UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

JULIO CÉSAR SANTOS DOS ANJOS

Adequação da Computação Intensiva em Dados para Ambientes *Desktop Grid* com uso de *MapReduce*

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Cláudio Fernando Resin Geyer
Orientador

Porto Alegre, abril de 2012

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Santos dos Anjos, Julio César

Adequação da Computação Intensiva em Dados para Ambientes *Desktop Grid* com uso de *MapReduce*
/ Julio César Santos dos Anjos. – Porto Alegre: PPGC da UFRGS, 2012.

113 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2012. Orientador: Cláudio Fernando Resin Geyer.

1. Sistemas Distribuídos. 2. MapReduce. 3. Desktop Grid. I. Resin Geyer, Cláudio Fernando. II. Título.

“Pax Domini sit semper vobiscum,
et cum Spiritu tuo.”
— Papa João Paulo II

AGRADECIMENTOS

Durante vários meses a seção de agradecimentos continha somente a seguinte frase: *"Agradeço a todos..."*. Ao término de um árduo trabalho, é importante avaliar o caminho percorrido e relembrar dos momentos em que alguém dedicou parte de seu tempo para dividir ideias e compartilhar sonhos. Nem todos os momentos serão relembrados certamente. Porém, como relâmpagos numa noite de chuva, surge na memória alguma cena para relembrar uma situação vivida.

A primeira coisa que pensei foi agradecer a Deus, que me deu o sopro da vida, por ter conseguido chegar ao fim dessa etapa. Logo em seguida, pensei em quem está muito perto de mim, palavras não serão suficientes para retribuir a dedicação, as horas de convivência perdidas e as noites mal dormidas, que minha esposa Eveline passou comigo nestes anos de estudo e, em especial, nos últimos meses de trabalho intenso.

Gostaria poder agradecer ao Wagner Kolberg, um grande amigo que me ajudou desde o final de sua graduação nas pesquisas com o *MapReduce*. Aos parceiros Valderi Leithardt, Pedro Marcos e Steffano pelo apoio e ajuda nas horas mais difíceis. Aos demais colegas da sala 205 meu especial agradecimento pelas críticas, convivência e parceria.

Ao meu amigo Prof. Mariano Nicolau, que me deu o primeiro empurrão para retomar os estudos após 17 anos da minha formatura em engenharia na PUC/RS. As palavras de incentivo serviram para mostrar uma nova estrada não experimentada. Ao mesmo tempo, renovou minha certeza de que pode-se mudar a vida completamente, desde que haja persistência, suor e força de vontade.

Ao Alexandre Miyazaki pelo apoio durante a execução dos testes. Ao amigo Alan Malta que, mesmo estando no CERN/Suíça, colaborou com suas críticas. Ao colega Eduardo Bezerra que, de Lugano/Suíça, deu um grande apoio e aos colegas da Compy pelo suporte dispensado nos últimos meses. Ainda, quero agradecer, aos meus pais e meus irmãos, pelo apoio e pelo carinho nestes anos.

Ao meu orientador Prof. Cláudio Geyer queria agradecer a paciência e a dedicação dispensada durante estes anos todos. A sua aposta na ideia de estudar o *MapReduce* foi fundamental para materializar este trabalho e o grupo que se formou em tono do tema. Também quero agradecer à Profª. Luciana Arantes pela leitura crítica de meus textos e por seu incansável trabalho de avaliação de minhas propostas. O seu trabalho de Co-orientação, durante o Mestrado, foi uma experiência muito gratificante para mim. Em fim, *"agradeço a todos..."*

Finalmente, queria agradecer ao CNPq através do Edital MCT/CNPq Nº 70/2009 – PGAEST- MCT/CNPq. A FAPERGS através do Edital FAPERGS/CNPq Nº 008/2009 Projeto de Pesquisa: "GREEN-GRID: Computação de Alto Desempenho Sustentável" que financiaram parcialmente esta pesquisa. A Grid5000, uma *grid* desenvolvida pelo projeto INRIA ALADDIN, suportado pelo CNRS, RENATER e várias Universidades.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

Alg.	Algoritmo(s).
API	*Application Programming Interface* - Interface de Programação.
BOINC	*Berkeley Open Infrastructure for Network Computing* - *Middleware* para Computação Voluntária.
CAPC	Medida da capacidade computacional de um computador.
Dist.	Distribuição.
DHT	*Distributed Hash Table* - Tabela de *Hash* Distribuída.
FIFO	*First-in-First-out* - Sistema de Fila.
FLOPS	Medida de operações de ponto flutuante por segundo em um processador.
Gbps	Gigabits por segundo.
GPU	Unidade de Processamento Gráfico.
HDFS	*Hadoop Distributed File System* - Sistema de Arquivos Distribuídos do Hadoop.
LISP	*LISt Processing* - Um tipo de linguagem de programação funcional.
Mbps	Megabits por segundo.
Md	Mediana - Medida de tendência central.
MPI	*Message Passing Interface* - Interface de programação paralela através de troca de mensagens.
MRSG	*MapReduce over SimGrid* - *MapReduce* criado sobre o SimGrid.
P2P	*Peer-to-Peer* - Arquitetura de rede onde cada computador compartilha seus recursos com todos.
RAM	Memória de acesso randômico em um computador.
SIMGRID	Simulador de ambientes de *Grids*.
Tam.	Tamanho.
SAGA	***S**imple **A**PI for **G**rid **A**pplications* - API para aplicações em *Grids*.
ZETTAFLOPS	10^{21} *flops*.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

RESUMO

O surgimento de volumes de dados na ordem de *petabytes* cria a necessidade de desenvolver-se novas soluções que viabilizem o tratamento dos dados através do uso de sistemas de computação intensiva, como o *MapReduce*. O *MapReduce* é um *framework* de programação que apresenta duas funções: uma de mapeamento, chamada *Map*, e outra de redução, chamada *Reduce*, aplicadas a uma determinada entrada de dados. Este modelo de programação é utilizado geralmente em grandes *clusters* e suas tarefas *Map* ou *Reduce* são normalmente independentes entre si. O programador é abstraído do processo de paralelização como divisão e distribuição de dados, tolerância a falhas, persistência de dados e distribuição de tarefas. A motivação deste trabalho é aplicar o modelo de computação intensiva do *MapReduce* com grande volume de dados para uso em ambientes *desktop grid*. O objetivo então é investigar os algoritmos do *MapReduce* para adequar a computação intensiva aos ambientes heterogêneos. O trabalho endereça o problema da heterogeneidade de recursos, não tratando neste momento a volatilidade das máquinas. Devido às deficiências encontradas no *MapReduce* em ambientes heterogêneos foi proposto o *MR-A++*, que é um *MapReduce* com algoritmos adequados ao ambiente heterogêneo. O modelo do *MR-A++* cria uma tarefa de medição para coletar informações, antes de ocorrer a distribuição dos dados. Assim, as informações serão utilizadas para gerenciar o sistema. Para avaliar os algoritmos alterados foi empregada a Análise 2^k Fatorial e foram executadas simulações com o simulador MRSG. O simulador MRSG foi construído para o estudo de ambientes (homogêneos e heterogêneos) em larga escala com uso do *MapReduce*. O pequeno atraso introduzido na fase de *setup* da computação é compensado com a adequação do ambiente heterogêneo à capacidade computacional das máquinas, com ganhos de redução de tempo de execução dos *jobs* superiores a 70 % em alguns casos.

Palavras-chave: Sistemas Distribuídos, MapReduce, Desktop Grid.

Adequacy of Intensive Data Computing to Desktop Grid Environment with using of MapReduce

ABSTRACT

The emergence of data volumes in the order of petabytes creates the need to develop new solutions that make possible the processing of data through the use of intensive computing systems, as MapReduce. MapReduce is a programming framework that has two functions: one called Map, mapping, and another reducing called Reduce, applied to a particular data entry. This programming model is used primarily in large clusters and their tasks are normally independent. The programmer is abstracted from the parallelization process such as division and data distribution, fault tolerance, data persistence and distribution of tasks. The motivation of this work is to apply the intensive computation model of MapReduce with large volume of data in desktop grid environments. The goal then is to investigate the intensive computing in heterogeneous environments with use MapReduce model. First the problem of resource heterogeneity is solved, not treating the moment of the volatility. Due to deficiencies of the MapReduce model in heterogeneous environments it was proposed the *MR-A++*; a MapReduce with algorithms adequated to heterogeneous environments. The MR-A++ model creates a training task to gather information prior to the distribution of data. Therefore the information will be used to manager the system. To evaluate the algorithms change it was employed a 2^k Factorial analysis and simulations with the simulant MRSG built for the study of environments (homogeneous and heterogeneous) large-scale use of MapReduce. The small delay introduced in phase of setup of computing compensates with the adequacy of heterogeneous environment to computational capacity of the machines, with gains in the run-time reduction of jobs exceeding 70% in some cases.

Keywords: Distributed Systems, MapReduce, Desktop Grid .

1 INTRODUÇÃO

1.1 Visão Geral

A quantidade de informação produzida pela humanidade está na casa de *Exabytes* de informações (HILBERT; LOPEZ; VASQUEZ, 2010). Este grande volume de dados exige o desenvolvimento de arquiteturas e sistemas que viabilizem o uso de aplicações intensivas em dados.

Diferentes soluções foram propostas para o tratamento de dados em sistemas distribuídos que resultaram no surgimento de arquiteturas como *clusters, grids, desktop grids* e *clouds*, assim como bibliotecas e linguagens de programação tais como: Java, MPI, Globus, #C, *.Net*, etc.

Porém, a computação de grandes volumes de dados associados a baixos tempos de resposta, ainda é uma questão em pauta na comunidade científica (ISARD et al., 2007; MACKEY et al., 2008; WILDE et al., 2009; MICELI et al., 2009; MORETTI et al., 2010).

O *MapReduce* é um modelo de computação intensiva em dados proposto em 2004, por Jeffrey Dean e Sanjay Ghemawato, para simplificar a construção de índices reversos na análise de pesquisas na *web* (DEAN; GHEMAWAT, 2004). Os algoritmos utilizados no *MapReduce* permitem o tratamento de dados intensivos em um sistema de arquivos distribuído, a abstração do paralelismo de tarefas e o controle de falhas.

Grandes empresas como, por exemplo, Yahoo, Facebook, Amazon e IBM utilizam o modelo do *MapReduce* como ferramenta para aplicações de *Cloud Computing*, através da implementação do Hadoop, um código *open source* do *MapReduce*, produzido pela Apache Software Foundation. Outras propostas surgiram mais tarde com enfoque diferente, como por exemplo: Dryad (ISARD et al., 2007), Phoenix (RANGER et al., 2007) e Pig (OLSTON et al., 2008).

O conceito computacional do *MapReduce* para a manipulação de dados é simples, entretanto, com entradas da ordem de *Petabytes* existe uma grande complexidade na distribuição e no gerenciamento dos dados. A entrada de dados é processada como tarefas menores, através de uma programação de linguagem interpretada com a habilidade de rodar uma consulta *ad hoc* sobre um conjunto inteiro de dados (WHITE, 2009).

As tarefas são distribuídas através de centenas ou até milhares de computadores. O processamento ocorre com intervalos de tempos de resposta cada vez menores. O *MapReduce* é um *framework* de programação que abstrai dos programadores a complexidade de paralelização e o gerenciamento de dados e tarefas (DEAN; GHEMAWAT, 2010).

O *MapReduce* fornece um mecanismo inspirado em primitivas *Map* e *Reduce* existentes em linguagens funcionais de alto nível como LISP e Haskell. Os dados da entrada são tratados e transformados em *tuplas* (chave,valor).

As aplicações devem ser escritas em forma de duas funções, uma de *Map* e outra de

Reduce. As aplicações mais comuns do *MapReduce* são algoritmos para o processamento de grafos em larga escala, processamento de texto, mineração de dados para aprendizagem de máquina, tradução automática, entre outras (DEAN; GHEMAWAT, 2010).

O *MapReduce* é utilizado preferencialmente em grandes *clusters*, com redes de baixa latência e com unidades de armazenamentos locais de baixo custo (DEAN; GHEMAWAT, 2004). A entrada de dados é particionada previamente em um sistema de arquivos distribuídos. As tarefas são enviadas para serem executadas perto dos nós. Desta forma, privilegia-se a execução de tarefas sobre dados locais, sem a necessidade de transferir grandes volumes de dados em tempo de processamento.

Os dados da entrada são processados pela tarefa *Map* que produz dados intermediários para serem consumidos pela tarefa *Reduce* na fase seguinte. O modelo de execução cria uma barreira computacional, que permite sincronizar a execução de tarefas entre produtor e consumidor. Uma tarefa *Reduce* não inicia seu processamento enquanto todas as tarefas *Map* não terminarem.

A pesquisa de modelos de computação intensivos em dados é relevante e pertinente. Diversos grupos de pesquisa de instituições como Carnegie Mellon University, UC Berkeley, Stanford University, Columbia University, Université de Lyon, UFRGS, entre outras, têm apresentado trabalhos científicos considerando o uso deste *framework* para o tratamento de grandes volumes de dados, como será apresentado na seção 2.5.

1.2 *MapReduce* em Plataformas Heterogêneas

A computação em *grid* utiliza diversos computadores distribuídos através de várias organizações virtuais. Os recursos computacionais são heterogêneos e aplicam trocas de mensagens para a computação de alto desempenho (BUYYA; BAKER; LAFORENZA, 2000). A abordagem funciona bem para a computação intensiva no domínio de uma rede local, porém pode tornar-se um problema se for necessário o acesso a grandes volumes de dados através das demais organizações.

As tarefas de *Map* ou *Reduce* são do tipo *Bag-of-tasks*, ou seja, sem dependências entre si. Tarefas independentes facilitam o processamento das informações em *grid*. Assim os programadores não precisam se preocupar com a ordem da execução das tarefas. Eventuais falhas podem ser recuperadas facilmente com o escalonamento de uma nova tarefa para os nós que estejam livres.

Por mais de uma década, plataformas de Computação Voluntária têm sido um dos maiores e mais poderosos sistemas de computação distribuída do planeta, oferecendo um alto retorno sobre o investimento para aplicações de uma ampla variedade de domínios científicos (incluindo a biologia computacional, a previsão do clima e física das altas energias) (KONDO et al., 2009).

As *grids* de computação voluntária, como *desktop grids*, são formadas por máquinas *desktops* que utilizam ciclos ociosos de máquina para o processamento de tarefas (ANDERSON; MCLEOD, 2007). Em *grids* formadas por nós dispersos em vários locais a heterogeneidade é muito maior do que aquelas formadas por organizações virtuais.

Várias implementações *desktop grid* utilizam recursos dispersos como, por exemplo, o BOINC (*Berkeley Open Infrastructure for Network Computing*) que é um *middleware* para o uso de recursos computacionais de forma voluntária (ANDERSON, 2004). Dentre os seus diversos projetos, o SETI@home funciona com cerca de 3 milhões de computadores, com capacidade de processamento de mais de 109,5 Zettaflops (BERKELEY, 2012).

As *desktop grids* caracterizam-se por serem largamente distribuídas, heterogêneas e de alta volatilidade (ANDERSON, 2004). Entretanto, a arquitetura de *desktop grid* não é voltada para aplicações intensivas em dados. O modelo do BOINC, por exemplo, é puramente mestre-escravo. No *MapReduce* o gerenciamento é mestre-escravo e os nós seguem o modelo P2P para a execução de tarefas.

Portanto, seriam necessários adaptações, para se usufruir deste poder computacional (LAM, 2011). Estudos feitos por Lin (LIN et al., 2009), com o projeto *Moon*, e Tang (TANG et al., 2010), com o *Bitdew*, propõem aplicar o modelo do *MapReduce* em ambientes heterogêneos como *desktop grid*.

1.3 Motivação

O processamento de diversas aplicações tais como simulações de prospecção de óleos em plataformas petrolíferas, de previsão do clima ou catástrofes e de pesquisas biomédicas levam ao tratamento de quantidades de dados cada vez maiores. Portanto, as aplicações intensivas em dados conduzem à necessidade de encontrar-se novas soluções para o processamento de dados.

O *MapReduce* é amplamente utilizado em grandes *data centers*. A aplicação de um modelo intensivo em dados em *grids*, como as *desktop grids*, pode ser viável desde que sejam encontradas respostas, principalmente, para a heterogeneidade de recursos. O uso do *MapReduce* em ambiente heterogêneo tem baixo desempenho se comparado com o ambiente homogêneo. Assim, para adotar-se o *MapReduce*, em ambientes *desktop grid*, é necessário primeiro adaptar seus algoritmos a ambientes heterogêneos.

Os trabalhos apresentados por Zaharia, Lin e Xie (ZAHARIA et al., 2008; LIN et al., 2009; XIE et al., 2010) tratam de alguns problemas relacionados com a heterogeneidade, mas não propõem uma solução adequada aos algoritmos do *MapReduce* para ambientes como *desktop grid*, como será apresentado na seção 2.5.2.

Zaharia (ZAHARIA et al., 2008) propõe a alteração do algoritmo de lançamento de tarefas especulativas. O problema do *MapReduce* em ambientes heterogêneos é a identificação de máquinas lentas de forma errada. A identificação errada causa a criação de tarefas especulativas que consomem recursos desnecessariamente e conduzem a desempenhos pífios. Entretanto, este problema não foi tratado pelo autor.

Lin (LIN et al., 2009) propõe um ambiente híbrido para controlar a heterogeneidade de recursos via replicação de dados. Porém a proposta não apresenta outros mecanismos que possam controlar o lançamento de tarefas conforme a capacidade computacional de cada máquina e não avalia o custo das cópias de dados em redes lentas.

Xie (XIE et al., 2010) propõe a divisão de dados em ambientes heterogêneos conforme a capacidade das máquinas. A proposta trata o problema de forma parcial, pois preocupa-se somente com as tarefas de *Map*, elimina o mecanismo de tolerância a falhas e não considera o desbalanceamento de carga na fase de *Reduce* gerado por sua abordagem. Outro problema é que a implementação em apenas cinco máquinas não reflete o ambiente de uso do *MapReduce*.

Tang (TANG et al., 2010) propõe o uso de um mecanismo P2P com o *Bitdew*, porém esta abordagem não segue o modelo de execução proposto para o *MapReduce* e os dados são enviados em tempo de processamento para as máquinas. As máquinas lentas não são tão facilmente detectadas e não é garantida a comunicação entre os *workers*.

Gufler (GUFLER et al., 2011) apresenta um conjunto de algoritmos para dividir os dados intermediários do *Map* aplicados somente a ambientes homogêneos. A abordagem

com o uso de granularidade fina das chaves intermediárias ao ser adequada ao ambiente heterogêneo, como em *desktop grid*, pode possibilitar o melhor desempenho dos *jobs*. Esta abordagem será explorada no *MR-A++* com bons resultados.

O *MR-A++* (***Map****R****educe*** com **A**lgoritmos **A**daptados ao **A**mbiente **H**eterogêneo), proposto neste trabalho, objetiva endereçar os principais problemas existentes na execução deste *framework* sobre ambientes heterogêneos como *desktop grid*. Assim, a solução desenvolvida poderá ser um facilitador para o uso de aplicações intensivas em dados sobre ambientes *desktop grid* em larga escala.

A proposta do *MR-A++* adéqua todos os algoritmos do *MapReduce* para o ambiente heterogêneo, como os algoritmos da divisão de dados e do lançamento de tarefas, e cria um mecanismo que permite escolher uma máquina mais rápida que a corrente para a execução de uma tarefa.

O *MR-A++* controla o progresso das tarefas conforme a capacidade computacional e possibilita um menor erro na identificação de máquinas lentas durante o processamento. Assim, a abordagem permite o melhor uso dos recursos disponíveis em um ambiente *desktop grid*. No entanto, as abordagens adotadas para o endereçamento da heterogeneidade não tratam da volatilidade de recursos que será contemplada em trabalhos futuros.

1.4 Contribuições

O *MR-A++* é um *MapReduce* concebido para ambientes heterogêneos que explora as caraterísticas dos recursos computacionais destes ambientes, adaptando os mecanismos do *MapReduce* à heterogeneidade.

O trabalho apresenta uma extensa bibliografia sobre o *MapReduce*. Os sistemas conhecidos até o momento para o tratamento de aplicações intensivas em dados são avaliados. As soluções que tratam de ambientes heterogêneos são relacionadas e comparadas. Os tipos de simuladores conhecidos e os algoritmos envolvidos no *MapReduce* são analisados.

Os problemas dos simuladores apontados na seção 2.8 e a necessidade de avaliação das abordagens em larga escala motivaram a criação de um novo simulador o MRSG, que foi desenvolvido pelo grupo de estudos em MapReduce do GPPD/UFRGS com a colaboração e supervisão deste autor.

Os resultados da avaliação de desempenho demonstrados neste trabalho, confirmam a eficiência do *MR-A++* e indicam que a proposta é adequada ao uso em ambientes heterogêneos e em *desktop grids*

1.5 Cooperação Internacional

Este trabalho é co-orientado pela Profª. Drª. Luciana Arantes, da *University of Pierre et Marie Currie* Paris VI com cooperação do *Laboratoire d'Informatique* da Universidade de Paris VI - França. Esta parceria foi iniciada, em junho de 2009, através de visitas técnicas feitas ao grupo de pesquisas do GPPD, na UFRGS, pela Profª. Drª. Luciana Arantes e ao *Laboratoire d'Informatique*, na França, pelo Prof. Dr. Cláudio Geyer.

