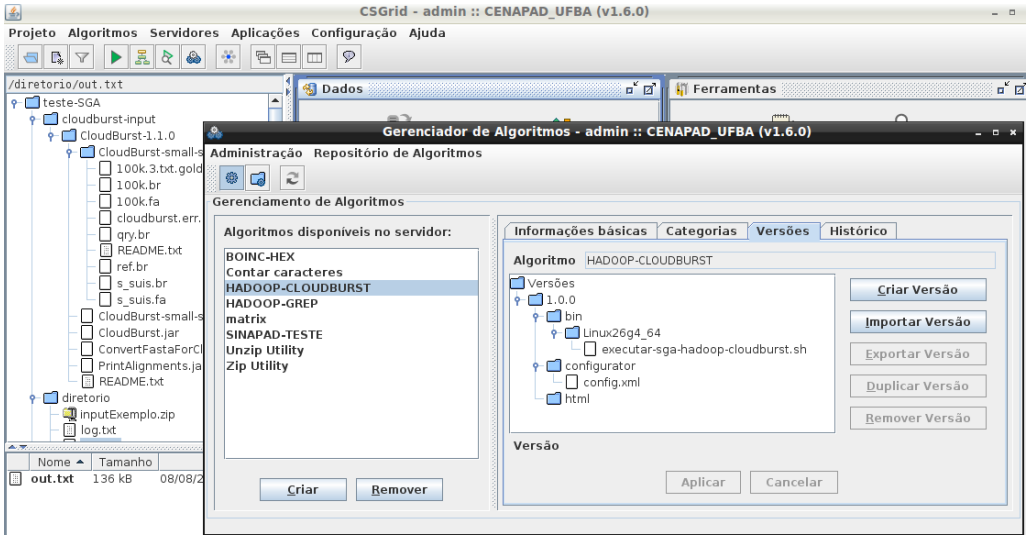

Figura B.3. Projeto do SGA-Hadoop no CSGrid

Script 22 config.xml do SGA-Hadoop para o CloudBurst

```
<?xml version='1.0' encoding='iso8859-1'?>
<algoritmo comando="executar-sga-hadoop-cloudburst.sh">
  <formato_no_comando>$VALOR_DO_PARAMETRO</formato_no_comando>
  <grupo rotulo='Parametros'>
    <arquivo_de_entrada nome="INPUT-00" rotulo="Ref"/>
    <arquivo_de_entrada nome="INPUT-01" rotulo="Qry"/>
    <arquivo_de_saida nome='OUTPUT' rotulo='Saida'
    tipo='TEXT'/>
  </grupo>
  <log/>
</algoritmo>
```

O arquivo *executar-sga-hadoop-cloudburst.sh* é o script que será executado pelo SGA-HADOOP na máquina remota. O conteúdo deste arquivo é apresentado no script 23. Ele deve ter permissão para execução, já que será executado na máquina do SGA-Hadoop. Ele se resume em duas funções. A primeira função a ser executada é a *verifyHadoop*. Ela verifica se o Hadoop está iniciado na máquina mestre, ou seja, onde o SGA-Haddop está instalado. A função *executeCloudBurst* transfere os dois arquivos de entrada para a máquina do SGA-Hadoop, recolhe o resultado no arquivo *out.txt* e excreve na saída $3. O controle de execução com sucesso ou falha deste script se resume em retornar o valor 0 caso a operação ocorra com sucesso ou 1 caso aja uma falha.

Na máquina remota do SGA-HADOOP também é necessário ter dois scripts para gerenciar o sistema de arquivos distribuídos do Hadoop (*HDFS*) e executar o programa