Esta cooperação foi consolidada em dezembro de 2009, com a presença do aluno Julio Anjos e da Profª. Drª. Luciana Arantes, quando foram elaborados o escopo do projeto e as metas a alcançar.

1.6 Organização

Este trabalho está organizado da seguinte maneira. No capítulo 2 é apresentado o estado da arte em computação de aplicações intensiva em dados, compara-se o modelo do *MapReduce* com outras formas de tratamento de dados e relaciona-se os principais modelos desenvolvidos para tratamento de dados em larga escala.

A seção 2.1 apresenta uma visão geral do funcionamento do *MapReduce*. Uma comparação é apresentada entre o *framework* e os modelos tradicionais de banco de dados . O modelo do Hadoop é mostrado na seção 2.2, a sua implementação é considerada uma das mais completas em relação ao *MapReduce* original. A seção 2.3 detalha o funcionamento do modelo através dos seus principais algoritmos. Os algoritmos de divisão de dados e divisão de tarefas são apresentados conforme a implementação feita sobre o Hadoop.

Na seção 2.4 apresenta-se as implementações do *MapReduce* para memória compartilhada e GPUs. Na seção 2.5 são apresentados estudos dos problemas encontrados para o uso do *MapReduce* em aplicações científicas com grande quantidade de cálculos. São discutidos os trabalhos relacionados com as implementações em memória distribuída do modelo do *MapReduce*. Os principais modelos implementados para memória distribuída com o Hadoop são comparados na seção 2.6.

Os estudos dos modelos de distribuição estatística de dados e de classes de distribuição são relacionados na seção 2.7. Estes estudos são fundamentais para compreensão de como o modelo do *MapReduce* comporta-se através de modelos estatísticos. Os modelos estatísticos de distribuição permitem prever o comportamento das execuções de tarefas em um determinado *cluster* e assim dinamicamente ajustar o escalonador a este comportamento.

Na seção 2.8 são descritos os simuladores criados por outros grupos de pesquisa até o momento para o estudo dos algoritmos do *MapReduce*. Na seção 2.9 são discutidos os principais problemas com o modelo, mecanismos de tolerância a falhas, uso de ambientes voláteis e os desafios existentes nas implementações atuais. Outros modelos de sistemas de memória distribuída criados para o tratamento da computação de aplicações intensivas em dados são discutidos na seção 2.10.

O capítulo 3 apresenta o modelo do *MR-A++*, criado para a adequar os algoritmos do *MapReduce* ao ambiente heterogêneo. As abordagens propostas são discutidas, assim como, a arquitetura e o detalhamento dos algoritmos criados neste trabalho.

A seção 3.1 detalha os objetivos principais e secundários deste trabalho. Na seção 3.2 são descritos as abordagens adotadas e os modelos criados para a solução do problema da heterogeneidade dos nós. As abordagens propostas para endereçar os problemas do *MapReduce* estão relacionadas com a distribuição de dados conforme a capacidade computacional das máquinas, criação de agrupamentos e a distribuição de tarefas. A arquitetura modular do *MR-A++* é apresentada na seção 3.3.

Os novos algoritmos propostos para endereçar os problemas da heterogeneidade são detalhados nas seções 3.4 e 3.5. A primeira, seção 3.4, apresenta os algoritmos utilizados para a distribuição de dados em ambiente heterogêneo tanto para o *Map* como para o *Reduce*. A segunda, seção 3.5, apresenta os algoritmos que controlam a execução e distribuição de tarefas.

O capítulo 4 apresenta a avaliação dos resultados obtidos. A metodologia aplicada nos experimentos é apresentada na seção 4.1. A arquitetura utilizada nos testes e a configuração dos experimentos estão apresentadas seção 4.2. Um detalhamento do simulador MRSG (*MapReduce over SimGrid*), desenvolvido pelo grupo de pesquisas em *MapReduce* do GPPD/UFRGS, é apresentado nesta seção, assim como, os experimentos feitos

na *Grid 5000* para sua validação.

Uma análise 2^k Fatorial é apresentada na seção 4.4, com o objetivo de determinar o comportamento dos algoritmos no ambiente de testes. Diversos comparativos de experimentos são apresentados na seção 4.5, assim como uma análise dos custos relacionados com a tarefa de medição, utilizada para coletar informações para o gerenciamento da distribuição de dados e de tarefas. Finalmente no capítulo 5 são apresentados conclusões e trabalhos futuros.

2 COMPUTAÇÃO INTENSIVA EM DADOS

A computação de aplicações intensivas em dados é um tema recente na comunidade científica. Neste capítulo serão apresentados os principais modelos conhecidos até o momento com suas características e propriedades, assim como, serão relacionados os principais desafios para a adequação do modelo na presença de ambientes heterogêneos.

2.1 Visão Geral do *MapReduce*

A computação de grandes volumes de dados e o uso eficiente de recursos computacionais, associados a baixos tempos de resposta, ainda são uma questão em pauta na comunidade científica (ISARD et al., 2007; MACKEY et al., 2008; WILDE et al., 2009; MICELI et al., 2009; MORETTI et al., 2010).

O *MapReduce* é um *framework* de programação que abstrai a complexidade do paralelismo das aplicações. O modelo trata a entrada de dados como uma função de *tuplas* (chave,valor). O conceito computacional simples da manipulação de dados esconde do programador a grande complexidade da distribuição e do gerenciamento de dados. A complexidade se deve aos dados serem de grandes volumes, por estarem espalhados através de centenas ou milhares de máquinas e pela necessidade da computação ser realizada com intervalos de tempos cada vez menores (DEAN; GHEMAWAT, 2004).

O modelo do *MapReduce* é constituído de uma máquina mestre que gerencia as demais máquinas escravas. A Figura 2.1, adaptada de (WHITE, 2009), apresenta o modelo do fluxo de dados do *MapReduce* com três fases distintas.

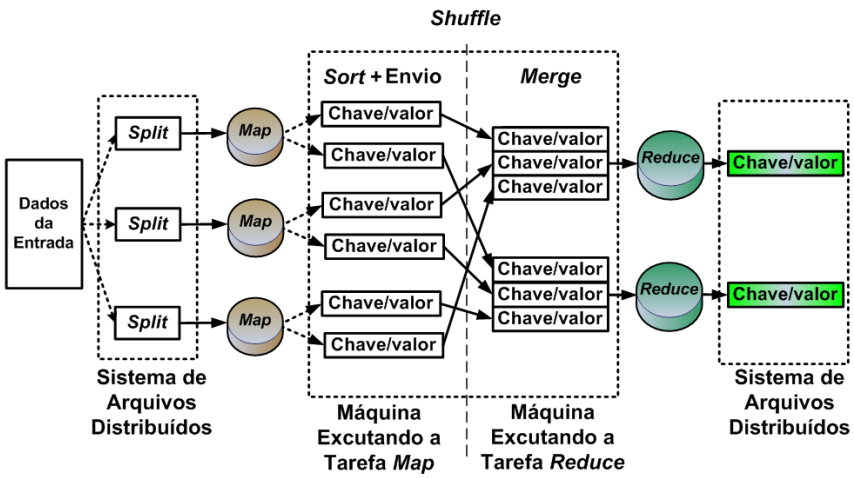


Figura 2.1: Modelo do fluxo de dados no *MapReduce*

As fases de *Map* e *Reduce* são acessíveis ao programador. A terceira, chamada de *Shuffle*, é criada pelo sistema durante sua execução. A tarefa *Map* executa uma função de mapeamento e gera dados intermediários da entrada que são armazenados na máquina local. A tarefa *Reduce* recebe estes dados intermediários e executa uma função de redução que agrupa as chaves iguais.

O *Shuffle* é constituído de dois processos: um na máquina que processa a tarefa *Map* onde executa-se um *sort* (ordenação de chaves) e serialização dos dados. Após os dados intermediários são enviados para a máquina que executará a tarefa de *Reduce*, nesta máquina é aplicado um *merge* nos dados recebidos das tarefas *Map* com o objetivo de agrupar adequadamente as chaves intermediárias antes de se executar a função *Reduce* (WHITE, 2009).

Uma grande entrada de dados é dividida em pedaços menores chamados de *chunks*, normalmente de 64 MB. O tamanho é um parâmetro fornecido pelo programador, embora o valor dependa da tecnologia dos discos e da taxa de transferência de dados da memória para o disco. Em discos de baixo custo, por exemplo, o tempo de busca de um bloco de dados é de $10ms$ (WHITE, 2009).

Um cliente submete um *job* com as definições das funções *Map* e *Reduce*. O *job* é dividido em várias tarefas ou *tasks* (*Map* e *Reduce*) no mestre. O mestre designa as tarefas às máquinas que irão executar cada etapa do processamento. Os dados, então, são divididos e após distribuídos em um sistema de arquivos que mantém um mecanismo de replicação dos dados. Uma tarefa *Map* transforma a entrada de dados em *tuplas* (chave,valor) que são armazenadas no disco local.

As chaves iguais de todas as tarefas *Map* são transferidas, no processo de *Shuffle*, para uma mesma máquina processar a tarefa *Reduce*. Uma função de redução é aplicada sobre estes dados intermediários e é emitido um novo resultado. O resultado em formato de *tuplas* é então armazenado no sistema de arquivos distribuído para ser disponibilizado ao cliente que submeteu o *job*. Uma função *Hash* é aplicada aos dados intermediários produzidos no *Map*, para determinar as chaves que irão formar as tarefas *Reduce* que serão executadas.

A arquitetura de comunicação no *MapReduce* é baseada em um modelo *mestre/escravo*, com um escalonador e diversos mecanismos interligados como *split* dos dados da entrada, sistema de arquivos distribuídos, *sort*/*merge* de dados e sistemas de monitoramento e gerenciamento.

O *MapReduce* é constituído de vários algoritmos, entre eles o algoritmo de distribuição de dados e de tarefas. O algoritmo de distribuição de dados executa a divisão e a distribuição dos dados sobre um sistema de arquivos distribuídos. O algoritmo de distribuição de tarefas, recebe o *job* e cria tarefas *Map* e *Reduce* para serem distribuídas aos nós que executarão o processamento. Os algoritmos de distribuição de dados e de tarefas são a base para a computação paralela no *MapReduce*.

A implementação do *MapReduce* particiona cada tarefa em um conjunto de subtarefas menores de tamanhos iguais, com características *Bag-of-Tasks* (tarefas sem dependências). Diferente de outros modelos em que os dados são transferidos para serem processados em um programa, no *MapReduce* a programação é levada aos dados.

O funcionamento do *MapReduce* pode ser comparado com uma linguagem de consulta SQL, em uma abstração em alto nível. A Figura 2.2, adaptada de (CHEN; SCHLOSSER, 2008), exemplifica uma aplicação para a contagem de palavras de uma determinada entrada de dados. Os dados da entrada são selecionados por uma tarefa *Map* que gera um conjunto de *tuplas* (chave_in,valor_in). As *tuplas* são então ordenadas para, na saída, se-

rem agrupadas em *tuplas* (chave_out, valor_out) por uma tarefa *Reduce*. No exemplo, a contagem dos valores das ocorrências das chaves iguais são somadas na tarefa de *Reduce*.

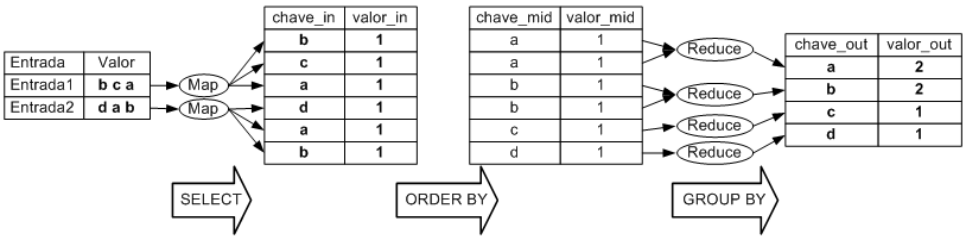


Figura 2.2: Estrutura do modelo de programação do *MapReduce*

Assim abstraindo-se as funções de *Map* e *Reduce* como em uma estrutura de banco de dados é possível ligeiramente associar este fluxo a uma consulta com a linguagem SQL. A representação da consulta ao banco de dados é exemplificada através das funções representadas nos comandos 2.1 e 2.2:

SELECT chave_in **FROM** entrada **ORDER BY** chave_in **CREC** (2.1)

SELECT COUNT(*), chave_mid **AS** chave_out **FROM** chave_mid **GROUP BY** chave_mid **CREC** (2.2)

Entretanto, diferentemente de uma consulta em banco de dados, o *MapReduce* executa esta busca nos dados em centenas ou milhares de máquinas simultaneamente, abstraindo o paralelismo das tarefas, através da declaração de duas funções, como apresentado no exemplo para a contagem de palavras (*Word Count*) no pseudocódigo 2.1.1, adaptado de (DEAN; GHEMAWAT, 2010).

Pseudocódigo 2.1.1 *Word Count*

```
map (line_number, text):
word_list[] = split (text)
for each word in word_list: do
    emit (word, 1)
reduce (word, values[]):
word_count = 0
for each v in values: do
    word_count+ = v
emit (word, word_count)
```

O objetivo do *Word Count* é contar a quantidade de ocorrências de cada palavra em um documento. Cada chamada da função *Map* recebe como *valor* uma linha de texto do documento, e como *chave* o número desta linha. Para cada palavra encontrada na linha recebida, a função emite um par (chave,valor), onde a chave é a palavra em si, e o valor é a constante 1 (um). A função *Reduce*, então, recebe como entrada uma palavra (*chave*), e um iterador para todos os valores emitidos pela função *Map*, associados com a palavra em questão. No final é emitido um par (chave,valor), contendo cada palavra com o total de sua ocorrência (DEAN; GHEMAWAT, 2010).

A Figura 2.3 exemplifica o funcionamento do *MapReduce* para uma operação *Word-Count* (contagem de palavras). As funções de *Map* e *Reduce* são aplicadas sobre um grande volume de dados, *e.g.* 1 TB de dados. Os dados contêm as vendas de carros de uma concessionária. O objetivo é encontrar o número de carros existentes no estoque.

Após um *job* ter sido submetido por um programador, uma tarefa *Map* processa as linhas da entrada de dados e emite um valor 1 a cada nome de carro não vendido. Na fase de *Shuffle* os dados intermediários produzidos no *Map* são ordenados por um *sort* e após, as chaves intermediárias são enviadas para serem consumida pelas tarefas *Reduce* na próxima fase. Finalmente, após as máquinas receberem todos os dados intermediários executa-se um *merge* para preparar os dados para a execução da tarefa *Reduce*. A tarefa *Reduce* soma os valores das chaves semelhantes e emite um resultado .

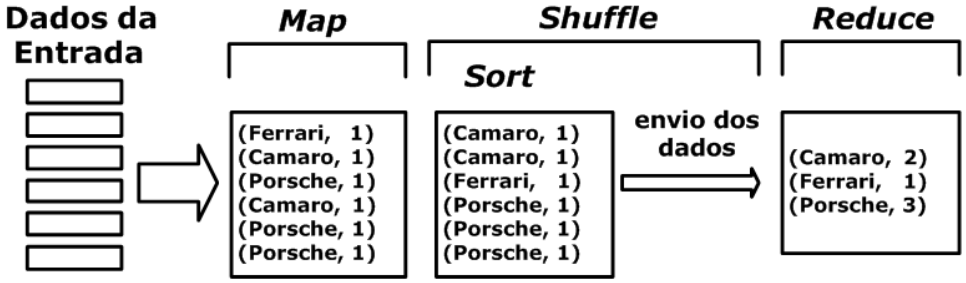


Figura 2.3: Exemplo do fluxo de dados do *MapReduce*

O *MapReduce* relaciona-se com um banco de dados pela forma de consulta. Porém diferentemente de uma consulta SQL, o *MapReduce* tem uma escalabilidade linear. Isto porque, podem ser feitas diversas consultas simultaneamente distribuídas entre várias tarefas. Os dados podem não ter qualquer estrutura interna ou serem estruturados ou semi-estruturados. Arquivos longos de todos os tipos são bem aceitos para serem analisados nesta estrutura, sendo desmembrados em tamanhos menores e manipulados por diversas máquinas (WHITE, 2009).

2.2 Hadoop: Uma Implementação do *MapReduce*

O Google criou uma implementação proprietária do *MapReduce*. O modelo de computação no entanto foi difundido para a comunidade científica como uma nova alternativa para o tratamento de dados intensivos em grandes *clusters*.

O Hadoop é uma implementação *open source* do *MapReduce*, proposta pela Apache Software Foundation, a partir de estudos de documentos produzidos pelo Google. A implementação do Haddop é utilizada por empresas como Yahoo, Amazon, FaceBook e IBM (VENNER, 2009). A maioria dos trabalhos e inclusive este são baseados nos algoritmos utilizados na implementação do Hadoop que é considerada a implementação *open source* mais fiel do *MapReduce* (DEAN; GHEMAWAT, 2010).

A arquitetura do Hadoop possui vários módulos que trabalham independentes mas de forma integrada para reproduzir os algoritmos do *MapReduce*. O sistema de arquivos distribuídos HDFS, o *JobTracker* e o *TaskTracker* são o coração do Hadoop. Alguns dos algoritmos foram estudados baseando-se em (DEAN; GHEMAWAT, 2004, 2010), (WHITE, 2009) e (VENNER, 2009) e serão apresentados na seção 2.3.

O *JobTracker* gerencia o sistema e distribui as tarefas, e o *TaskTracker* executa tarefas recebidas e as transferências de dados. Assim como no *MapReduce*, o Hadoop tem uma arquitetura cliente/servidor para o gerenciamento de tarefas. A comunicação entre as máquinas escravas segue uma arquitetura P2P independente e tem a função de efetuar cópias de dados intermediários ou réplicas de dados remotos (WHITE, 2009).

O HDFS (*Hadoop Distributed File System*) é um sistema de arquivos distribuído. A

estrutura de blocos tem um tamanho fixo. No HDFS, um algoritmo de distribuição de dados é responsável por manter um mecanismo de tolerância a falhas, para evitar perdas e o reparticionamento de dados em tempo de execução. A quantidade de réplicas é um parâmetro de configuração fornecido pelo programador. Os dados são replicados observando-se as distâncias entre as máquinas. Uma máquina recebe o dado e faz uma segunda réplica para uma máquina em outro *rack* no mesmo *data center*. Então, a segunda máquina replica os dados para uma terceira no mesmo *rack* (WHITE, 2009).

A distância entre as máquinas recebe pesos que serão utilizados para a distribuição dos dados. Para um processo que executa na mesma máquina a distância é zero; para diferentes máquinas no mesmo *rack* a distância é dois; máquinas em diferentes *racks* no mesmo *data center* a distância é quatro e, finalmente, em máquinas de diferentes *data centers* a distância é seis.

Existe uma forte ligação entre o sistema de arquivos distribuído e a implementação do *MapReduce*. Isto ocorre pois as máquinas escravas são também servidores de dados. Portanto, quando o escalonador do *MapReduce* atribui uma tarefa de mapeamento, ele tenta fazê-lo para uma máquina que já possua uma réplica dos dados locais, a fim de evitar a transferência de informações pela rede durante a execução (DEAN; GHEMAWAT, 2010).

As comunicações ocorrem por um mecanismo de troca de mensagens chamado *Heartbeat*. No período de cada 5 segundos o *TaskTracker* envia uma mensagem com informações do estado de sua operação para o *JobTracker*. Este intervalo de mensagens é estabelecido pelo sistema e automaticamente após os primeiros 100 nós, a cada 100 novos nós acrescenta-se mais 1 segundo a este tempo. Assim, o lançamento de tarefas em um *job* no *MapReduce* ocorre em forma de ondas, como demonstra o diagrama de Gantt na Figura 2.4. O objetivo é evitar a sobrecarga na comunicação e nas transferências de dados para o mestre (DEAN; GHEMAWAT, 2010).

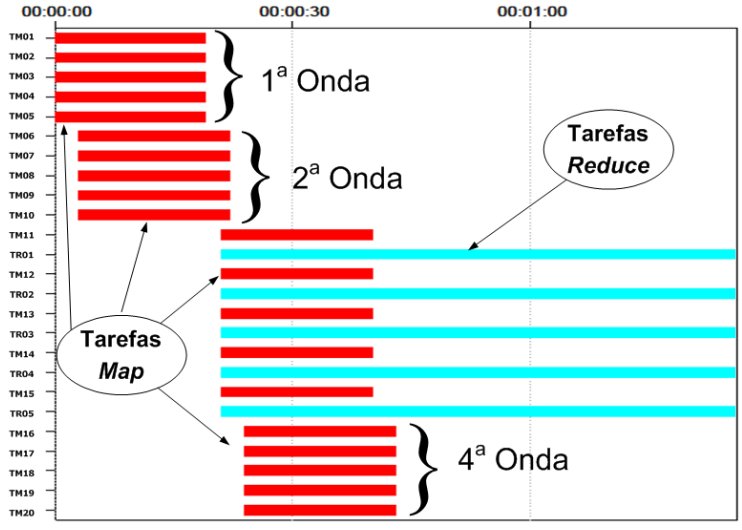


Figura 2.4: Exemplo de execução de um *job* no *MapReduce*

Os algoritmos apresentados na seção 2.3 foram organizados baseados nas principais tarefas necessárias ao funcionamento do *MapReduce*. O processo da divisão de dados da entrada (*split*) é controlado pelo sistema de arquivos HDFS. O gerenciamento de tarefas é realizado pelo *JobTracker* e pelo *TaskTrackers* ocorre sua execução. O progresso da execução de tarefas é controlado pelo *TaskRunner*.

2.3 Algoritmos do *MapReduce*

O algoritmo de distribuição de tarefas possui duas versões, um para tarefas *Map* e outro para tarefas *Reduce*. Réplicas das tarefas são geradas conforme uma quantidade definida pelo programador.

O escalonador cria as tarefas e as réplicas associadas as tarefas. Porém, ele distribui uma tarefa de cada réplica por vez. As demais tarefas somente serão lançadas para outra máquina livre no caso de se constatar alguma falha ou através da análise do tempo de progresso de uma tarefa.

Um algoritmo de distribuição de dados faz o *split* dos dados da entrada para criar os *chunks*. Um mecanismo de replicação faz uma cópia dos dados. Quando as tarefas forem executadas, os dados normalmente estarão na máquina local, portanto, sem perdas associadas com a transferência de dados, em tempo de execução.

Neste tipo de aplicação, a política sobre os dados da entrada força a escrita do dado uma única vez, executando após diversas leituras e manipulações sobre os mesmos dados, política também conhecida como (*Write Once, Read Many Times*) (WHITE, 2009).

Dois mecanismos são utilizados para otimizar a execução das tarefas: o lançamento remoto e o especulativo. O lançamento remoto ocorre quando uma máquina encontra-se livre mas não tem uma réplica dos dados para processar no disco local. Assim a máquina faz uma cópia dos dados em tempo de execução de uma outra máquina que contenha a réplica do *chunk*. O lançamento especulativo ocorre após não existirem tarefas pendentes para serem lançadas. A tarefa especulativa é uma tarefa criada a partir da constatação de uma possível falha, originada da análise do tempo de progresso de uma tarefa (DEAN; GHEMAWAT, 2004).

Um processo denominado $Job\_inProgress$ controla o progresso de execução das tarefas. Se uma máquina está executando uma tarefa por mais de 60 segundos e seu progresso é igual ou inferior à 20% do tempo médio total das execuções das tarefas de cada fase, a máquina é marcada como *straggler* (máquina lenta). Assim, as máquinas são classificadas de acordo com seu progresso de execução em máquinas normais, máquina que falham e máquinas *stragglers*.

Após não haverem mais tarefas pendentes, o escalonador lança uma tarefa especulativa para uma máquina livre que contenha uma réplica dos dados, com o objetivo de aumentar o desempenho das execuções das máquinas lentas. Quando qualquer uma das máquina (a *straggler* ou a especulativa), enviar uma mensagem de *heartbeat* informando a conclusão da tarefa, a outra será finalizada (VENNER, 2009; WHITE, 2009; DEAN; GHEMAWAT, 2010).

Uma tarefa *Map* é associada a cada *chunk* no momento em que ocorre a divisão dos dados no sistema de arquivos distribuídos durante o processo de *split*. Então, as tarefas são colocadas no mestre em uma fila FIFO para serem processadas pelas máquinas. As máquinas escravas enviam uma mensagem de *heartbeat* solicitando uma tarefa ao mestre. A tarefa de *Map* é enviada para a máquina que contenha uma réplica do *chunk*. Caso o *chunk* não esteja na máquina local, o escalonador verifica qual máquina contém uma réplica do *chunk* e envia a tarefa a ela (DEAN; GHEMAWAT, 2004).

Na fase de *Map* se a máquina tornar-se livre, as tarefas são escolhidas na seguinte ordem: tarefa que falhou, tarefa pendente com *chunk* local, tarefa pendente com *chunk* que está em outra máquina e, finalmente, tarefa especulativa. Uma tarefa lançada para uma máquina livre que não possua os dados locais é denominada de tarefa remota. Em uma tarefa remota, a máquina necessitará copiar os dados de outra máquina antes de executar essa tarefa (WHITE, 2009).

Na fase de *Reduce* as tarefas são sinalizadas às máquinas através de uma função *Hash* aplicada às chaves intermediárias produzidas durante a fase de *Map*. A função de *Hash* divide as chaves intermediárias pelo número de *Reduces* definido pelo programador. As máquinas que irão processar a tarefa *Reduce* iniciam a cópia dos dados intermediários quando 5% das tarefas *Map* já estiverem concluídas, num processo denominado de *prefetch*. Porém, o processamento das tarefas *Reduces* somente iniciarão após todas as tarefas *Map* estarem concluídas. Na fase de *Reduce*, quando uma máquina tornar-se livre, as tarefas são escolhidas na seguinte ordem: tarefa que falhou, tarefa pendente e tarefa especulativa. (VENNER, 2009; WHITE, 2009)

Nos algoritmos do *MapReduce*, no ambiente homogêneo, são assumidas algumas simplificações como, por exemplo, as máquinas executam todas as tarefas com mesmo tempo de execução, o lançamento de uma tarefa especulativa não tem nenhum custo adicional associado, o progresso de execução das tarefas são praticamente iguais e os dados quase sempre estão à mesma distância nas máquinas (ZAHARIA et al., 2008).

2.3.1 Algoritmo para a Divisão de Dados

O algoritmo 2.3.1 descreve o processo de divisão dos dados (*data split*), executado pelo HDFS quando é lançado um *job* por um usuário. A entrada de dados é dividida em tamanhos iguais (Tam_chunk) que podem ser de 16, 32, 64 ou 128 MB (o padrão é 64 MB). Cada bloco de dados recebe o nome de *chunk*.

Uma vez definido o início e o fim de cada *chunk*, os dados são preparados e enviados às máquinas. No processo de envio, o HDFS testa o espaço disponível no *storage* de cada nó e distribui os dados de forma balanceada.

Um sistema de tolerância a falhas replica os *chunks* para evitar a redistribuição de dados em tempo de execução, caso ocorra a queda de uma máquina. O número de réplicas ($Num\_replicas$) é um parâmetro definido pelo programador (o padrão é 3). As réplicas são transferidas para as máquinas considerando também o balanceamento de carga (VENNER, 2009; WHITE, 2009).

Algoritmo 2.3.1 HDFS: *Split* dos dados

1. $Dados \leftarrow$ entrada de dados no HDFS
2. **if** $(Dados\ \%\ Tam\_chunk) \neq 0$ **then**
3. $\quad Num\_chunks \leftarrow (\frac{dados}{Tam\_chunk}) + 1$
4. **else**
5. $\quad Num\_chunks \leftarrow \frac{dados}{Tam\_chunk}$
6. **for** $j \leftarrow 1$ to Num_hosts **do**
7. $\quad HD(j) \leftarrow$ verifica tamHD do $Host(j)$
8. **for** $i \leftarrow 1$ to Num_chunks **do**
9. $\quad Chunk(i) \leftarrow Dados[i]$
10. $\quad$ **if** $HD(j) \geqslant Tam\_chunk$ **then**
11. $\quad\quad Host(j) \leftarrow Chunk(i)$
12. $\quad\quad HD(j) \leftarrow HD(j) - Tam\_chunk$
13. $\quad$ **else**
14. $\quad\quad j \leftarrow (j\ \%\ Num\_hosts) + 1$ { Vai para a próxima iteração }
15. $\quad\quad i \leftarrow i - 1$
16. $\quad$ **for** $r \leftarrow 1$ to $Num\_replicas$ **do**
17. $\quad\quad Host(j + r) \leftarrow Chunk(i)$
18. $\quad\quad j \leftarrow (j\%Num\_hosts) + 1$

2.3.2 Algoritmos do *TaskRunner* e *JobTracker*

Cada uma das tarefas *Map* tem um *buffer* de memória circular associado de 100 MB para escrever a saída de dados. Quando o *buffer* atinge 80% de sua capacidade inicia-se uma *thread* para executar um processo *spill* que descarrega seu conteúdo para o disco. As saídas do *Map* irão continuar a ser escritas no *buffer* enquanto o processo *spill* ocorrer. O processo segue um algoritmo *round-robin*, escrevendo para o *buffer* de memória e do *buffer* para o disco de forma circular até a conclusão da tarefa (WHITE, 2009).

Uma máquina recebe uma ou mais tarefas conforme sua quantidade de *slots* livres. Um *slot* define quantas tarefas podem ser processadas simultaneamente. O número é um parâmetro de configuração global definido pelo programador. As informações do progresso de uma tarefa são enviadas para o mestre através de um *heartbeat*. O progresso das tarefas é monitorado através de um processo chamado *TaskRunner* (WHITE, 2009).

A cada tarefa são associados uma prioridade e um tempo estimado para o término de sua execução, conforme é apresentado no algoritmo 2.3.2 para o *Map* e no algoritmo 2.3.3 para o *Reduce*. Caso o tempo de progresso da tarefa seja maior que o tempo médio das execuções das tarefas no *cluster* ($Average\_taskMap$) a máquina será marcada como *straggler*.

Ao término das tarefas pendentes uma nova tarefa *backup* é lançada pelo *JobTracker* para uma máquina que contenha uma réplica dos dados. O processo de lançamento de tarefas *backup* é chamado de execução especulativa. O *JobTracker* fica aguardando a conclusão da tarefa e a máquina que mandar primeiro uma mensagem informando da conclusão da tarefa (tarefa normal ou especulativa) força o cancelamento da outra.

Algoritmo 2.3.2 Hadoop: *TaskRunner-Map*

1. $Receive() = Heartbeat[StatusHost()]$
2. **if** Job não terminou **then**
3. $\quad New\_taskId() \leftarrow f(Map[Chunk_t])$
4. $\quad$ **if** $Time\_taskExec > 60s$ e $Job\_inProgress \leqslant 0,2 * Average\_taskMap$ **then**
5. $\quad\quad Host(j) \leftarrow Straggler$
6. $\quad\quad$ Troca prioridade da tarefa
7. $\quad\quad$ Lança tarefa especulativa para o $Host_{j+1}$ que tenha $Chunk_t$ {Tarefa especulativa}
8. Aguarda tarefa completar

No lançamento de tarefas *Reduce*, como no algoritmo 2.3.3, as chaves intermediárias residem na máquina que executa a tarefa de *Map*. Os dados são copiados das máquinas que executaram as *Map* para aquelas que executarão as tarefas *Reduce*, durante um processo de cópia na fase de *Shuffle*. Quando uma tarefa especulativa é lançada as chaves intermediárias $Map_n(key[n], value[n])$ devem ser copiadas antes da execução da tarefa.

Algoritmo 2.3.3 Hadoop: *TaskRunner-Reduce*

1. $Receive() = heartbeat[StatusHost()]$
2. **if** Job não terminou **then**
3. $\quad New\_taskId() \leftarrow f(Reduce(Map(key[n], value[n]))$
4. $\quad$ **if** $Time\_taskExec > 60s$ e $job\_inProgress \leqslant 0,2 * Average\_taskReduce$ **then**
5. $\quad\quad Host(j) \leftarrow straggler$
6. $\quad\quad$ Troca prioridade da tarefa
7. $\quad\quad New\_taskId_{esp}() \leftarrow f(Reduce(Map(key[n], value[n]))$ {Cria tarefa especulativa}
8. $\quad\quad Send(taskReduce, Host_{j+1})$
9. $\quad\quad Send(locais(P_{interm}(key[n], value[n]), Host_{j+1})$ { $Host_{j+1}$ Copia pares}
10. Aguarda tarefa completar

O lançamento de tarefas executado no mestre é apresentado no algoritmo 2.3.4. O processo *JobTracker*, executado no mestre, faz o controle e o escalonamento das tarefas do *MapReduce*. Quando o usuário submeter um *job*, o *JobTracker* verifica o número de máquinas participantes do *cluster* e cria um *Job_Id* para cada entrada de dados do usuário. Um processo de *split* divide os dados no HDFS, define as funções *Map* associadas a cada *chunk* e a localização dos dados.

O *JobTracker*, após receber uma chamada para criar uma nova tarefa de *Map* ou *Reduce*, cria a respectiva tarefa e coloca-a em uma fila FIFO. As propriedades das tarefas são definidas nas configurações do *job* que recebem o nome de *JobConf*. As propriedades do *JobConf* são copiadas para a tarefa e enviadas junto com um número de identificação dessa tarefa para a máquina processar.

As tarefas recebem uma prioridade que pode ser definida como: muito baixa, baixa, normal, alta e muito alta. Por padrão as tarefas são configuradas com prioridade normal e são então lançadas para execução. A prioridade das tarefas muda no $Job\_inProgress$ conforme o progresso da tarefa.

Uma função de *Hash* é aplicada sobre os pares intermediários da fase de *Map* para dividir os dados de forma balanceada para as máquinas. As tarefas *Reduce* são criadas conforme o número de *Reduces* ($Num\_reduces$), definido pelo programador. Quando uma máquina é notificada pelo mestre da localização dos pares intermediários, ela lê os dados intermediários do disco local de cada máquina. Após ler os dados intermediários, aplica-se um *sort* sobre as chaves intermediárias na fase de *Shuffle* para, então, serem agrupadas conforme sua ocorrência. O *sort* é necessário porque tipicamente muitas chaves intermediárias diferentes são produzidas pelo *Map* para uma mesma tarefa *Reduce* (DEAN; GHEMAWAT, 2010).

Algoritmo 2.3.4 Hadoop: *JobTracker*

1. **for** $i \leftarrow 1$ to Num_chunks **do**
2. $Submit\_task(x) \rightarrow Create\ New\_taskMap(i)$
3. **while** $t < Num\_hosts$ **do**
4. set $Job\_priority(x) \rightarrow taskMap(i)$
5. $Send(taskMap, Host_j)$
6. Executa Hadoop: *TaskRunner-Map*
7. **if** $Job\_inProgressTask[Map(finish)] > 5\%$ **then**
8. **for** $j \leftarrow 1$ to $Num\_reduces$ **do**
9. Divide chaves intermediárias com uma função de *Hash*
10. $Submit\_task(x) \rightarrow Create\ New\_taskReduce(i)$
11. set $Job\_priority(x) \rightarrow taskReduce(i)$
12. $Send(taskReduce, Host_j)$
13. $Send(locais(P_{interm}(key[n], value[n]), Host_j)$ { $Host_j$ Copia pares}
14. Executa Hadoop: *TaskRunner-Reduce*

2.4 *MapReduce* em Memória Compartilhada e GPUs

O *MapReduce* foi proposto para uso em ambientes de memória distribuída, como *clusters*. Porém, rapidamente foram apresentadas novas implementações do *framework* sobre memória compartilhada.

O objetivo é melhorar o desempenho do *MapReduce* para pequenas implementações. Ainda, mais recentemente, foi apresentado um modelo híbrido que utiliza ambas implementações de memória compartilhada e distribuída.

2.4.1 *Phoenix*

O trabalho de (RANGER et al., 2007) apresenta uma implementação do *MapReduce* em sistemas de memória compartilhada, incluindo uma API de programação e um sistema de *runtime*. O *Phoenix* gerencia automaticamente a criação de *threads*, o escalonamento dinâmico de tarefas, a partição de dados e o processo de tolerância a falhas.

A implementação do *Phoenix* é baseada nos mesmos princípios do *MapReduce*. Porém, objetivando o uso de sistemas em memória compartilhada com processadores *multi-core* e simétricos. Utiliza *threads* para criar tarefas paralelas de *Map* e *Reduce*, com *buffers* em memória compartilhada para facilitar a comunicação sem a cópia excessiva de dados.

O *runtime* escalona tarefas automaticamente através dos processadores disponíveis com o objetivo de proporcionar balanceamento de carga e maximizar o *throughput* de tarefas. Assim, o *runtime* gerencia a granularidade e a sinalização de tarefas paralelas, e recupera-se de transientes ou falhas durante a execução (RANGER et al., 2007).

Os fatores alocados pelo *runtime* são o tamanho das unidades, o número de máquinas envolvidas, as unidades que são atribuídas às máquinas dinamicamente e o espaço dos *buffers*. A decisão de qual fator alocar pode ser totalmente automática ou guiada pelo programador da aplicação.

As decisões tomadas permitem a execução eficiente da programação ao longo de uma grande variedade de máquinas e cenários sem modificações no código fonte. A API do *Phoenix* provém dois conjuntos de funções, um para inicializar o sistema e emitir pares de saídas e outro para definições do programador.

A Figura 2.5, adaptada de (RANGER et al., 2007), apresenta a estrutura do fluxo de dados do *runtime*. O *runtime* é controlado pelo escalonador, o qual é inicializado pelo código do usuário. O *worker* executa as tarefas de *Map* e *Reduce*.

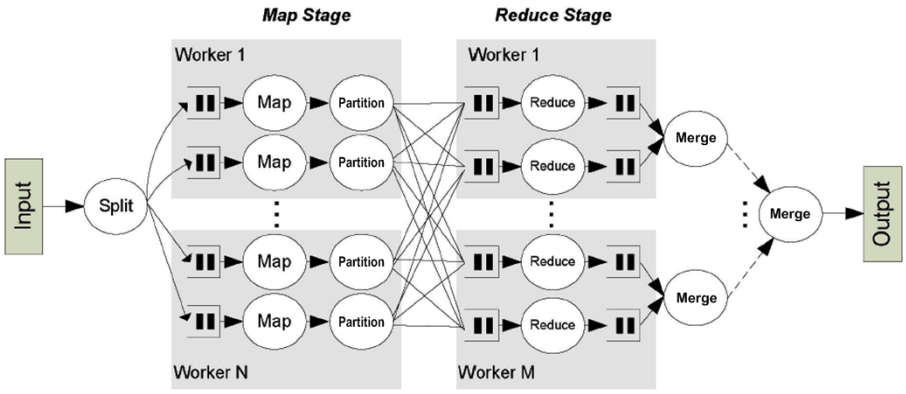


Figura 2.5: Controle do fluxo de fados do *runtime* no *Phoenix*

O dados intermediários produzidos no *Map* são divididos em partições, chamadas de partição de dados intermediários. As funções garantem que dentro de uma partição de dados intermediários, os pares serão processados na ordem da chave. Isto permite a produção de uma saída ordenada porém, não existem garantias da ordem de processamento da entrada do *Map*.

Após a inicialização o escalonador determina o número de núcleos para uso de sua computação e cria uma *thread worker* em cada núcleo. Ao iniciar o estágio de *Map* os dados são divididos através do *Splitter* em unidades de dados de tamanhos iguais para serem processados.

Então, o *Splitter* é chamado novamente pela tarefa *Map* e retorna um ponteiro para os dados da tarefa a ser processada. A tarefa *Map* aloca o número de *workers* necessários para a execução da função *Map*, então cada *worker* processa os dados e emite um par de chaves intermediárias.

A função *Partition* separa os pares intermediários em unidades para a execução das tarefas *Reduce*. A função assegura que todos os valores da mesma chave estarão na mesma unidade. Dentro de cada *buffer* os valores são ordenados por chave através de um *sort*. O escalonador aguarda todas as tarefas *Map* encerrarem para depois lançar as tarefas *Reduce*.

A tarefa *Reduce* aloca os *workers* necessários para processarem as tarefas de maneira dinâmica. Uma mesma chave é agrupada em uma única tarefa para ser repassada pelo *worker Reduce*. Nesta fase pode-se ter grande desbalanceamento de carga. Assim, o escalonamento dinâmico é muito importante para evitar este problema.

Quando as saídas *Reduce* estão ordenadas por chave é executado um *merge* de todas as tarefas em um único *buffer*. O *merge* gasta $\log_2(\frac{numWorkers}{2})$ passos para completar, onde $numWorkers$ é o número de *workers* usados (RANGER et al., 2007).

2.4.2 MARS

O MARS é uma implementação do *MapReduce* sobre GPUs, para fornecer aos desenvolvedores um *framework* sobre esta plataforma (FANG et al., 2011). A implementação pode ser utilizada para máquina de memória compartilhada ou de memória distribuída.

O projeto é dividido em três estágios: *Map*, Grupo e *Reduce*. O estágio Grupo é utilizado para produzir os dados intermediários na saída do *Map* e para calcular o tamanho dos dados intermediários. O objetivo do cálculo é determinar a área de *cache* necessária para criar cada *thread* na GPU. O fluxo da execução do MARS é apresentado na Figura 2.6, adaptada de (FANG et al., 2011).