Script 23 script do SGA-Hadoop para o CloudBurst

```
#!/bin/ksh

function executeCloudBurst {
    exe="cd /usr/local/hadoop/; ./execute-cloudburst.sh
$1 $2 out.txt;"
    result=`sshpass -p 'qweasd' ssh hduser@192.168.188.6
"$exe"`

    result=`sshpass -p 'qweasd' ssh hduser@192.168.188.6
"cat /usr/local/hadoop/out.txt"`
    echo $result >> $3
}

function verifyHadoop {
    exe="cd /usr/local/hadoop/; ./verificaJps.sh out.txt;"
    echo "$exe"
    result=0
    result=`sshpass -p 'qweasd' ssh hduser@192.168.188.6
"$exe"`
    if [ $result = 1 ]; then
        echo "Hadoop is running..." > $1
        result=`sshpass -p 'qweasd' ssh hduser@192.168.188.6
"cat /usr/local/hadoop/out.txt"`
        echo $result >> $1
    elif [ $result = 0 ]; then
        echo "Hadoop is not running!" > $1
        result=`sshpass -p 'qweasd' ssh hduser@192.168.188.6
"cat /usr/local/hadoop/out.txt"`
        echo $result >> $1
        exit 1
    fi
}

verifyHadoop $3

executeCloudBurst $1 $2 $3

exit 0
```

CloudBurst. Estes script são o *verificaJps.sh* e o *execute-cloudburst.sh*. Com o primeiro script 24 é possível obter uma resposta indicando se o Hadoop está executando normalmente. Através dele é verificado se os serviços *TaskTracker*, *NameNode*, *JobTracker* e *DataNode* estão ativos. Com o segundo script 25 é possível excluir o diretório *HDFS* onde ficará os arquivos de entrada, recriá-los, copiá-los e executar o CloudBurst no Hadoop.

Através deles é possível excluir os diretórios de entrada e saída de dados, recriar um novo diretório e copiar os arquivos de entrada para este diretório. Depois é executado o comando *bin/hadoop* para iniciar o processamento *MapReduce*.

B.5. Integração do programa BLAST na plataforma mc^2

O BLAST é um algoritmo para comparação de informações de sequências biológicas primárias. Para realizar este processamento é necessário utilizar alguns pacotes de arquivos e programas *MapReduce*, escritos em Java, disponíveis no site do SALSA *(Service Aggregated Linked Sequential Activities)*. O processamento realizado pelo programa BLAST tem como entrada arquivos FASTA, formato de arquivos para representar sequências de nucleotídeos ou de aminoácidos. O script 26 demonstra as linhas de comando para iniciar o processamento de uma tarefa no BLAST integrado com a plataforma mc^2.

O arquivo *executar-sga-hadoop-blast.sh* é o script que será executado pelo SGA-HADOOP na máquina remota. O conteúdo deste arquivo é muito semelhante ao SGA-Hadoop do CloudBurst, apresentado no script 23. Ele deve ter permissão para execução, já que será executado na máquina do SGA-Hadoop. Ele se resume em duas funções. A primeira função a ser executada é a *verifyHadoop*. Ela verifica se o Hadoop está iniciado na máquina mestre, ou seja, onde o SGA-Haddop está instalado. A função *executeBlast* transfere os dois arquivos de entrada para a máquina do SGA-Hadoop, recolhe o resultado no arquivo *out.txt* e excreve na saída $3. O controle de execução com sucesso ou falha deste script se resume em retornar o valor 0 caso a operação ocorra com sucesso ou 1 caso aja uma falha.

Na máquina remota do SGA-HADOOP também é necessário ter dois scripts para gerenciar o sistema de arquivos distribuídos do Hadoop (*HDFS*) e executar o programa BLAST. Estes script são o *verificaJps.sh*, descrito no subapêndice anterior, e o *execute-blast.sh*. Com o primeiro script 24 é possível obter uma resposta indicando se o Hadoop está executando normalmente. Através dele é verificado se os serviços *TaskTracker*, *NameNode*, *JobTracker* e *DataNode* estão ativos. Com o segundo script 26 é possível excluir o diretório *HDFS* onde ficará os arquivos de entrada, recriá-los, copiá-los e executar o BLAST no Hadoop. Através deles é possível excluir os diretórios de entrada e saída de dados, recriar um novo diretório e copiar os arquivos de entrada para este diretório. Depois é executado o comando *bin/hadoop* para iniciar o processamento *MapReduce*.