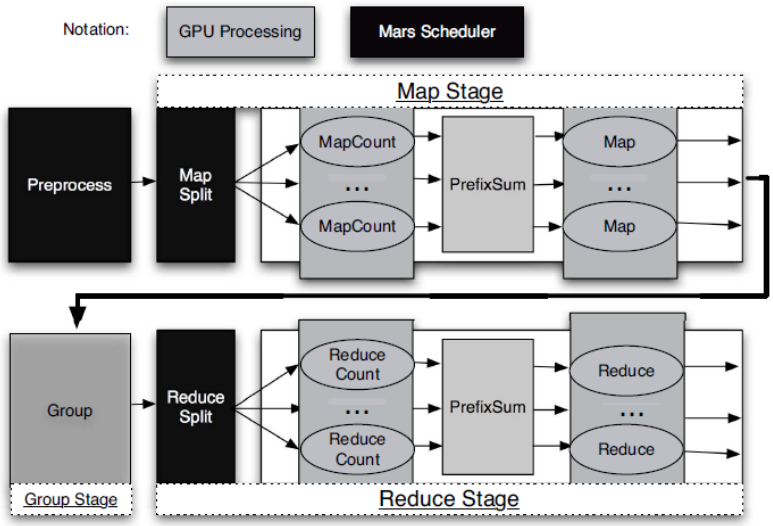


Figura 2.6: Fluxo de execução do MARS

As fases de *Map Count, Reduce Count* e *PrefixSum* são responsáveis por calcular o tamanho dos dados para alocar a memória necessária dentro das GPUs em função de que nesta estrutura a alocação dinâmica de memória não é permitida.

O estágio de *Map* faz o *split* despachando a entrada gravada para *threads* nas GPUs. Cada *thread* executa uma função *Map* definida pelo usuário. Na fase de Grupo, um *short*

e um *hash* são executados para preparar a próxima fase. O estágio *Reduce* separa os dados do Grupo e os envia às GPUs para executar a função *Reduce* e obter o resultado em um *buffer* único.

A estrutura dos dados no MARS afeta o fluxo de dados, o acesso à memória e a concepção do sistema. O MARS processa em CPU a entrada dos discos, transformando a entrada em pares chave/valor na memória principal, para ser transferida para a GPU.

A implementação em memória distribuída utiliza o Hadoop para distribuir o processamento em CPU/GPU para outras máquinas. Nesta implementação o sistema de arquivos distribuído, a divisão dos dados, o escalonamento de tarefas e os mecanismos de tolerância a falhas são funções executadas pelo Hadoop.

2.5 Trabalhos Relacionados

Vários esforços estão sendo feitos para buscar soluções que compatibilizem o uso de aplicações intensivas em dados com os recursos computacionais existentes. Entretanto, nem todo tipo de aplicação pode utilizar os modelos existentes ou então, o modelo pode não estar adequado às características dos ambientes disponíveis.

Nesta seção serão apresentados os estudos do *MapReduce* em memória distribuída que relacionam-se com as abordagens adotadas neste trabalho para a adequação dos algoritmos em ambientes heterogêneos.

2.5.1 Aplicações Intensiva em Dados vs. Computação Científica

Alguns cientistas vem estudando os processos utilizados no *MapReduce* para identificar o potencial uso deste *framework* em aplicações científicas intensivas em dados. O maior desafio para a implementação está no fato de que nem todas a aplicações podem ser escritas diretamente sobre a semântica do *MapReduce*.

Os trabalhos apresentados a seguir são o resultado de estudos de cientistas preocupados em criar novos mecanismos que permitam uso do *MapReduce* em experimentos científicos de larga escala. Chen e Steven estudaram o desempenho de alguns tipos de aplicações intensivas em dados com uso do *MapReduce* como, por exemplo, tarefas de computação com aprendizado de máquina de larga escala, simulações físicas e processamento de dados digitais (CHEN; SCHLOSSER, 2008). Estas aplicações tem problemas de desempenho porque as tarefas realizam muitos cálculos complexos além de serem intensivas em dados. O estudo aponta três tipos diferentes de aplicações que suportam computação em larga escala intensiva em dados, tais como:

1. Aprendizado de máquina sobre bases de dados grandes para, por exemplo, predizer a preferência de usuários a um determinado produto.

2. Cálculo de dados de simulações científicas de eventos sísmicos para criar modelos de regiões em formato 3D, com uso de CVM (*Community Velocity Model*).

3. Processamento de imagens para estimar informações geográficas, com um total de 6 milhões de imagens em 1 TB de dados.

O uso do *MapReduce* é desejável para estas aplicações, porém oneroso em consumo de recursos e tempo de processamento para estes conjuntos de dados. Os autores ponderam que o *MapReduce* está no meio termo entre os sistemas de base relacional, que tem operações bem definidas sobre os dados, mas são restritos quanto ao conjunto de dados e

cálculos, e os sistemas de programação paralela como MPI, que tem pouca restrição sobre os conjuntos de dados e o tipo de transformação que um programa pode fazer mas, no entanto, o sistema tem conhecimento limitado sobre as bases de dados e cálculos (CHEN; SCHLOSSER, 2008).

Os autores sugerem diversas mudanças para o uso mais eficiente do *MapReduce* como, por exemplo:

1. Uso de uma estrutura de análise dos dados da entrada, quando uma pequena fração dos dados é realmente necessária. Por exemplo, um esquema de indexação poderia acelerar significativamente o processamento.

2. O sistema deve permitir a composição flexível de componentes e o desligamento de componentes para melhorar o desempenho. Por exemplo, o *Map* poderia ser desativado, se a entrada de dados já está classificada

3. Permitir que múltiplas funções *Map* leiam fontes de entrada de arquivos diferentes para conectar-se à mesma função *Reduce*.

4. Trocar o algoritmo de agrupamento *merge-sort* por *HashMerge* que é mais rápido conforme na literatura em (GRAEFE, 1993).

5. Otimizar a execução de tarefas de longa duração. O sistema poderia calcular uma previsão para o tempo de execução de tarefas e identificar se há espaço suficiente no disco local para geração de dados intermediários.

No trabalho de (MACKEY et al., 2008), os autores mostram a importância do uso do *MapReduce* para aplicações intensivas em dados, como em simulações científicas, devido a sua simplicidade na programação e a necessidade do sistema ter alta disponibilidade e confiabilidade.

Aplicações científicas HEC (*High End Computing*) na escala de *Petabytes* de dados tem tornado-se uma fronteira para a maioria das aplicações científicas, que consistem de uma fase de simulação, geração de dados intermediários e uma fase de análise dos resultados dos dados, que são repetidas inúmeras vezes sobre uma mesma base de dados.

A Figura 2.7 apresenta a taxonomia para os modelos de programação paralela, adaptado de (MACKEY et al., 2008). Os aplicativos podem ser fracamente acoplados e exigirem baixa comunicação entre os nós como, por exemplo, os modelos do *Dryad*, BOINC e *MapReduce*.

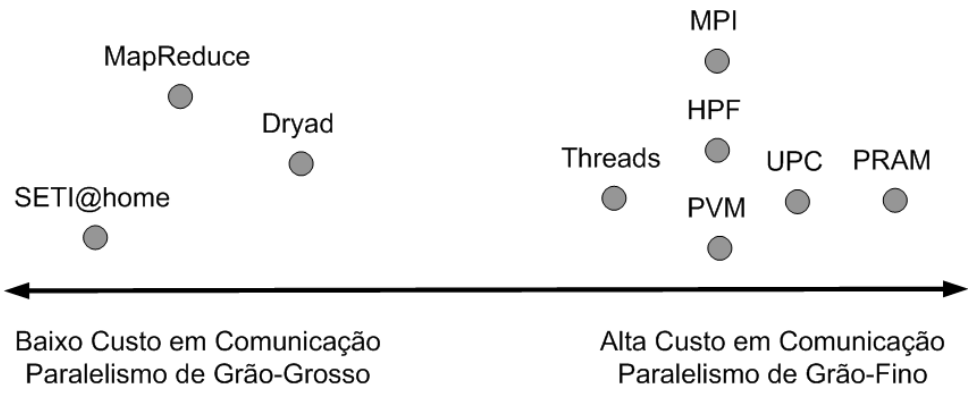


Figura 2.7: Taxonomia dos modelos de programação paralela

Em contrate, os aplicativos podem ser fortemente acoplados e terem de sincronizar suas tarefas especialmente quando estão sobre um mesmo conjunto de dados como, por exemplo, os modelos do MPI (*Message-Passing Interface*), HPF (*High Performance Fortran*), PVM (*Parallel Virtual Machine*), PRAM (*Parallel Random Access Machines*), UPC (*Unified Parallel C*), etc.

O uso do *MapReduce* neste contexto permite a manipulação de grandes quantidades de dados, porém o custo de perda da granularidade da programação ainda é um fator relevante. Os dados e o paralelismo destas tarefas são motivação para a migração total ou parcial destas aplicações para o uso do *MapReduce*. Porém, o uso deste *framework* para este tipo de aplicações ainda é uma questão em aberto.

O trabalho de Gufler trata do problema do particionamento de tarefas *Reduce* em aplicações científicas que exibem propriedades para as quais os atuais sistemas do *MapReduce* não foram concebidos (GUFLER et al., 2011). As aplicações científicas estudadas pelos autores tem complexidade não-linear no tempo para as execuções de tarefas *Reduce*. A distribuição de dados, neste caso, são geralmente heterogêneas. Assim, uma elevada complexidade conduz a tempos de execução muito diferentes. Portanto, máquinas com baixa carga de trabalho devem esperar por outras com carga de trabalho elevada.

O problema é tratado como o algoritmo *bin packing problem* (BOYAR et al., 2006) de complexidade *NP-Hard*. O algoritmo *bin packing problem* é uma variação do problema da mochila. A abordagem consiste de particionar as tarefas *Reduces* em aglomerados de partições com uma quantidade de chaves diferentes de maneira a minimizar o número de aglomerados. Um aglomerado é um subconjunto de todos os pares (chave/valor) que compartilham a mesma chave.

A Figura 2.8 apresenta o modelo de distribuição de chaves intermediárias para o *Reduce* estudado pelos autores. O objetivo é adequar o tempo de execução a uma quantidade de aglomerados para ter-se um custo de execução homogêneo entre as máquinas. Os autores argumentam que o processamento de um pequeno número de grandes aglomerados de chaves leva muito mais tempo para ser executado do que o processamento de muitos pequenos aglomerados.

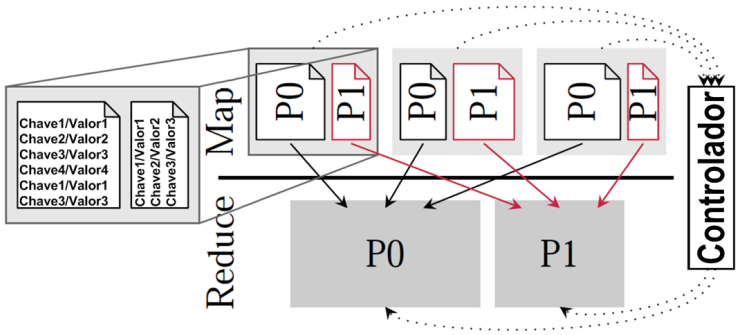


Figura 2.8: Modelo de distribuição de chaves intermediárias para o *Reduce*

O modelo tem dois algoritmos, um para calcular o custo das partições e outro para calcular a carga de trabalho para as máquinas que executam as tarefas *Reduces*. Os autores adotaram duas abordagens para o particionamento dos dados intermediários. A primeira chamada de particionamento de granularidade fina, divide os dados da entrada em um número fixo de partições. O número de partições é maior do que o número de *Reduces*. O objetivo é distribuir as partições conforme os tempos de execução de forma semelhante para todas as máquinas. Uma partição de granularidade fina não controla o custo das

partições, enquanto elas são criadas, mas consegue cargas equilibradas pela distribuição de partições caras para diferentes máquinas.

A segunda abordagem é chamada de fragmentação dinâmica. As partições de dados intermediários são divididas localmente em cada máquina que executa uma tarefa *Map* e, enquanto as partições são criadas, tuplas são replicadas se necessário. Como resultado, o custo das partições é mais uniforme e é fácil ser obtido um balanceamento de carga para distribuições de chaves altamente distorcidas, ou seja, chaves que tem quantidades de dados não uniformes.

A abordagem é utilizada em *clusters* homogêneos. Os autores advertem que como o algoritmo *bin packing problem* tem complexidade *NP-Hard*, o cálculo de um número moderado de aglomerados poderá tornar-se mais custoso para ser obtido do que o custo da execução atual das tarefas *Reduces*. Assim a exata monitoração em nível de aglomerado pode não ser uma tarefa fácil. Os autores apontam que uma das possíveis soluções seriam a coleta de dados em nível da granularidade entre aglomerados e particionamento de chaves.

Os autores não trabalharam a solução para adequar sua proposta ao ambiente heterogêneo. Assim uma abordagem com o uso da granularidade fina das chaves intermediárias pode constituir-se em uma possibilidade de estudos para melhorar o desempenho dos *jobs* em ambientes *desktop grid*.

2.5.2 Ambientes Heterogêneos

A maioria das pesquisas sobre os problemas de desempenho do *MapReduce* concentra-se na adequação ao uso o *framework* para ambientes heterogêneos. Os esforços são no sentido de alterar os algoritmos de distribuição de dados e lançamento de tarefas especulativas.

O trabalho de Zaharia trata de problemas de desempenho do *MapReduce* em ambientes heterogêneos, causados pela execução de diferentes aplicações em grandes *clusters* (ZAHARIA et al., 2008). O estudo aponta alguns problemas encontrados com o uso do modelo do *MapReduce* para lançamento de tarefas especulativas.

A heurística simples faz o escalonador lançar tarefas especulativas conforme o progresso das tarefas de cada máquina, baseado na média do progresso de todas as tarefas. Assim, em um ambiente heterogêneo, as tarefas são executadas como se todas tivessem a mesma taxa de progresso, adequada somente para o uso de aplicações homogêneas em *clusters* de *data centers*.

Quando o escalonador percebe que uma máquina está lenta, ele lança a mesma tarefa para uma outra máquina que esteja livre com o objetivo de evitar atrasos e minimizar o tempo de resposta. Entretanto, o processo consome recursos de CPU e compete pelo uso da rede com outras execuções normais.

O problema é que o escalonador acaba lançando mais tarefas que deveria em ambientes heterogêneos, em função de que os recursos computacionais podem ter tempos de resposta diferentes para cada tarefa. Alguns experimentos apresentados pelos autores constatam que mais de 80% das tarefas eram executadas especulativamente.

São apontados pelos autores os seguintes problemas:

1. As tarefas especulativas não são livres - elas competem por alguns recursos com as outras tarefas em execução, *e.g.* a rede.

2. A escolha da máquina para executar uma tarefa especulativa é tão importante quanto a escolha da tarefa.

3. No ambiente heterogêneo, pode ser difícil distinguir entre as máquinas que são ligeiramente mais lentas do que a média e aquelas que são realmente *stragglers*.

4. A demora na identificação de máquinas *stragglers* compromete o tempo de resposta de toda aplicação.

Para resolver estes problemas os autores implementaram alterações no escalonador de tarefas para minimizar a degradação em ambientes heterogêneos. O escalonador executa o algoritmo chamado LATE (*Longest Approximate Time to End*) para lançar tarefas especulativas.

O algoritmo lança uma tarefa especulativa, somente se a tarefa executada na máquina *straggler* for aumentar o tempo de resposta do *job*. A heurística adota uma estimativa da taxa de progresso em percentual para cada tarefa, dada pela equação 2.3.

$$\textbf{\textit{ProgressRate}} = \frac{ProgressScore}{T}\ \% \tag{2.3}$$

Onde T é o tempo total de execução da tarefa que está rodando. A estimativa do tempo de conclusão da tarefa é dado pela equação 2.4.

$$\textbf{\textit{Tempo de Conclusão}} = \frac{1 - ProgressScore}{ProgressRate} \tag{2.4}$$

Assim, o progresso das tarefas ocorrem a um ritmo mais ou menos constante. Entretanto, podem existir casos em que essa heurística possa falhar, se a heterogeneidade for maior que a esperada, mas é eficaz em empregos típicos do Hadoop. O algoritmo LATE, identifica as tarefas que irão ter um tempo de resposta mais sofrível e as executa o mais rápido possível (ZAHARIA et al., 2008).

A heterogeneidade do ambiente simulado é obtida através de máquinas virtuais sobre *clusters* homogêneos. Aplicações nas máquinas virtuais executam processos de leitura e escrita intensiva em disco nos sistemas operacionais convidados. A degradação do sistema representa 2,5 vezes a do sistema homogêneo.

A avaliação dos experimentos demonstra que o uso do LATE, comparado com a execução nativa do Hadoop em modo especulativo, obteve ganhos variáveis de 8,5% a 58% dependendo da aplicação e da quantidade de máquinas. O ganho do LATE é menor com tarefas menores do tipo *WorldCount*, executando com um número maior do que 900 máquinas.

Outros autores como Xie propõem o aumento de desempenho do *MapReduce* em ambientes heterogêneos pela alocação de dados conforme a capacidade das máquinas (XIE et al., 2010). A proposta consiste em adotar um algoritmo de realocação dos dados adicionado ao sistema de distribuição de dados do sistema de aquivos distribuídos.

Os algoritmos foram testados sobre o HDFS e são divididos em três esquemas: um de divisão dos dados por capacidade, outros de redistribuição dos dados divididos conforme o uso e um terceiro para entradas de novos dados durante o processamento. Uma execução da aplicação é feita para descobrir o tempo de execução das tarefas no início.

Um dos problemas da abordagem adotada é a eliminação do sistema de replicação de dados existente no *Mapreduce*, sob o argumento de que a replicação de dados causa uso desnecessário de espaço em disco. A modelagem realoca dados em tempo de execução quando for adicionado dados novos por um cliente. Porém, no modelo do *MapReduce*,

esta realocação por entrada de novos dados não ocorre em tempo de execução sobre um mesmo *job*.

O *MR-A++* utiliza um mecanismo mais apurado para a descoberta da capacidade das máquinas, além de utilizar o tempo de execução de tarefas. Um algoritmo específico é utilizado no *MR-A++* para distribuir os dados para a execução da tarefa de coleta de informação. O autor propõe utilizar uma grande quantidade de dados, como por exemplo 1 GB, para identificar os tempos de execução. Os valores desta ordem são proibitivos em ambientes heterogêneos de larga escala devido aos custos associados com a transferência de dados.

O autor, propõe a eliminação da replicação de dados. Porém, o custo associado é a perda de dados quando ocorrer a queda de uma máquina que contenha dados ainda não processados. Uma falha permanente da tarefa cria uma redistribuição de dados, sobrecarregando as máquinas escravas e a rede. A metodologia dos testes realizados por Xie, com uso de apenas 5 máquinas heterogêneas, não garante a escalabilidade dos algoritmos apresentados.

O escalonamento com carga dinâmica é outra proposta para melhorar o desempenho do *MapReduce* em ambientes heterogêneos (YOU; YANG; HUANG, 2011). Os autores consideram o escalonamento de tarefas especulativas segundo uma política de escalonamento dinâmico conforme a capacidade computacional de cada máquina.

Os dados da carga de processamento das máquinas são coletadas em tempo de execução para estimar quanto uma determinada máquina está ocupada para poder completar sua tarefa. O objetivo é diminuir o número de tarefas especulativas. Entretanto, a medida de carga das máquinas não muda a localização dos dados já distribuídos. Assim, as máquinas com capacidades de processamento incompatíveis com a carga de trabalho continuarão gerando tarefas especulativas.

Os autores não consideram a cópia de dados durante o re-escalonamento das tarefas e a consequente distribuição dos dados a serem processados. Assim na fase de *Map*, por exemplo, tem-se o uso do barramento de rede para executar-se as adequações em tempo de processamento. Portanto, a abordagem pode gerar a competição pelo uso do barramento de rede.

O trabalho ainda apresenta uma escalabilidade baixa. O testes foram executados em 8 máquinas de 8 processadores. O *Mapreduce* por definição é utilizado em aplicações intensivas em dados e não somente no processamento intensivo. Assim, por exemplo, com um número de 2.000 a 4.000 máquinas, o processo de sincronização é um fator crítico a ser observado.

2.5.3 Ambientes Voláteis

Alguns cientistas, mais recentemente, estão envolvidos em pesquisas para adequar os algoritmos do *MapReduce* ao ambiente heterogêneo e volátil. A adequação permitirá o uso de dados intensivos sobre plataformas de ambientes oportunistas.

A primeira proposta foi feita por Lin, no projeto MOON (***M****apReduce* ***O****n* ***O****pportunistic* *e****N****vironments*) (LIN et al., 2009), os autores propõem um modelo híbrido de computação voluntária com o uso de máquinas não voláteis. O objetivo é compensar as perdas com a replicação que ocorreriam em um ambiente volátil, com a indisponibilidade momentânea das máquinas na *grid*. Entretanto, os autores não apresentam uma solução para o escalonamento de tarefas. Porém, apontam que em emulações de ambientes de computação voluntária existem problemas com a volatilidade de recursos oportunistas com o uso do Hadoop como:

1. O HDFS fornece um mecanismo de replicação que, para ambientes voláteis, pode ter um custo muito alto de replicação para prover alta disponibilidade dos dados. Para uma máquina com indisponibilidade de 40%, seriam necessárias 11 réplicas dos dados para ter-se 99,99% de disponibilidade.

2. Os dados intermediários produzidos pelo *Map* não são salvos em um sistema de arquivos distribuídos como o HDFS. Isto causa a reexecução de tarefas em caso de perda de dados.

3. O escalonador do Hadoop pressupõe que a maioria das tarefas ocorrem sem problemas até sua conclusão. No entanto, as tarefas podem ser frequentemente suspensas ou interrompidas em sistemas de computação voluntária. A estratégia de replicação de tarefas do *MapReduce* é insuficiente para lidar com a alta volatilidade das plataformas de computação voluntária.

Os autores propõem então três estratégias para resolver o problema: Adoção de um modelo híbrido com o uso máquinas voláteis e não voláteis. As máquinas não voláteis seriam dedicadas; Estender o HDFS para aproveitar os recursos dedicados e aplicar uma política diferenciada de replicação para diferentes tipos de tarefas utilizando o *MapReduce*; Estender o algoritmo de lançamento de tarefas especulativas levando em conta a volatilidade dos recursos e colocar estrategicamente a execução de tarefas nas máquinas dedicadas e nas voláteis.

A Figura 2.9, adaptado do trabalho de (LIN et al., 2009), apresenta a estratégia adotada pelos autores para decidir se os dados devem ser armazenados em ambiente dedicado ou volátil.

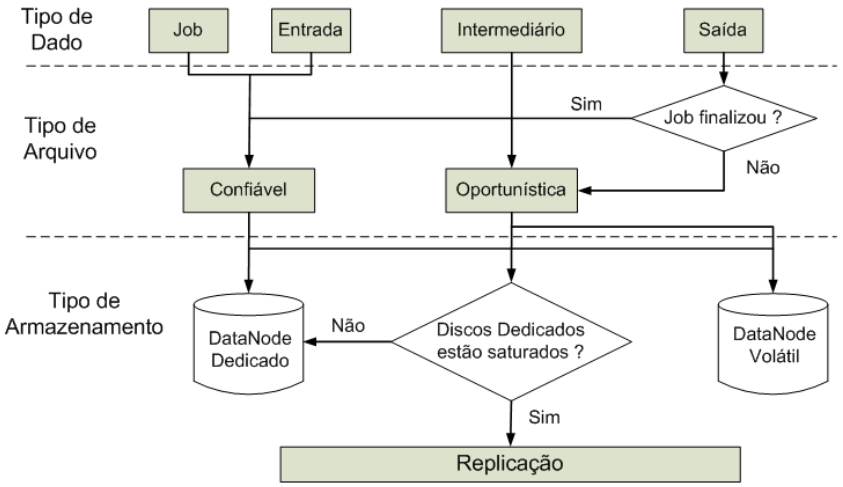


Figura 2.9: Processo de decisão para armazenamento de dados

Na execução do *Map*, para os arquivos com réplicas em ambos os *DataNodes* voláteis e dedicado, as máquinas voláteis sempre buscam primeiro os dados no seu disco local, caso não esteja disponível procuram alguma réplica volátil primeiro e, se não a encontrar, irão requisitar os dados para o *DataNode* dedicado.

Se o dado é intermediário, primeiro será gravado no disco local da máquina volátil, caso o *DataNode* dedicado não esteja saturado, gravará também uma réplica. Os resultados do *Reduce* serão sempre gravados nos *DataNodes* dedicados (LIN et al., 2009).

Este processo de cópia, embora traga maior confiabilidade em ambientes com máquinas voláteis, aumenta o processo de comunicação pela rede. Como não há um histórico de disponibilidade, não se tem a priore uma probabilidade de falha das máquinas. Assim, o *DataNode* dedicado pode ocupar totalmente seus recursos e vir a falhar.

Os dados confrontados dos experimentos do MOON (com parâmetros alterados para o lançamento de tarefas especulativas do Hadoop) com o MOON-Híbrido (com adição de máquinas dedicadas) tem pouca diferença nos resultados um com o outro. O desempenho é praticamente igual se for considerando o cenário de indisponibilidades inferiores a 30% com tarefas de curta duração de até 5 min. Entretanto, com a abordagem, ainda há um lançamento significativo de tarefas especulativas.

Algumas propostas tem usado outros *middlewares* adaptados ao uso sobre *MapReduce*, *e.g.* o *Bitdew* (FEDAK; HE; CAPPELLO, 2008). O *Bitdew* é um *middleware* de gerenciamento de *desktop grid* que utiliza uma infra-estrutura P2P. Uma das propostas foi feita por Tang (TANG et al., 2010) para criar uma implementação de uma API do *MapReduce* sobre o *Bitdew*. A implementação utiliza máquinas voláteis em uma *desktop grid* para realizar a execução de tarefas e a distribuição de dados.

Os dados são distribuídos sob demanda para o processamento das máquinas e toda carga computacional é previamente avaliada em tempo de execução. As barreiras computacionais entre as fases de *Map* e *Reduce* foram eliminadas. A implementação tem um comportamento diferente se comparado com o modelo do *MapReduce*.

A arquitetura apresentada na Figura 2.10 adaptada de (TANG et al., 2010), mostra os *workers* e o *master* e uma API *MapReduce* que gerencia os serviços. O núcleo dos dados tem a função de dividir os dados e mantê-los nas máquinas sobre uma DHT (*Distributed Hash Table*).

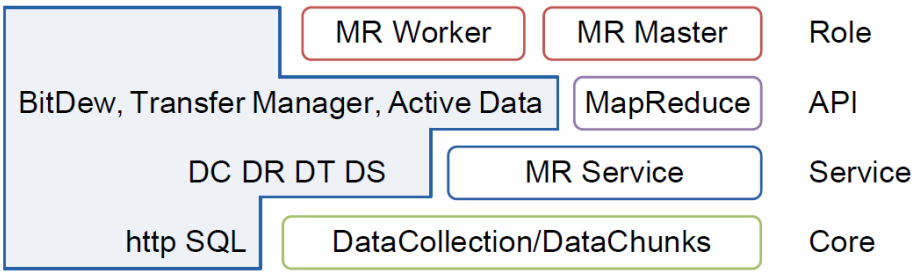


Figura 2.10: Arquitetrura do *MapReduce* sobre *Bitdew*

O ambiente utiliza uma arquitetura híbrida, com máquinas estáveis para o gerenciamento como o *master* e máquinas voláteis (*workers*) que executam as tarefas de *Map* e *Reduce*. O armazenamento de dados é feito sobre os *workers* sobre um mecanismo replicação sobre uma DHT utilizando os protocolos *http, ftp* e *BitTorrent*.

O tamanho do *chunk* é variável conforme a aplicação e os dados são recebidos no momento do processamento, diferentemente do modelo original do *MapReduce* que distribui os dados para as máquinas antes de escalonar as tarefas.

Os dados são distribuídos para o processamento no momento da execução das tarefas e apenas uma máquina executa a função de *Reduce* fazendo diversas interações até que todos os dados intermediários sejam processados.

A comunicação direta entre os *workers* não é garantida devido as características da rede, assim as máquinas podem processar diversos arquivos concorrentemente através do uso de filas de tarefas. O número de tarefas concorrentes é configurado antes de ser

realizada a computação pelas máquinas. Os dados intermediários produzidos na função de *Map* são mantidos dentro da DHT com uso de replicação.

A barreira computacional existente no *MapReduce* não é implementada embora o mecanismo pode detectar o tempo de processamento do *Reduce* em tempo de execução. As máquinas lentas não são detectados com facilidade, ao invés disto aumentam o fator de replicação das tarefas restantes para compensar o efeito de máquinas *stragglers*.

Os algoritmos utilizados são diferentes da implementação original do *MapReduce*, que embora válidos, realizam diversas alterações no funcionamento para adequar-se ao ambiente heterogêneo e volátil.

2.6 Comparativo entre Implementações do *MapReduce*

A Tabela 2.1 apresenta um comparativo entre as implementações e as principais soluções propostas do *MapReduce* para aplicações intensivas em dados em memória distribuída. A comparação baseia-se nos algoritmos e nas características da implementação original do *MapReduce*.

Tabela 2.1: Comparativo do estado da arte do *MapReduce*

Implementação	*MapReduce* Google	Hadoop	Hadoop LATE[1]	Hadoop Moon	Hadoop Data Placement	BitDew
Principal Autor	J.Dean	Cafarella	Zaharia	Lin	Xie	Tang
Ano	2004	2006	2008	2009	2010	2010
Tam. Dados *Map*	Tamanho fixo; *chunks*	=	=	=	=	Cap. Rede
Tam. Dados *Reduce*	*Hash*; N^o Maq. *Reduces*	=	=	=	=	=
Dist. Dados *Map*	Balanceada; Tam. HD	=	=	=	Capac. Proc.	Demanda
Dist. Dados *Reduce*	N^o Maq. *Reduces*	=	=	=	=	=
Tarefas *Map*	N^o *chunks*	=	=	=	=	Filas
Tarefas *Reduce*	*Hash*; N^o Maq. *Reduces*	=	=	=	=	Filas
Replicação Dados	Sim	Sim	Sim	Híbrido	Não	Sim
Tarefas Especulativas	$\overline{\mu}$ Exec $> 60s$ e Prog$< 20\%$	=	LATE[1]	LATE[1]	Não	Não
Sist. Arquivos	GFS	HDFS	HDFS	HDFS	HDFS	DHT
Ambiente Homogêneo	Sim	Sim	Sim	Sim	Sim	Não
Ambiente Heterogêneo	Não	Não	Sim	Sim	Sim	Sim
Trata Volatilidade	Não	Não	Não	Replicação	Não	Sim

"="Mantém o modelo original; [1] ***Longest Approximate Time to End*; DHT** $\rightarrow$ ***Distributed Hash Table*** **sobre P2P e** ***Bittorrent***

2.7 Distribuição Estatística de Dados

A probabilidade é uma ferramenta essencial para o projeto e a análise de algoritmos probabilísticos e aleatórios. A probabilidade é definida em função de um espaço amostral. O espaço amostral é um conjunto de elementos que recebem a designação de eventos (CORMEN et al., 2001).

Uma distribuição de probabilidade é um espaço amostral que pode ser finito ou contavelmente infinito . As funções contidas neste espaço amostral são ditas variáveis aleatórias. Se uma variável aleatória pode assumir um número contável de valores possíveis ela é dita discreta. De outra forma, existem variáveis aleatórias cujo conjunto de valores possíveis são incontáveis e neste caso elas são chamadas de contínuas (ROSS, 2010).

Uma variável aleatória contínua tem função densidade de probabilidade com propriedades semelhantes para um intervalo fechado $a \leqslant X \leqslant b$. Também é possível dizer que X é uma variável aleatória normal ou normalmente distribuída, dada pela função $f(x)$ da equação 2.5 (ROSS, 2010).

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}}, \forall x \in -\infty < x < \infty \tag{2.5}$$

A distribuição normal é um exemplo de distribuição de probabilidade, porém existem outras classes de distribuições que podem representar o comportamento das máquinas em um *cluster*, tais como *Heavy-tailed*, *Gamma, Weibull* e Distribuição Beta.

2.7.1 Classes de Distribuições

No trabalho de Crovella (CROVELLA; HARCHOL-BALTER; MURTA, 1998) os autores apresentam uma classe de distribuições, que geralmente é utilizada para capturar as características altamente variáveis de processos estocásticos, ou seja, que são mais variáveis do que a distribuição exponencial, conhecida com Classe de Distribuição *Heavy-tailed*. O objetivo do estudo era aumentar o desempenho em ambientes com desbalanceamento de carga com uso de uma política *task-assignment*. O algoritmo para a política *task-assignment* atribui uma tarefa de acordo com um processo de *Poisson* com taxa λ, específica para cada máquina.

Esta abordagem poderia ser utilizada para a distribuição de dados da entrada no *MapReduce*, em um ambiente heterogêneo. O estudo de (HARCHOL-BALTER, 1999) usa a mesma distribuição aplicada a tarefas de duração desconhecidas que são escalonadas segundo uma política FIFO.

As aplicações de duração desconhecida citadas por HARCHOL-BALTER podem ser comparadas no *MapReduce*, de uma forma abstrata, às tarefas de *Reduce*. Uma vez que não se pode prever com exatidão a quantidade de dados intermediários gerados para cada chave durante a fase de *Map*. Portanto, poderia ser implementado uma política de escalonamento FIFO para essas tarefas.

Para Ucar (UCAR et al., 2006), na política de *task-assignment* em ambientes heterogêneos o custo de execução depende do processador na qual a tarefa é executada. Para otimizar o desempenho de algumas métricas, tais como a utilização do sistema e do tempo de resposta, as tarefas são então modeladas usando um gráfico de interação de tarefas TIG (*Task Interaction Graph*). No modelo TIG, os vértices do grafo correspondem às tarefas e as arestas correspondem às comunicações inter-tarefa.

No estudo de Javadi (JAVADI et al., 2009) através de estudos empíricos do *SETI@home* foram identificados conjuntos de máquinas com propriedades estatísticas similares que possuem modelos de disponibilidade semelhantes em sistemas distribuídos de larga escala. As aplicações investigadas necessitavam de pequeno tempo de resposta rápido e incluíam tarefas em lote de computação intensiva estruturadas como um grafo acíclico orientado (DAG - *directed acyclic graph*).

O estudo conclui que a distribuição global para todas as máquinas, com disponibilidade independente e identicamente distribuída (iid), sofre influência do modelo dos algoritmos de escalonamento estocásticos. Diferentes distribuições podem ser adotadas, como, por exemplo, as distribuições de probabilidade *Gamma, Weibull* e *Heavy-tailed*, com objetivo de prever o comportamento das execuções das aplicações em um determinado ambiente.

O trabalho de Dongarra (DONGARRA et al., 2007) mostra que se a execução de um DAG seguir uma taxa de falhas constante, o modelo de falhas segue uma lei da potência, dada por uma variável aleatória "λ" que representa a taxa de falhas.

Assim, usando a distribuição de *Poisson*, a probabilidade de uma tarefa qualquer executar corretamente é dada por $e^{-\lambda t}$, desde que o número de tarefas seja muito maior que a quantidade de máquinas. Dongarra propõe dois algoritmos estocásticos para ambientes heterogêneos: o *Optimal* para a ótima alocação de tarefas unitárias independentes e o HEFT (*Heterogeneous Earliest Finish Time*) para o caso genérico.

As abordagens anteriores são semelhantes à estrutura de processamento de dados do *MapReduce* conforme relata a análise de traços de *clusters MapReduce* no trabalho de (KAVULYA et al., 2010). Os autores ao constatarem que os tempos de execução de *jobs* seguem uma distribuição de cauda longa, indicam que estes modelos de distribuição também poderiam ser utilizados para a distribuição de tarefas e predição de falhas, considerando as características do ambiente heterogêneo e volátil.

2.8 Uso de Simuladores em *MapReduce*

As implementações de ambientes em larga escala com milhares de máquinas é muito complexa. Os poucos recursos disponibilizados para simulações científicas, como a *Grid 5000* (INRIA; CNRS, 2011), possuem ambientes de *clusters* homogêneos onde a criação de ambientes heterogêneos em larga escala é muito difícil. Assim, o estudo dos simuladores existentes para o *MapReduce* justifica-se para identificar qual ferramenta possa ser utilizada para o estudo de modificações dos algoritmos sobre ambientes heterogêneos.

2.8.1 Simuladores Existentes para MapReduce

O estudo de algoritmos de escalonamento e distribuição de tarefas em ambientes intensivos em dados, como o *MapReduce*, exige uma grande quantidade de máquinas, para ter-se um tratamento adequado de grandes volumes de dados em larga escala. A construção de experimentos com centenas ou até milhares de máquinas em computação voluntária, como em *desktop grids*, não é uma tarefa trivial. Portanto, a utilização de um simulador se faz necessária para este estudo.

Cardona *et al.* apresentaram um simulador para *grids* baseado nos simuladores *GridSim* e *SimJava* com foco na simulação de funções *Map* e *Recude* em um sistema de arquivos semelhante ao HDFS (CARDONA et al., 2007). O simulador, entretanto tem muitas simplificações sobre o ambiente do *MapReduce* como, por exemplo, não considerar replicações, execuções especulativas e *split* de dados.

Outro simulador é o Mumak (TANG, 2009) que foi desenvolvido para simular o ambiente de aplicações do *MapReduce*. O objetivo é ser uma ferramenta para planejar a implantação de ambientes de larga escala. O simulador é aplicado somente a ambiente homogêneo, não desempenha simulações de entrada/saída e computação, assim como, não detecta operações de comunicação como *heartbeat*. Assim não é possível simular funções como execuções especulativas.

Wang (WANG et al., 2009) propôs o uso de um simulador chamado *MRPerf*, como um meio de planejar a configuração de parâmetros de otimização para a implantação do *MapReduce* em ambientes de larga escala. A implementação atual limita-se à modelagem de um único dispositivo de armazenamento por máquina, com suporte a apenas uma réplica para cada bloco de dados como num sistema HDFS. Não aplica as técnicas de otimizações da modelagem reais do *MapReduce*, tais como execução especulativa e replicação

da entrada de dados.

O simulador MRSim foi desenvolvido baseado no GridSim e SimJava e simula aplicações do *MapReduce*. As características do simulador incluem o *split* de dados e a replicação de dados no nível do *rack* local (HAMMOUD et al., 2010). Entretanto ambientes de execução real tem centenas ou milhares de máquinas distribuídas sobre diversos *racks*. Neste caso, os dados são replicados em diferentes *racks*. Porém, nenhuma interface é fornecida para modificar os algoritmos como,por exemplo, o escalonamento de tarefas e a distribuição de dados.

O uso destes simuladores se mostrou inadequado para a simulação de novos ambientes tais como *grids* ou *desktop grids*. A falta de módulos básicos do *MapReduce* e a necessidade da reestruturação das funções internas dos simuladores, representam um esforço de desenvolvimento muito alto. Diante deste cenário, com o estudo dos simuladores existentes verifica-se que existe a necessidade de construção de um simulador que atenda aos requisitos para a simulação de ambientes heterogêneos, para permitir alterar os algoritmos do *MapReduce*.

2.9 Caracterização do Problema

Alguns dos trabalhos apresentados neste capítulo, apontaram problemas e soluções relacionados com o uso do modelo do *MapReduce*. Os problemas em aberto refletem o tamanho do desafio em adequar o modelo às necessidades de tratamento das aplicações intensivas em dados em ambientes heterogêneos. Os principais problemas a serem superados para uso de aplicações intensivas em dados em ambientes *desktop grid*, podem ser resumidos, não intensivamente, em:

1. **Problemas na concepção do modelo**: Simplificações do modelo foram feitas para uso de tarefas curtas, tais como:
   - **Uso em *clusters***: O *MapReduce* foi construído para ser utilizado em grandes *clusters* e as simplificações refletem este ambiente;
   - **Tempo de execução de tarefas**: Os escalonadores foram projetados para que as tarefas executem em tempos iguais;
   - **Cópias de Dados**: O custo de cópias de dados em redes de alta disponibilidade é baixo;
   - **Problemas com heterogeneidade**: Performance baixa em ambientes heterogêneos, principalmente em *links* lentos;
2. **Falha do mestre**: O mecanismo de Tolerância a Falhas não prevê a falha do mestre;
3. **Semântica para escrever aplicações**: A semântica do *MapReduce* dificulta a portabilidade de aplicações mais complexas, como aplicações científicas;
4. **Volatilidade**: *MapReduce* não tolera facilmente ambientes voláteis, devido a perda de dados e reestruturação da rede em tempo de execução;
5. ***Jobs* longos**: Falta de mecanismos para a predição do tempo de execução dificultam uso de aplicações com tarefas longas.

2.9.1 Problemas do Modelo

O modelo do *MapReduce* foi projetado inicialmente para ambientes de grandes *clusters* homogêneos e para otimizar a distribuição de tarefas, algumas simplificações foram feitas. Entretanto, para a aplicação em ambientes heterogêneos, as simplificações causam a degradação do sistema (ANJOS et al., 2010).

As aplicações concebidas para o uso do *MapReduce* normalmente são tarefas curtas que iniciam e terminam rapidamente (ZAHARIA et al., 2008). Assim, os tempos de execução das tarefas de *Map* e *Reduce* são considerados iguais em cada fase e o progresso das fases ocorre em frações constantes de tempo.

Para o caso do *Reduce*, com uma fase de cópia, *merge* e redução de dados, considera-se $1/3$ do tempo do processamento total para cada uma destas etapas (ZAHARIA et al., 2008). Esta abordagem não é adequada para ambientes onde ocorre a variação do tempo de execução de tarefas, como por exemplo, devido a diferenças na capacidade computacional das máquinas ou em presença de falhas.