Script 24 script para verificar o Haddop

```
#!/bin/ksh
> $1
flag=""
verify=`/usr/java/jdk1.6.0_34/bin/jps | grep TaskTracker`
if [ -n "$verify" ]; then
    echo "TaskTracker running\n" #>> $1
else
    echo "TaskTracker not running\n" >> $1
    flag="true"
fi
verify=`/usr/java/jdk1.6.0_34/bin/jps | grep NameNode`
if [ -n "$verify" ]; then
    echo "NameNode running\n" #>> $1
else
    echo "NameNode not running\n" >> $1
    flag="true"
fi
verify=`/usr/java/jdk1.6.0_34/bin/jps | grep JobTracker`
if [ -n "$verify" ]; then
    echo "JobTracker running\n" #>> $1
else
    echo "Jobtracker not running\n" >> $1
    flag="true"
fi
verify=`/usr/java/jdk1.6.0_34/bin/jps | grep DataNode`
if [ -n "$verify" ]; then
    echo "DataNode running\n" #>> $1
else
    echo "DataNode not running\n" >> $1
    flag="true"
fi
if [ -n "$flag" ]; then
    #Error!
    echo 0
    exit
fi
echo 1
```

Script 25 script do SGA-Hadoop para o CloudBurst

```
#!/bin/ksh

remove="/usr/local/hadoop/bin/hadoop dfs -rmr
/user/hduser/cloudburst-results"
result=`$remove`

remove="/usr/local/hadoop/bin/hadoop dfs -rmr
/user/hduser/cloudburst"
result=`$remove`

create="/usr/local/hadoop/bin/hadoop dfs -mkdir
/user/hduser/cloudburst"
result=`$create`

copy="/usr/local/hadoop/bin/hadoop dfs -copyFromLocal $1
/user/hduser/cloudburst/"
result=`$copy`

copy="/usr/local/hadoop/bin/hadoop dfs -copyFromLocal $2
/user/hduser/cloudburst/"
result=`$copy`

exe="/usr/local/hadoop/bin/hadoop jar CloudBurst.jar
/user/hduser/cloudburst/qry.br /user/hduser/cloudburst/ref.br
/user/hduser/cloudburst-results 36 38 3 0 1 20 4 2 2 128 16"
result=`$exe`

echo "part-00000" >> out.txt
list="/usr/local/hadoop/bin/hadoop dfs -cat
/user/hduser/cloudburst-results/part-00000"
result=`$list`
echo "$result" > out.txt

echo "part-00001" > out.txt
list="/usr/local/hadoop/bin/hadoop dfs -cat
/user/hduser/cloudburst-results/part-00001"
result=`$list`
echo "$result" > out.txt

result=`cat out.txt`
echo "$result \n" >> $3

exit 0
```

Script 26 script do SGA-Hadoop para o BLAST

```
#!/bin/ksh

remove="rm -R /usr/local/hadoop/blast-output/"
result=`$remove`

create="mkdir /usr/local/hadoop/blast-output"
result=`$create`

remove="/usr/local/hadoop/bin/hadoop dfs -rmr
 /user/hduser/blast/blast_output"
result=`$remove`

remove="rm -R /usr/local/hadoop/blast-input"
result=`$remove`

create="mkdir /usr/local/hadoop/blast-input"
result=`$create`

untar="tar -C /usr/local/hadoop/blast-input/ -xzvf $1"
result=`$untar`

/usr/local/hadoop/bin/hadoop jar
 /usr/local/hadoop/blast-hadoop.jar
 /user/hduser/blast/BlastProgramAndDB.tar.gz bin/blastx
 /tmp/hadoop-test/ db nr /user/hduser/blast/blast_input/
 /user/hduser/blast/blast_output/ '-query #_INPUTFILE_#
 -outfmt 6 -seg no -out #_OUTPUTFILE_#'

copy="/usr/local/hadoop/bin/hadoop dfs -copyToLocal
 /user/hduser/blast/blast_output/*
 /usr/local/hadoop/blast-output/"
result=`$copy`

> /usr/local/hadoop/output/out.txt

for arq in /usr/local/hadoop/blast-output/*; do
        exe="cat $arq"
        #echo "$exe"
        result=`$exe`
        echo "$result \n" >> /usr/local/hadoop/out.txt
done

result=`cat /usr/local/hadoop/out.txt`
echo "$result \n" > $2

exit 0
```

Mestrado Multiinstitucional em Ciência da Computação - MMCC

MMCC-IM-UFBA, Campus de Ondina
Av. Adhemar de Barros S/N, Salvador - Bahia, CEP 40.170-110
ceapgmat@ufba.br http://wiki.dcc.ufba.br/MMCC