O processamento do *Map* produz dados intermediários que são armazenados temporariamente na própria máquina. Caso a mesma máquina venha a executar o *Reduce*, os dados serão consumidos localmente e tem-se um ganho de desempenho. Porém, os dados intermediários normalmente deverão ser transferidos pela rede. Neste caso, se ocorrerem falhas em uma tarefa *Map*, uma nova tarefa deverá ser lançada sobre os mesmos dados que estejam replicados em outra máquina. Portanto, falhas na geração de dados intermediários geram atrasos no progresso do *job* e degradam todo o sistema. O problema aumenta quando tem-se um *data flow* (*jobs MapReduce* em cascata) na presença de máquinas que falham (KO et al., 2009).

A fase de redução só inicia o processamento dos dados depois que todos os dados intermediários tenham sido copiados. Assim, os ambientes heterogêneos e voláteis podem afetar o comportamento do escalonamento de tarefas em diferentes áreas do modelo do *MapReduce*.

Para distribuir os dados da entrada o Hadoop leva em conta unicamente o tamanho do disco e a distância entre as máquinas. Quando ocorrem problemas de indisponibilidade de máquinas, uma grande quantidade de dados precisa ser transferida e há um *overhead* significativo na rede. O problema é crítico e afeta a performance do Hadoop (XIE et al., 2010). Os dados intermediários produzidos pelas tarefas *Map*, necessitam ser copiados para as máquinas para serem executadas as tarefas *Reduce*. Assim, não se pode aproveitar as características de localidade dos dados, devido ao número de chaves diferentes produzidas em cada *Map*. Em ambientes homogêneos, a distribuição de dados intermediários são geralmente heterogêneas que se traduzem em uma elevada complexidade para o tratamento de tarefas com tempos de execução muito diferentes (GUFLER et al., 2011).

2.9.2 Mecanismos de Tolerância a Falhas

Como apontado no trabalho de Lin (LIN et al., 2009), o sistema de arquivos distribuídos utilizado no Hadoop (HDFS - *Hadoop Distributed File System*) prevê o armazenamento de dados confiáveis através de mecanismos de replicação.

Porém, em sistemas voláteis, uma distribuição de *chunks* inadequada poderia ter um custo de replicação proibitivo para fornecer alta disponibilidade (LIN et al., 2009). Isto porquê, os dados intermediários produzidos na fase de *Map* não são replicados e quando uma máquina torna-se inacessível, as tarefas da próxima fase não serão executadas, ocasionando então re-execuções de tarefas e a degradação do sistema.

Um grande gargalo da implementação do Hadoop e do *MapReduce* é a falha do mestre. O gerenciamento de tarefas no *MapReduce* segue o modelo mestre/escravo. O *MapReduce* ainda não endereça o problema de falhas do mestre. No trabalho de (SALBAROLI, 2009) o autor propôs a construção de um *Fault tolerant Hadoop jobTracker* para agregar ao *jobTracker* um mecanismo de replicação.

2.9.3 Modelos de Aplicação

No trabalho de Mackey (MACKEY et al., 2008) os autores mostram a importância do uso do *MapReduce* para aplicações de grande volume de dados, como em simulações científicas, devido a sua simplicidade na programação, com alta disponibilidade e confiabilidade.

A maioria das aplicações científicas consistem em uma fase de simulação, geração de dados intermediários e uma fase de análise dos resultados dos dados, que são repetidas inúmeras vezes sobre uma mesma base de dados. Assim, o grande volume de dados e o paralelismo das tarefas são motivação para as aplicações científicas utilizarem o *MapReduce*. Entretanto, o uso deste *framework* com estes tipos de aplicações, ainda é uma questão aberta.

2.9.4 Problemas com Ambientes Voláteis

Poucos estudos tem sido feitos sobre a implementação do *MapReduce* em ambientes voláteis. Um dos principais problemas relaciona-se com a perda de dados e a necessidade de reexecuções. O modelo híbrido proposto por (LIN et al., 2009) tenta resolver o problema da volatilidade ao acrescentar máquinas fixas confiáveis (não voláteis) para armazenar dados temporários e manter réplicas.

Nesta arquitetura as máquinas voláteis recebem as tarefas para processamento e encaminham uma réplica de seus dados para a máquina não volátil. Esta abordagem é limitada em um ambiente de Computação Voluntária porque gera problemas de gargalos.

Na França um grupo de pesquisadores (TANG et al., 2010) apresentou uma implementação do *MapReduce* sobre a arquitetura do *BitDew* (FEDAK; HE; CAPPELLO, 2008) que é um *middleware* de gerenciamento de *desktop grid* que utiliza uma infra-estrutura P2P. Esta implementação possui seus próprios algoritmos para atender aos requisitos de processamento e altera a forma de distribuir os dados em relação ao *MapReduce* original.

2.9.5 Distribuição Estatística de Dados

Estudos feitos por pesquisadores de *Carnegie Mellon University* (KAVULYA et al., 2010) sobre o ambiente de produção de um *cluster* utilizando *MapReduce* identificam potencialidades de pesquisa, como por exemplo, melhorar o escalonamento de *jobs* longos ou explorar a localidade temporal através do uso de preditores para calcular os tempos de execução dos *jobs*.

O *MapReduce* segue uma distribuição de calda loga ou seja *Heavy-Tailed*. O uso de modelos de distribuição de probabilidade para calcular a previsão dos tempos de execução e atribuir uma característica dinâmica ao escalonamento do modelo pode ser uma alternativa interessante de pesquisa.

Com base em alguns dos problemas citados acima, principalmente os relacionados ao modelo, foram criados novos escalonadores e mecanismos de gerenciamento de tarefas para adequar o *MapReduce* ao uso em ambientes heterogêneos, que serão apresentados e detalhados em seção própria.

2.10 Outros Modelos de Computação Intensiva em Dados

Outros modelos surgiram para tratar a computação intensiva em dados em grande escala. Algumas das propostas utilizam abordagens parecidas com o *MapReduce*, aplicando o Hadoop para obter proveito de suas funções. Entretanto, outras apresentam uma semântica completamente diferente, com características próprias para o tratamento dos dados.

2.10.1 *All-PAIRS*

A abstração *All-Pairs* (MORETTI et al., 2010) é um modelo de um produto cartesiano de um grande número de objetos com uma função de comparação personalizada. *All-Pairs* é semelhante a outras abstrações como *MapReduce* (DEAN; GHEMAWAT, 2004) , *Dryad* (ISARD et al., 2007), *Phoenix* (RANGER et al., 2007), *Pig* (OLSTON et al., 2008) e Swift (WILDE et al., 2009), mas aborda uma categoria diferente de aplicativos.

O problema é expresso pela equação 2.6. *All-Pairs* compara todos os elementos de A com todos os elementos de B através de uma função $F(x)$ retornando uma matriz $M[i, j]$. Essa abstração também é conhecida como o produto cartesiano dos conjuntos A e B.

$$(A, B, F(x))|M[i, j] = F(A[i], B[j]) \tag{2.6}$$

As variações de *All-Pairs* ocorrem em muitos ramos da ciência e da engenharia, onde o objetivo é compreender o comportamento de uma função F recém-criada em conjuntos A e B, ou calcular a co-variância dos conjuntos A e B em uma norma interna do produto F. Entretanto, a função nem sempre é simétrica (MORETTI et al., 2010).

Um programa sequencial é fácil de ser construído para executar esta operação em um simples computador, porém o desempenho será muito lento. Entretanto, usar um *cluster* para efetuar a operação resulta em baixo aproveitamento dos recursos. Porque os dados não podem ser facilmente divididos em partições disjuntas. Todos os dados são necessários para as comparações serem feitas e, assim a máquina necessitará transferir dados em tempo de execução.

A proposta dos autores é sinalizar cada tarefa, composta por um par de elementos, a uma máquina e fazer as chamadas de leitura e escrita em um sistema de arquivos compartilhados sob demanda. O sistema não sabe qual é o tipo dos dados, até o momento em que a tarefa iniciar as chamadas ao sistema. Os usuários especificam os dados e a necessidade computacional. Assim, permite ao sistema particionar e distribuir os dados de acordo com a necessidade computacional dos dados distribuídos.

O sistema testa a entrada dos dados, para determinar o tamanho de cada elemento e o número de elementos em cada conjunto. Uma tarefa testa um pequeno conjunto de dados para determinar o tempo de execução típico para cada chamada.

A banda da rede e a latência do escalonador, para o envio de tarefas, são testadas para determinar o número de tarefas, o tamanho de cada tarefa e o número de máquinas necessárias. Os dados são entregues às máquinas via uma *spanning tree* e o fluxo de transferência se completa em um tempo logarítmico (MORETTI et al., 2010).

2.10.2 SAGA

O SAGA (***S**imple **A**PI for **G**rid **A**pplications*) apresenta uma interface de programação de alto nível que oferece a capacidade de criar aplicações distribuídas em *grids* e *clouds* (MICELI et al., 2009).

O modelo da arquitetura do SAGA é representado na Figura 2.11, adaptado de (MICELI et al., 2009). A implementação SAGA-*MapReduce* é relativamente mais complexa e naturalmente mais lenta que o *MapReduce* original. A aplicação consiste de 2 processos o *master* e o *worker*.

O *master* é responsável por gerenciar os dados, controlar réplicas, manipular dados intermediários e lançar as tarefas para os *workers*. Os *workers* unicamente tem a função de executar as tarefas.

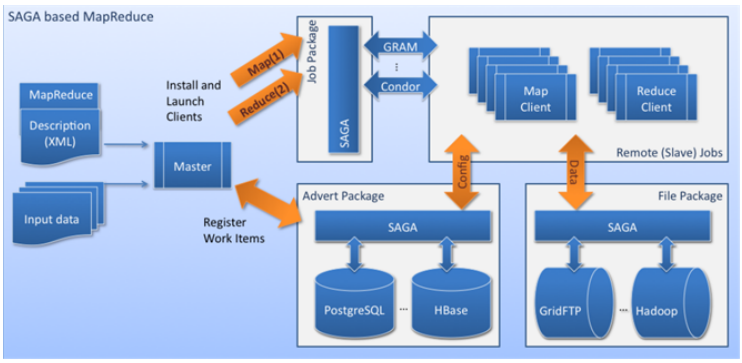


Figura 2.11: Arquitetura do SAGA usando *MapReduce*

No SAGA a implementação cria uma interface para o usuário. O código lógico recebe instruções explícitas para onde devem ser escalonados as tarefas. A aplicação não é obrigada a executar em um sistema que fornece a semântica originalmente exigida pelo *MapReduce*, assim pode ser portável para uma ampla gama de sistemas genéricos.

O SAGA pode lançar *jobs* e utilizar tanto a infra-estrutura do Globus/GRAM (FOSTER, 2005) como a do Condor (THAIN; TANNENBAUM; LIVNY, 2003) para execução das tarefas. A abstração do *MapReduce* é feita sobre a implementação do All-Pairs.

Os arquivos podem estar em qualquer tipo de sistema de aquivos distribuídos como por exemplo HDFS, Globus/GridFTP, KFS ou nos sistemas de arquivos locais das máquinas. O *master* cria partições de processos análogos aos *chunks* no *MapReduce*. A arquitetura é mestre/escravo e a comunicação é feita através do SAGA *advert package*. Uma das desvantagens do SAGA-*MapReduce* é não ser *multi-thread*.

2.10.3 Dryad

O *Dryad* é uma alternativa ao *MapReduce* proposta pela Microsoft (ISARD et al., 2007) que combina vértices computacionais com canais de comunicação para formar um grafo de fluxo de dados. Os *jobs* são grafos acíclicos orientados (dígrafos), onde em cada vértice há um programa e as arestas representam canais de dados.

A proposta consiste em dar ao desenvolvedor um controle fino sobre grafos de comunicação, bem como, sub-rotinas que exitem em seus vértices. O desenvolvedor pode especificar um grafo acíclico orientado arbitrariamente para descrever os parceiros de comunicação e expressar seus mecanismos de transporte.

A forma de especificar um grafo permite flexibilidade para compor as operações básicas com entradas de dados simples. As entradas de dados simples ou múltiplas geram uma saída simples. Entretanto, as aplicações exigem que o desenvolvedor deva compreender a estrutura computacional da computação e as propriedades dos recursos dos sistemas.

A Figura 2.12, adaptada de (ISARD et al., 2007), especifica a organização do sistema. Um *job* no *Dryad* é coordenado por um processo chamado gerenciador de *jobs* (JM).

O gerenciador de *jobs* contém os códigos e bibliotecas específicas das aplicações para construir a comunicação entre as tarefas. Quando a comunicação estiver definida serão escalonados os *jobs* através dos recursos disponíveis. Todo dado é enviado diretamente entre os vértices. O gerenciador de *jobs* é responsável somente pelo controle de decisões e não será um gargalo para qualquer transferência de dados. Um servidor de nomes (NS) fornece o suporte ao descobrimento de computadores disponíveis.

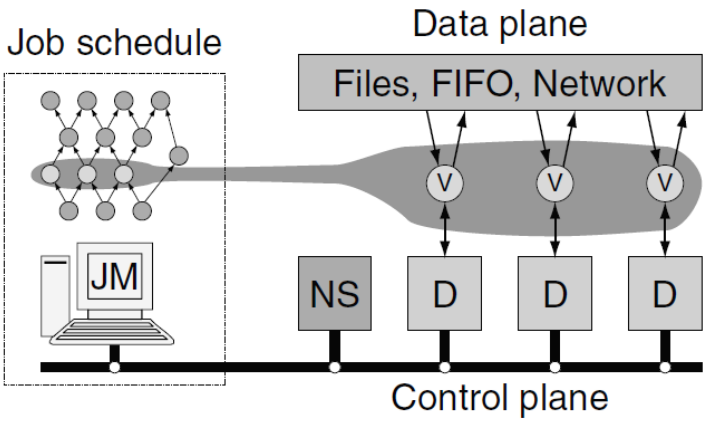


Figura 2.12: Arquitetura do *Dryad*

Um *daemon* (D) é responsável por criar os processos em favor do gerenciador de *jobs* em cada computador no cluster. A primeira vez que um vértice (V) é executado em um computador, seus binários são enviados do gerenciador de *jobs* para o *daemon*. O *daemon* atua como um *proxy* para o gerenciador de *jobs* poder comunicar o estado da computação e a quantidade de dados lidos ou escritos nos canais.

Os grafos são construídos pela combinação de modelos simples de um conjunto de operações. Todas as operações preservam a propriedade que resulta em um grafo acíclico. O objeto básico é dado pela equação 2.7, G contém uma sequência de vértices V_G, um conjunto de arestas E_G, e dois direcionadores $I_G \subseteq V_G$ e $O_G \subseteq V_G$ que indicam se o vértice é de entrada ou saída respectivamente. Nenhum grafo pode conter uma aresta inserindo uma entrada em I_G ou saída em O_G, estas indicações são utilizadas somente para as composições de operações.

$$\mathrm{G} = \langle V_G, E_G, I_G, O_G \rangle \tag{2.7}$$

A entrada e saída de arestas de um vértice são orientadas e uma aresta conecta a portas específicas em um par de vértices. Um par de vértices também pode estar conectado a múltiplas arestas. Cada canal de comunicação tem um protocolo associado. O canal é implementado usando um arquivo temporário: um produtor escreve no disco local (normalmente) e um consumidor lê deste disco o arquivo (ISARD et al., 2007).

Os protocolos de comunicação dos canais podem ser: arquivos (preservado até a conclusão da tarefa), TCP *pipe* (que não requer discos, mas deve ser escalonado fim-a-fim entre os vértices ao mesmo tempo) ou memória compartilhada FIFO (que tem baixo custo de comunicação, mas os vértices devem rodar no mesmo processo).

Os vértices do *Dryad* contém puramente código sequencial com suporte à programação baseada em eventos, usando um *pool* de *threads*. O programa e as interfaces dos canais tem forma assíncrona, embora seja mais simples utilizar interfaces assíncronas que escrever códigos sequenciais usando interfaces síncronas (ISARD et al., 2007).

2.10.4 SWIFT

O trabalho de (WILDE et al., 2009) apresenta um conceito diferente para o processamento em larga escala. A proposta consiste em explorar um modelo de programação paralela orientada ao fluxo de dados, que trata as aplicações como funções e o conjunto de dados como objetos estruturados. A técnica cria múltiplas cópias de um programa sequencial para executá-las ao mesmo tempo em um *script* dentro de um ambiente paralelo (BECKMAN et al., 2010).

O *Swift* combina a sintaxe da linguagem de programação C com características de programação funcional. A linguagem é projetada para expor as oportunidades de execução paralela, evitar a introdução de falta de determinismo, simplificar o desenvolvimento de programas que operam em sistemas de arquivos e permitir uma execução eficiente em computadores paralelos de memória distribuída. A arquitetura do *Swift* é apresentada na Figura 2.13, adaptado de (WILDE et al., 2009). A arquitetura de software é constituída de quatro camadas que são:

1. O *Swift*: é uma linguagem de *script* que coordena as tarefas, otimizações, gerenciamento de dados e reinicialização;

2. CDM (*Collective data management*): é responsável pela transmissão de grandes quantidades de dados de entradas comuns por *broadcast*, dispersão e união de pequenos arquivos;

3. *Falkon*: é um *dispatcher* de tarefas que usa a combinação de escalonamento *multithread* com uma arquitetura hierárquica, promovendo o balanceamento de carga;

4. *ZeptoOS*: é um sistema operacional Linux que fornece acesso para as primitivas *fork*() e *exec*() utilizadas no lançamento de novos programas.

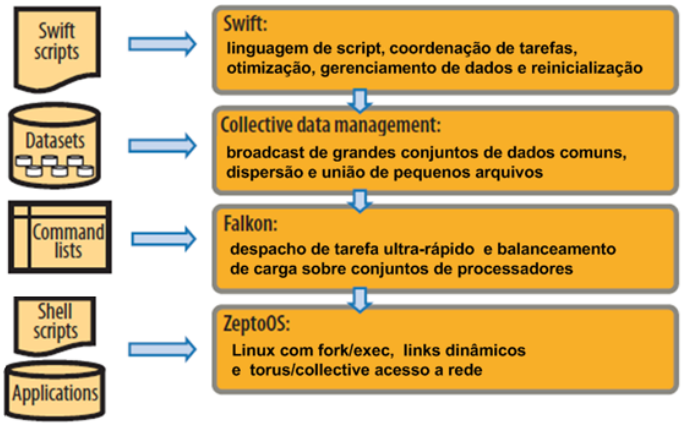


Figura 2.13: Arquitetura do *Swift*

Um programa produz os dados em um arquivo que, imediatamente, são consumidos por um segundo programa. O *Swift* assegura que para a variável compartilhada (representada pelo arquivo) não é atribuído nenhum valor até o primeiro programa completar sua execução. A quantidade de códigos necessários para expressar aplicações dessa forma é menor do que *scipts ad hoc* ou *shell scripts* e menos expressivos como anotações de grafos acíclicos orientados (dígrafos) da proposta do Dryad (WILDE et al., 2009).

3 MODELO DO *MR-A++*

Neste capítulo serão apresentadas as abordagens utilizadas na criação do *MR-A++*. Diversas modificações foram necessárias para alterar a estrutura do Hadoop e seus algoritmos afim de obter-se um desempenho aceitável, principalmente, em redes com baixa banda como a Internet. As adaptações realizadas nos algoritmos têm a premissa de que a quantidade de tarefas processadas é muito maior que a quantidade de máquinas disponíveis na *grid*.

3.1 Objetivos

Com base nos problemas apresentados nas seções 2.5 e 2.9, este trabalho tem por objetivo o desenvolvimento de novos algoritmos de distribuição de dados e de escalonamento de tarefas para o *MapReduce*. Os algoritmos adequam o uso do *framework* para ambientes heterogêneos, tanto para *clusters* como para *desktop grids*. Os *clusters* heterogêneos são formados quando ocorrem aquisições de máquinas com novas gerações de processadores ou com maior quantidade de recursos computacionais. A adequação para o ambiente heterogêneo é uma das necessidades para o estudo futuro do *Mapreduce* em presença de volatilidade.

No *MR-A++* os dados são distribuídos conforme a capacidade de processamento de cada máquina. A divisão dos dados pela capacidade computacional é automática e sem a necessidade de interferências do programador. As tarefas são então lançadas conforme uma política *task-assignment* (UCAR et al., 2006).

Não se exige do programador o conhecimento prévio da heterogeneidade das máquinas ou da banda da rede. Os algoritmos ajustam o lançamento de tarefas para obter-se o menor tempo de execução possível, assim evitando o desperdício de recursos computacionais.

3.1.1 Objetivos Secundários

Como objetivo secundário criou-se o MRSG (KOLBERG; ANJOS, 2011), um simulador *MapReduce* desenvolvido sobre a plataforma SIMGRID (SIMGRID, 2011) para o estudo deste *framework* em ambientes homogêneos e heterogêneos em larga escala. Este sub-projeto é justificado pela carência considerável de recursos necessários para o desenvolvimento de experimentos em grande escala.

O simulador MRSG, detalhado na seção 4.3, é um projeto concebido e executado pelo grupo de pesquisa *MapReduce*, formado a partir dos estudos realizados durante o desenvolvimento desse trabalho. Os algoritmos do Hadoop foram alterados com base em estudos feitos através do MRSG, visando a adequação ao ambiente heterogêneo.

3.2 Abordagens

O uso do *MapReduce* em ambientes de *clusters* heterogêneos, como apresentado na seção 2.5.2, tem baixo desempenho se comparado com ambientes homogêneos, assim como, na seção 2.5.3, verificam-se diversos problemas com o ambiente *desktop grid*. Então, a abordagem deste trabalho é de atuar em três áreas da implementação do modelo do *MapReduce*: agrupamentos, distribuição de dados e escalonamento de tarefas.

Agrupamentos são formados naturalmente no *MapReduce* quando marcam-se as máquinas com *status* de falha, lenta e normal. O fato de considerar uma máquina com falha não é um problema, pois ela é sinalizada assim para evitar o recebimento de tarefas que não serão executadas.

A diferença de capacidade computacional entre as máquinas no ambiente heterogêneo ocasiona um desbalanceamento de carga. As simplificações do modelo para o ambiente homogêneo, de que as tarefas executam em tempos iguais, resultam na existência de muitas máquinas consideradas como *stragglers*. Portanto, os algoritmos originais causam um aumento de tarefas especulativas e remotas em ambientes heterogêneos.

Para evitar que os tempos de execução sejam muito diferentes é necessário distribuir melhor os dados no *Map* e adequar o tamanho dos dados intermediários, distribuindo assim, uma quantidade de chaves compatível com a capacidade de processamento das máquinas na fase de *Reduce*. A estratégia fornece uma maior quantidade de dados para as máquinas mais rápidas. O tempo de execução das tarefas em uma máquina lenta é compatível com o tempo de execução de tarefas nas máquinas mais rápidas, assim balanceando o processamento entre elas.

Os novos algoritmos de distribuição de dados e o de agrupamentos consideram o tempo da execução das tarefas a partir de medições dos tempos locais antes dos dados serem efetivamente particionados. Assim, após o registro das máquinas um único *chunk* de 1 MB será distribuído para todas as máquinas na *grid*. O objetivo é processar uma tarefa de medição para o *Map* e o *Reduce* em cada máquina.

A tarefa de medição coletará informações de *hardware* das máquinas e os tempos das execuções de um *job*. O processo de execução da tarefa de medição ocorre antes da distribuição dos dados e em intervalos regulares durante o *job* para obter-se tempos médios de execução de um ambiente *desktop grid*.

O mecanismo de replicação, existente no sistema de aquivos distribuídos, influencia no desempenho do sistema. A replicação, além de fornecer um mecanismo de tolerância a falhas, possibilita lançar uma tarefa sem a necessidade de cópias de dados pela rede em tempo de execução. Porém, a quantidade de réplicas está limitada a um parâmetro de configuração do programador. Assim, replicar os dados conforme a capacidade computacional também poderá ajudar a evitar o lançamento de tarefas remotas que necessitem de cópias de dados.

O escalonamento de tarefas matem os dois mecanismos para melhorar o desempenho do *MapReduce*, o lançamento de tarefas remotas e o lançamento especulativo, porém os algoritmos serão adequados ao ambiente heterogêneo, com objetivo de reduzir a alocações de recursos desnecessários.

A cópia dos dados sobre a rede, durante o processamento de um *job*, altera o tempo da execução total do *job* e resulta em tempos de execução de tarefas nas máquinas muito diferentes umas das outras. O novo algoritmo de lançamento de tarefas remotas considera a banda da rede e os tempos de execução das tarefas das máquinas envolvidas, para evitar lançamentos de tarefas remotas que representem o aumento dos tempos de execução.

Após o processamento de uma tarefa *Map* os dados intermediários são copiados para

as máquinas que farão as tarefas *Reduces*. Neste momento, caso haja algum lançamento especulativo de tarefas *Map*, pode-se ter uma competição pelo barramento da rede, principalmente quando os *links* são lentos. O novo algoritmo de lançamento de tarefas especulativas minimiza a quantidade de tarefas especulativas para evitar esta competição.

No modelo original, uma máquina marcada como *straggler*, na primeira onda, não receberá uma nova tarefa em um segundo *slot* livre, favorecendo o lançamento de tarefas remotas (que precisam efetuar cópias de dados antes de sua execução), além de aumentar o número de tarefas especulativas. O resultado é um desempenho pobre do sistema e uso do barramento de rede para cópias desnecessárias de dados como poderá ser verificado na seção 4.

Em um ambiente heterogêneo muitas máquinas podem ser marcadas como lentas em um contexto que gera falso positivo. Por exemplo, uma máquina pode estar lenta se comparada a uma máquina que tenha três vezes a sua capacidade computacional, mas por outro lado, pode não estar se comparado a outra máquina com capacidade computacional semelhante.

Um novo agrupamento, denominado de *g_dist_bruta* foi criado para agrupar máquinas com mesma capacidade de processamento. O objetivo é de gerenciar o lançamento de tarefas através de máquinas com capacidades de processamento semelhantes. As tarefas em um agrupamento *g_dist_bruta* finalizam praticamente ao mesmo tempo. Assim, o progresso da execução das tarefas é comparado com máquinas de propriedades semelhantes.

Portanto, as máquinas que vierem a estar lentas, neste contexto, não geram falsos positivos e serão marcadas como *stragglers* porque realmente estão lentas para executar suas tarefas. A vantagem de não classificar uma máquina erradamente como *straggler* está no fato de que todos os *slots* desta máquina ficam indisponíveis para receber tarefas. Assim, a solução proposta no *MR-A++* permite um número menor de máquinas marcadas como *straggler* e um maior número de tarefas sendo executadas concorrentemente em um ambiente heterogêneo.

3.2.1 Distribuição dos Dados

No Hadoop, a máquina escrava registra-se no mestre para fornecer seu espaço em disco disponível para formar o sistema de arquivos distribuídos. O sistema de arquivos distribuídos irá então utilizar os discos de cada máquina para armazenar os *chunks*. Os dados, com os algoritmos originais, são distribuídos na mesma quantidade para as máquinas. Entretanto, esta abordagem não é suficiente para ambientes heterogêneos.

No modelo do *MR-A++*, a capacidade de processamento das máquinas é avaliada antes de se dividir os dados através do uso de dados coletados em uma tarefa de medição. Duas estratégias foram criadas para a distribuição dos dados. Na primeira, os dados são distribuídos antes de um cliente submeter um *job*. Na segunda, os dados são reorganizados quando já estiverem sido distribuídos previamente.

No novo modelo, a distribuição dos dados dependerá da entrada de dados, da capacidade computacional da máquina, do custo computacional da tarefa em relação à função *Map* ou *Reduce*, da latência de I/O (latência de memória, acesso ao disco e acesso à rede) e do tempo de cópia.

A tarefa de medição é executada sobre um mesmo *chunk* de 1 MB em todas as máquinas, antes dos dados serem distribuídos e periodicamente durante o processamento. A tarefa de medição é criada a partir do *job* submetido pelo usuário. O *chunk* de 1MB é um extraído dos próprios dados que serão repartidos no sistema de arquivos distribuído. Ao

término da execução da tarefa de medição, cada máquina escrava informa ao mestre: o tempo de execução gasto para processar o *Map*, denominado de $t_e Map$ dado em segundos; o tempo de execução gasto para processar o *Reduce*, denominado de $t_e Reduce$ dado em segundos; a capacidade computacional da máquina em *flops*; a latência da rede em segundos e a banda de rede em bits por segundo. A latência será medida na máquina por um teste de *ping* e a banda será obtida através de um teste de transferência de 1 MB de dados sobre a rede em cada máquina.

O mestre armazena as informações coletadas em um banco de dados para serem utilizadas nos algoritmos de distribuição, escalonamento e gerenciamento do progresso de tarefas. A capacidade computacional de processamento denominada de C, medida em *flops*, é obtida através de *benchmarks* como ocorre no BOINC (ANDERSON; MCLEOD, 2007). Os fatores considerados no *benchmark* são o número de núcleos do processador, a sua capacidade computacional e a memória disponível na máquina.

A execução periódica da tarefa de medição é feita após cada onda de execução do *job* corrente, como os dados já foram copiados previamente, tem-se um custo adicional mínimo, conforme será demonstrado na seção 4.5. Quando for submetido um novo *job* sobre um mesmo conjunto de dados, com uma função *Map* ou *Reduce* diferentes, uma nova tarefa de medição será realizada para obter-se novos dados. Caso os tempos de execução se alterem, os dados são movidos de uma máquina para outra afim de ajustar a capacidade de processamento de cada máquina.

O algoritmo de replicação de dados no sistema de arquivos foi alterado para dar uma réplica dos dados para as máquinas vizinhas pertencentes a uma mesma *g_dist_bruta* para diminuir o lançamento de tarefas remotas e controlar o lançamento de tarefas especulativas na fase de *Map*. O objetivo é minimizar o impacto do uso do barramento de rede com cópias de dados.

Entretanto, o desbalanceamento de carga natural que ocorre na função de *Map* devido aos dados intermediários, aumenta com esta abordagem e causa um problema de sobrecarga no uso da rede para as máquinas mais rápidas. O tempo gasto para copiar dados intermediários representa um peso considerável na fase de *Suffle*. Assim, o resultado é o maior uso do barramento de rede para a transferência destes dados.

Quando uma função de *Hash* é aplicada sobre os pares intermediários, ela utiliza o número de tarefas *Reduce* para dividir os dados. As tarefas *Reduces* são definidas pelo programador no modelo original. O número de *Reduces*, normalmente, é igual à quantidade de máquinas disponíveis. Assim, várias chaves intermediárias diferentes são distribuídas para uma mesma tarefa de *Reduce*. Então, é necessário que ao invés do número de *Reduces* a função de *Hash* utilize outro tipo de informação que reflita uma granularidade adequada.

A solução encontrada foi diminuir a granularidade das tarefas *Reduce*, o que representa uma maior quantidade de pequenas tarefas *Reduces* por máquina. Portanto, quando os pares intermediários forem distribuídos para as máquinas, existirá um menor número de chaves diferentes por tarefa *Reduce* para serem processadas. A abordagem de diminuir a granularidade dos dados intermediários é semelhante ao algoritmo de partições finas apresentado por Gufler (GUFLER et al., 2011).

As chaves contidas nos dados intermediários são particionadas de forma que todas as chaves iguais fiquem na mesma partição. O número de tarefas *Reduce* serão definidas por um fator de granularidade, chamado de Fg, calculado a partir dos dados coletados durante a tarefa de medição. O número de partições dos dados intermediários será maior que a quantidade de máquinas da *grid*. O fator Fg irá substituir a definição do número de

tarefas *Reduces* de forma automática. O resultado será um maior número de tarefas curtas sobre os mesmos dados produzidos pela fase de *Map*.

As tarefas são colocadas no mestre em uma fila, com a localização dos dados intermediários na rede. As máquinas quando estiverem livres requisitam as tarefas *Reduce* por *heartbeat* para serem processadas. Com uma quantidade menor de chaves diferentes por tarefa ocorre uma maior adequação à capacidade de processamento das máquinas.

Caso a aplicação exija que a função *Reduce* tenha somente uma única redução, o programador deverá indicar na submissão do *job*. Assim, para este caso específico, os algoritmos de particionamento dos dados intermediários do *Reduce* não se aplicam.

3.2.2 Agrupamentos

Um agrupamento *g_dist_bruta* é um grupo de máquinas com mesma capacidade computacional que recebem o mesmo número de *chunks* para ser processado. Espera-se que as máquinas em um mesmo agrupamento terminem suas tarefas ao mesmo tempo, embora, como no modelo original, não se possa garantir. Assim é análogo dizer que as máquinas na fase de *Map* com uma mesma *dist_bruta* terminam suas tarefas aproximadamente com o mesmo tempo de execução, num ambiente heterogêneo.

O progresso de execução de uma tarefa determina se uma máquina é considerada lenta ou não. Assim, com o *g_dist_bruta* o lançamento de tarefas especulativas é controlado pelo tempo médio das execuções das máquinas pertencentes a cada grupo e não mais pela média das execuções de todas as máquinas, portanto esta estratégia permite reduzir falsos positivos e escolher uma máquina mais rápida que a corrente para executar uma tarefa.

Entretanto, para a função *Reduce*, os tempos das tarefas podem ser muito diferentes. Isto ocorre porquê as tarefas *Map* podem gerar maior ou menor quantidade de chaves intermediárias, conforme a função de mapeamento e/ou a entrada de dados. Assim, as tarefas *Reduce* podem processar uma maior ou menor quantidade de dados e o tempo de execução pode variar. Portanto, o uso da abordagem de agrupamentos por *g_dist_bruta* não tem o mesmo resultado no *Reduce*.

No *MapReduce* o número de tarefas é muito maior que a quantidade de máquinas disponíveis para o processamento. Assim, inspirado no modelo de temporização de tarefas (COTTET et al., 2002), é possível considerar o escalonamento de tarefas do *MR-A++* como mostra a Figura 3.1.

As tarefas *Map* têm um tempo para a execução de um *chunk* em cada máquina, obtido dos valores de $t_e Map$. Em um ambiente heterogêneo, cada máquina terá um $t_e Map$ diferente para a execução de um *chunk*. Na máquina mais lenta da *grid* este tempo é chamado de t_e. Assim, o tempo para executar um *chunk* está dentro de um intervalo $[t_0, d_f]$. O tempo máximo da execução de um ou mais *chunks* é dado por d_f, sendo $d_f \leq t_e$.

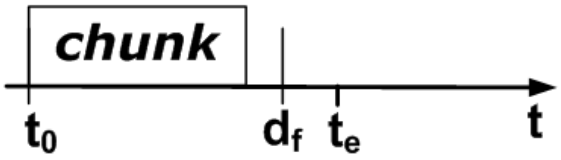


Figura 3.1: Modelo de temporização para tarefas

Uma máquina poderá processar um ou mais *chunks* em um mesmo t_e, conforme sua capacidade computacional. Uma *dist_bruta* (D) representa uma proporção de *chunks* que a máquina pode processar em função da sua capacidade computacional C. Para cada

máquina M_j existe uma trinca com um valor para C_j, D_j e d_f. Os valores associados à máquina $M_j(C_j, D_j, d_f)$ governam o escalonamento das tarefas do ***MR-A++***.

No *MapReduce* é definido um número de *slots* livres que representa quantas tarefas simultâneas podem ser executadas em uma máquina. O escalonador calcula a proporção de *chunks* para ser distribuída para cada máquina, de uma *dist_bruta* (D), como mostra a Equação 3.1, baseado nas informações coletadas na tarefa de medição.

- t_e é o tempo de execução de uma tarefa *Map* na máquina mais lenta;
- $\varepsilon_{h \to r}$ é o tempo de transferência de um *chunk* de uma máquina local para uma máquina remota. Se a tarefa for executada na máquina local $\varepsilon_{h \to r} = 0$;
- $t_e Map$ é o tempo da execução da tarefa de *Map* na máquina local;
- C_{total} é a soma das capacidades computacionais de todas as máquinas na *grid*;
- Num_chunks é a quantidade total de *chunks* a ser distribuída.

$$D = \lceil \frac{C}{C_{total}} * Num\_chunks \rceil \tag{3.1}$$

O valor de uma *dist_bruta* está entre $\frac{t_e}{(t_e Map_j + \varepsilon_{h \to r})} \leqslant D \leqslant Num\_chunks$, assim a máquina com a menor capacidade computacional deve executar pelo menos um *chunk* para cada *slot* no intervalo de tempo t_e, para obter-se o uso eficiente dos recursos computacionais. A máquina com maior capacidade irá executar o maior número possível de *chunks* neste mesmo intervalo.

O algoritmo 3.4.3 apresentado na subseção 3.4.2 detecta se a execução de uma tarefa em uma máquina muito lenta irá prejudicar o tempo de execução de todo o *job*. Se isto for ocorrer, o algoritmo irá alterar o valor da *dist_bruta* para zero e nenhum *chunk* ou tarefa será atribuída a esta máquina.

3.2.3 Escalonamento de Tarefas

As tarefas no *MapReduce* estão relacionadas com a localidade dos dados e são lançadas primeiro em máquinas que contenham uma réplica local dos dados. Portanto, num ambiente de *clusters* homogêneos onde a rede tem grande disponibilidade de banda, pode obter-se o aumento de desempenho ao serem lançadas tarefas remotas e especulativas.

Entretanto, em ambientes heterogêneos, o escalonador marca muitas máquinas como *stragglers* embora não o sejam. Assim, o escalonador lança uma quantidade de tarefas que é incompatível com a capacidade de processamento das máquinas, ocasionando a geração de muitas tarefas remotas e especulativas.

Quanto mais adequada for a distribuição de *chunks* à capacidade de processamento das máquinas, em um ambiente heterogêneo, menor será o lançamento de tarefas remotas e especulativas. Caso sejam distribuídos dados insuficientes para a capacidade de processamento das máquinas ou existam pequenas variações no tempo de execução das tarefas, o escalonador ainda poderá lançar tarefas pendentes em máquinas remotas. A Figura 3.2 ilustra este problema existente no *MapReduce* original em ambientes heterogêneos.

Por exemplo, 11 tarefas devem ser executadas por 4 máquinas. As tarefas são lançadas quando as máquinas estiverem livres. Os *chunks* são distribuídos de forma a manter-se o balanceamento de carga. Assim, as máquinas tem a seguinte distribuição dos dados: Máquina 2 (*chunks* 2, 4, 8 e 9), Máquina 3 (*chunks* 1, 5, 10 e 11), Máquina 4 (*chunks* 3, 6, 8 e 11) e Máquina 6 (*chunks* 4, 7, 8 e 11).

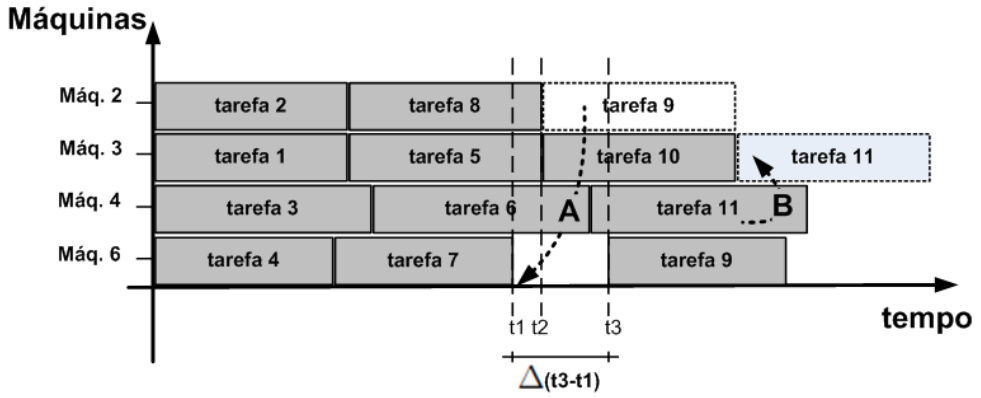


Figura 3.2: Problemas no lançamento de tarefas

As máquinas ao ficarem livres enviam uma mensagem de *heartbeat* para o mestre solicitando uma tarefa para ser executada. Na primeira rodada as máquinas recebem uma tarefa cada uma. As máquinas, sendo heterogêneas, executam as tarefas em tempos diferentes. Assim, o progresso de uma tarefa em cada uma é diferente. Um lançamento remoto pode ocorrer no ponto A, quando a tarefa 9, no instante t1, necessitar ser lançada. Porém a máquina 6 não tem os dados no disco local, assim a máquina 2 deve transferir uma cópia de seus dados para serem processado pela máquina 6.

Embora a máquina 6 leve menos tempo para executar uma tarefa que a máquina 2, o tempo total para o processamento da tarefa, dado pela soma do tempo de transferência dos dados ($\Delta(t_3 - t_1)$) mais a execução da tarefa 9, é maior que o tempo restante para a conclusão da tarefa 8 ($t_2 - t_1$) na máquina 2 somado ao tempo de execução da tarefa 9 na mesma máquina. Assim, o escalonador de tarefas do *MR-A++* antes de lançar uma tarefa remota, verifica se vale a pena lançar a tarefa remota ou aguardar a conclusão da tarefa corrente e evitar a transferência de dados pela rede.

Após o lançamento das tarefas pendentes, se a máquina 4 for considerada uma máquina lenta, então será lançada uma tarefa especulativa, como no ponto B, em outra máquina livre que contenha uma réplica dos dados para executar a tarefa 11, como por exemplo a máquina 3, da Figura 3.2 ou, até mesmo, uma tarefa especulativa remota.

O lançamento especulativo remoto só acontece no *Map*, quando é necessário lançar uma tarefa especulativa em uma máquina que não contém uma réplica dos dados locais. No caso do exemplo da Figura 3.2, o lançamento especulativo iria consumir recursos da rede e do processador da máquina que receber a tarefa.

O novo escalonador calcula uma previsão para a conclusão da tarefa, através do tempo médio das execução das tarefas de cada agrupamento. Então, o tempo médio calculado é utilizado para gerenciar progresso e o lançamento das tarefas. O resultado é a redução do número de lançamento de tarefas especulativas na fase de *Map*.

O lançamento de tarefas depende diretamente do algoritmo de distribuição de dados para obter-se um menor volume de cópias de dados em tempo de execução. Assim, quanto menor for a banda disponível melhor será o resultado dos novos algoritmos sobre a distribuição de dados no *Map*.

A quantidade de dados intermediários produzidos na fase de *Map* pode variar conforme a densidade da entrada. O custo para determinar a quantidade e o número de chaves de todos os *chunks*, em tempo de execução, é alto. Assim, para ajustar a quantidade de dados na fase de *Reduce* à capacidade de processamento de cada máquina, foi necessário diminuir a granularidade dos dados intermediários.

As tarefas *Reduce* são colocadas em uma fila no mestre e as máquinas livres vão solicitando tarefas para serem executadas. Uma menor quantidade de pares intermediários por tarefa diminui a diferença do tempo total das execuções das tarefas *Reduce* como é ilustrado na Figura 3.3. Reduzir esta diferença implica em aumentar o paralelismo das tarefas e resulta na redução do tempo global do *job*.

A abordagem faz com que os dados transferidos pelo mecanismo de *prefetch* sejam menores. Porém, o processamento de um pequeno número de grandes agrupamentos de chaves leva muito mais tempo para ser executado do que o processamento de muitos pequenos agrupamentos, conforme constata (GUFLER et al., 2011).

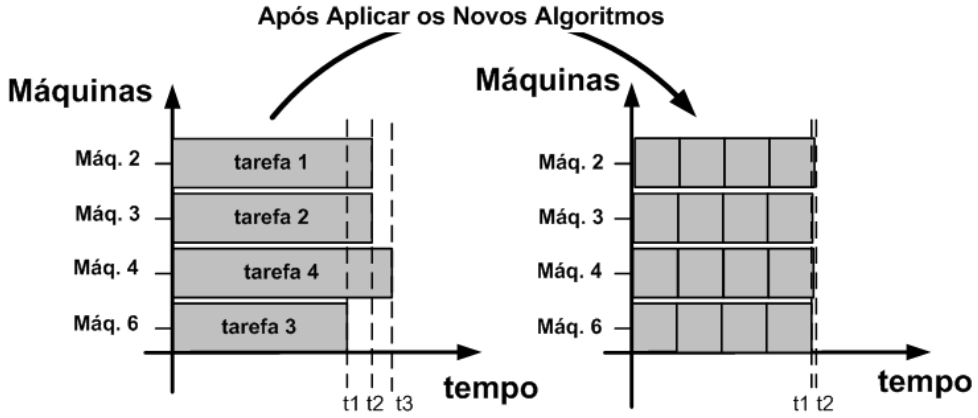


Figura 3.3: Temporização de tarefas *Reduce*

3.3 Arquitetura do *MR-A++*

Para adaptar o modelo do *MapReduce* ao ambiente heterogêneo foram feitas diversas modificações nos algoritmos existentes. Novos módulos foram criados para dar suporte ao escalonador de tarefas e ao mecanismo de distribuição de dados. Os módulos em branco são os que foram alterados em relação ao modelo original. A Figura 3.4 apresenta os diversos módulos que compõe o *MR-A++*.

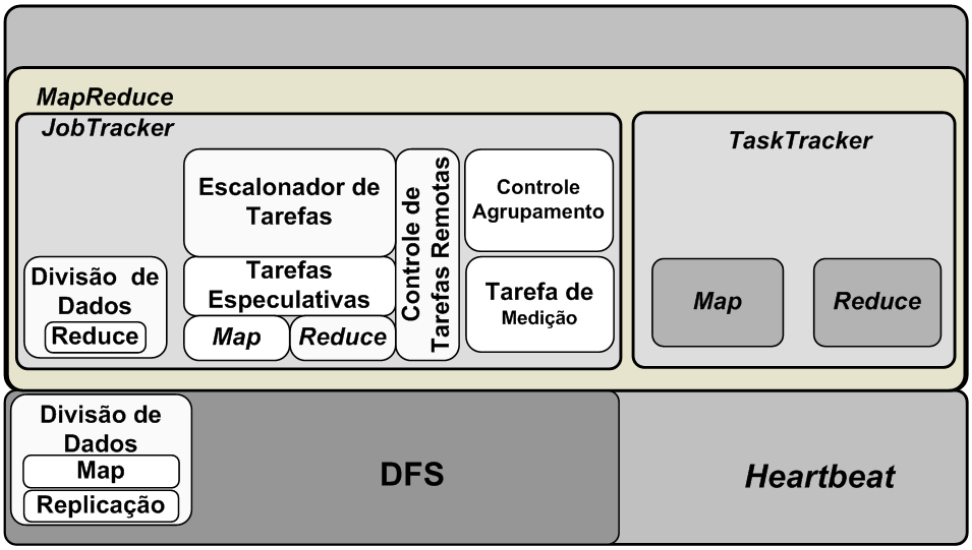


Figura 3.4: Arquitetura do *MR-A++*

- **Módulo de Divisão de Dados**: é responsável pela divisão dos dados de *Map* e *Reduce* conforme a capacidade computacional das máquinas. A divisão dos dados do *Map* é executada sobre o sistema de arquivos distribuídos. No *Reduce*, a divisão dos dados é um processo executado no mestre, que permite a criação de uma fila de tarefas. As filas serão gerenciadas pelo módulo de escalonamento de tarefas. O sub-módulo de replicação dá suporte à replicação de dados do *Map* respeitando o agrupamento das máquinas;

- **Módulo Escalonador de Tarefas**: é o módulo que controla o escalonamento das tarefas de *Map* e *Reduce*. Uma função importante neste módulo é o controle de do progresso das tarefas. O controle de tarefas especulativas foi dividido em dois módulos, um para o *Map* e outro para o *Reduce*. O módulo de *Controle de Tarefas Remotas* controla o lançamento de tarefas remotas;

- **Módulo Controle de Agrupamento**: é responsável pela geração dos agrupamentos conforme a capacidade de computacional das máquinas. Os agrupamentos juntamente com os tempos de tarefas governam o controle das execuções de tarefas;

- **Módulo de Tarefa de Medição** : é o módulo que organiza, controla e distribui os dados para a execução da tarefa de medição. As informações coletadas neste módulo são vitais para a divisão de dados e o controle das execuções de tarefas;

3.4 Detalhamento dos Algoritmos de Divisão de Dados

Distribuir os dados conforme a capacidade computacional objetiva dar uma quantidade de dados adequada ao processamento de cada máquina. O balanceamento de carga irá permitir que o progresso da execução de tarefas ocorra em tempos aproximadamente iguais entre as máquinas de cada *g_dist_bruta*. Assim, espera-se um número menor de lançamentos de tarefas remotas e especulativas, resultando em uma menor competição pelo barramento de rede entre as máquinas, durante a execução do *MapReduce*.

Na fase de *Reduce* a distribuição dos dados intermediários com menor quantidade de chaves diferentes por tarefa, como pode ser constatado na seção 4.5, resulta na redução dos tempos de execução das tarefas. Máquinas mais rápidas irão processar uma maior quantidade de dados que máquinas mais lentas. Portanto, se obtém um balanceamento de carga através da distribuição de dados adequada à capacidade de processamento das máquinas. Nesta seção serão detalhados os algoritmos que foram criados para atingir o objetivo de adequar o *MapReduce* ao ambiente heterogêneo, conforme as estratégias apresentadas anteriormente.

3.4.1 Tarefa de Medição

Quando um cliente submete um *job*, ele indica a localização dos dados para serem copiados para o sistema de arquivos distribuídos. Antes dos dados serem distribuídos, um mecanismo faz a cópia de 1 MB dos dados e cria um *chunk* para ser distribuído a todas as máquinas. Após então, envia uma tarefa de *Map* e *Reduce* para coletar seus tempos de processamento. O valor de 1 MB de dados foi definido em função do tempo necessário para a transferência de dados para as máquinas. Os tamanhos de arquivos avaliados foram de 64, 32, 16, 8 e 1 MB. O melhor custo/benefício com baixo *overhead* foi obtido com 1 MB de dados.

Afim de evitar uma sobrecarga na transferência de dados, o algoritmo 3.4.1 é iniciado para executar a distribuição dos dados e da tarefa de medição. O algoritmo é inspirado no modelo do Algoritmo *k-d Tree*, que utiliza o método da mediana (FLOYD; RIVEST, 1975) para definição dos níveis da árvore. Uma árvore de n máquinas e m níveis pode ser construída com tempo $O(n \log m)$ (AL-FURAJH et al., 2000).

O algoritmo 3.4.1 cria uma árvore a partir do cálculo da mediana, linhas 2 a 5 do algoritmo. O Md é a mediana da soma dos índices das potências da decomposição binária do número de máquinas ou Num_hosts. Tam_tree é a soma dos valores de $2^{Md}+2^{Md+1}+ .... + 2^i$ potências decompostas a partir do índice da mediana e representa a quantidade de máquinas possíveis de serem acomodadas na árvore (linha 3). Se o número de máquinas for maior que Tam_tree, então a quantidade de níveis deverá ser $Md + 2$ para acomodar todas as máquinas.

A altura mínima é igual a $Md + 1$ e a máxima é igual a $Md + 2$. A árvore é construída a partir da raiz, onde fixa-se o mestre. Então, cada nível a partir da raiz tem 2^A máquinas, sendo $A \in \{Md, Md + 1, Md + 2, ..., i\}$ o número máximo de máquinas por nível. Um número de 2^{Md} máquinas são colocadas no primeiro nível e, em cada nível seguinte, cada máquina pode ter no máximo 2 filhos. A soma de máquinas em todos os níveis, excluindo-se a raiz, será a quantidade de Num_hosts, linhas 6 a 7 do algoritmo 3.4.1.

Após a árvore estar completa, o mestre distribui um *chunk* de 1 MB para os seus filhos. Então, cada filho irá replicar uma cópia do *chunk* recebido para seus descendentes e assim sucessivamente, até que todas as máquinas tenham recebido um *chunk*, linhas 8 a 10 do algoritmo 3.4.1. Quando todas as máquinas receberem um *chunk* o *job* de medição é enviado.

A Figura 3.5 mostra a criação de uma árvore para 31 máquinas, seguindo o algoritmo 3.4.1. Os passos para a formação da árvore e a execução da tarefas de medição foram detalhados para o melhor entendimento do algoritmo, considerando a distribuição de um *chunk* de 1 MB de dados, conforme a seguir:

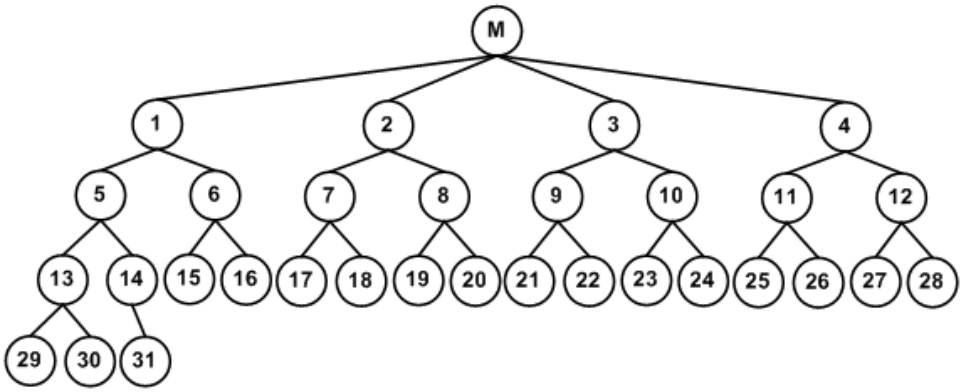


Figura 3.5: Construção da árvore para a tarefa de medição

1. Fatorar o valor de 31 em potência de 2 e encontrar o maior índice para 2^i, neste caso 2^4, assim $i = 4$;

2. Calcular $Md = (1 + 2 + 3 + 4)/4 = 2,5 \therefore Md = 2$

3. Calcular $Tam\_tree = 2^2 + 2^3 + 2^4 = 28 \therefore Tam\_tree < 31 \therefore$ Níveis$= 4$

4. Cria-se a árvore com 4 níveis, da seguinte forma:

(a) A partir de do mestre, o primeiro nível é $2^2 = 4$ máquinas;

(b) O próximo nível é $2^3 = 8 \therefore 4 + 8 = 12$ máquinas;

(c) O próximo nível é $2^4 = 16 \therefore 4 + 8 + 16 = 28$ máquinas;

(d) No último nível colocam-se as 3 máquinas restantes.

5. O mestre copia o *chunk* de 1 MB para seus filhos.

6. Em cada nível as máquinas replicam o *chunk* para seus sucessores.

7. Após todos os *chunks* terem sido copiados, o mestre envia a todas as máquinas as tarefas de *Map* e *Reduce* para serem executadas.

Algoritmo 3.4.1 Tarefa de medição

1. Calcular maior potência de 2^i para Num_hosts
2. $Md = \frac{\sum_{a=1}^{i} a}{i}$ {Calcula a mediana dos índices de 2^i.}
3. $Tam\_tree = \sum_{a=Md}^{i} 2^a$
4. **if** $Tam\_tree < Num\_hosts$ **then**
5. Níveis= $Md + 2$
6. Fixar o mestre na raiz da árvore e colocar 2^{Md} máquinas no primeiro nível.
7. Colocar 2 filhos por máquina em cada nível até ter-se Num_hosts filhos do mestre.
8. **for** $j \leftarrow 1$ to 2^m **do**
9. $Send(chunk, Host_j)$ {Envia o *chunk* às máquinas.}
10. Cada máquina pai no nível $Md + 1$ replica o *chunk* para seus descendentes.
11. **for** $j \leftarrow 1$ to Num_hosts **do**
12. $Send(taskMap, Host_j)$ {Tarefa de Medição *Map*.}
13. $Send(taskReduce, Host_j)$ {Tarefa de Medição *Reduce*.}

Cada máquina retorna por *heartbeat* um vetor com os valores de: o volume do disco (Vol_{hd}); o tempo de execução da tarefa *Map* ($t_e Map$); o tempo de execução da tarefa *Reduce* ($t_e Reduce$); a capacidade computacional da máquina (C_j); a banda da rede (Bw_j) e a latência da rede (Lat_j). Os dados coletados são armazenados em uma base de dados no mestre para o gerenciamento do escalonador.

A tarefa de medição é executada antes dos dados serem distribuídos para o sistema de arquivos e em intervalos periódicos, após ocorrer a primeira onda de tarefas *Map*. O objetivo é poder corrigir possíveis alterações no comportamento das máquinas em uma *desktop grid*. Os custos das novas execuções são baixos porque os dados já foram distribuídos, somente existem os custos associados com o processamento das tarefas de *Map* e *Reduce*. Então, os dados coletados são armazenados em uma base de dados no mestre, com tempos médios de execução que serão utilizados durante o gerenciamento do progresso das tarefas.

3.4.2 Algoritmo da Divisão de Dados para o *Map*

O algoritmo 3.4.2 é utilizado para a divisão dos dados. O novo algoritmo faz uma tarefa de medição antes de serem distribuídos os dados às máquinas no sistema de arquivos. Nas linhas 1 a 3 verifica-se a execução da tarefa de medição e o recebimento das informações pelo mestre.

Nas linhas 4 a 6, o mestre calcula o valor da *dist_bruta* e aplica o algoritmo 3.4.3, para formar os agrupamentos e após, nas linhas 9 a 11, dividir os dados conforme a capacidade computacional das máquinas.

O algoritmo de replicação, linhas 12 a 14, diferente do modelo original que replica os dados aleatoriamente, envia réplicas somente aos seus vizinhos com mesma *dist_bruta*. O objetivo é evitar os problemas discutidos na subseção 3.2.3.

Algoritmo 3.4.2 Divisão de dados *Map*

```
1. Executa Tarefa de Medição
2. for j ←1 to Num_hosts do
3.     Receive ( [t_e Map; t_e Reduce; C; Bw; Lat; Vol_hd], Host_j)
4.     t_e = t_e Map_j {Procura a máquina mais lenta}
5.     D_j = ⌈ C / C_total * Num_chunks ⌉ {Cálculo da dist_bruta}
6. Executa Algoritmo menor_MAIOR
7. while t < Num_chunks do
8.     for j ← 1 to Num_hosts do
9.         if D_j > 0 then
10.            for k ← 1 to D_j do
11.                Host_j ← chunk_j {Distribui os dados pela Capacidade Processamento}
12.                while r < Num_replica do
13.                    Procura Host_{j+1} com mesma D_j
14.                    Host_{j+1} ← chunk_j {Máquinas vizinhas recebem uma réplica do chunk}
```

3.4.3 Algoritmo menor_MAIOR

O algoritmo menor_MAIOR foi criado para identificar se a execução de uma tarefa em uma máquina muito lenta irá prejudicar o tempo de execução de todo o *job*. O objetivo principal é identificar as máquinas mais lentas para a execução de uma tarefa no *job* e diminuir a carga de trabalho a ser distribuída para eles. Outra finalidade é adequar a quantidade de *chunks* calculada no processo com a quantidade real de *chunks* que devem ser processados. A quantidade maior pode ocorrer devido a erros de arredondamento na fase de inicial do algoritmo.

Por exemplo, um *job* é submetido com 1.280 MB de dados para ser processado por cinco máquinas heterogêneas. As máquinas tem capacidade computacional conforme a coluna Capac da Tabela 3.1 (a). O percentual da capacidade computacional de cada máquina em relação à capacidade total é chamado de % Capac. A quantidade de dados equivale a 20 *chunks*, com 64 MB por *chunk*. Assim, toma-se o número total de *chunks* e multiplica-se por esta proporção para obter-se o valor de *chunks* que serão processados por cada máquina. A Tabela 3.1 (a) apresenta na coluna D o valor para este cálculo, por exemplo 20 x $0,7503 = 15$, 20 x $0,1327 = 2,65$ s (arredonda-se para o maior inteiro, no caso 3) e assim por diante. A coluna D indica a *dist_bruta* calculada para cada máquina. A última linha da coluna apresenta a soma dos valores obtidos para D.

Após, é necessário calcular-se os valores de *prev_exec* e *temp_corr*. O valor de *prev_exec* é igual à multiplicação do valor de D_j por $t_e Map$, que resulta na previsão do tempo de execução, em segundos, por exemplo 15 x $0,345 = 5,1750$ s, ou seja, 15 *chunks* seriam processados em 5,175 segundos. Os valores calculados são colocados na coluna *prev_exec* na Tabela 3.1 (a). O valor de *temp_corr* é obtido com a soma de *prev_exec* mais uma unidade de $t_e Map$, ou seja *temp_corr* = $5,1750 + 0,3450 = 5,5200$ s. Os valores calculados estão na coluna *temp_corr* da Tabela 3.1 (a). O objetivo é identificar se pode ser movido um *chunk* para uma máquina, sem aumentar o tempo total das execuções. Na última linha da Tabela 3.1 (a), na coluna *prev_exec* calcula-se o maior valor de *prev_exec* e na coluna *temp_corr* calcula-se o menor valor de *temp_corr*.

O algoritmo 3.4.3 tem duas rodadas. Na primeira rodada, verifica-se o número total de D. Se a soma do valor calculado para D é maior que a quantidade de *chunks*, en-

tão, remove-se uma unidade de D, da linha com o maior valor de *prev_exec*. Os valores de *prev_exec* e *temp_corr* da linha modificada são recalculados. Após compara-se novamente os valores do maior *prev_exec* e do menor *temp_corr* respectivamente. O processo se repete até que o valor da soma de D seja igual ao número de *chunks*.

Tabela 3.1: Exemplo da execução do Algoritmo menor_MAIOR

(a) Início

Máquina	$t_e Map$ **(s)**	**Capac (*Mflops*)**	**% Capac.**	D	***prev_exec* (s)**	***temp_corr* (s)**
1	0,3450	14.780	0,7503	15	5,1750	5,5200
2	1,9510	2.614	0,1327	3	5,8530	7,8040
3	4,2827	1.190	0,0604	1	4,2827	8,5655
4	9,1269	558	0,0283	1	9,1269	18,2538
5	9,1269	558	0,0283	1	9,1269	18,2538
		19.700		21	9,1269	5,5200

(b) Primeira Rodada

Máquina	$t_e Map$ **(s)**	**Capac (*Mflops*)**	**% Capac.**	D	***prev_exec* (s)**	***temp_corr* (s)**
1	0,3450	14.780	0,7503	15	5,1750	5,5200
2	1,9510	2.614	0,1327	3	5,8530	7,8040
3	4,2827	1.190	0,0604	1	4,2827	8,5655
4	9,1269	558	0,0283	1	9,1269	18,2538
5	9,1269	558	0,0283	0	0	9,1269
		19.700		20	9,1269	5,5200

(c) Segunda Rodada

Máquina	$t_e Map$ **(s)**	**Capac (*Mflops*)**	**% Capac.**	D	***prev_exec* (s)**	***temp_corr* (s)**
1	0,3450	14.780	0,7503	16	5,520	5,8650
2	1,9510	2.614	0,1327	3	5,8530	7,8040
3	4,2827	1.190	0,0604	1	4,2827	8,5655
4	9,1269	558	0,0283	0	0	9,1269
5	9,1269	558	0,0283	0	0	9,1269
		19.700		20	5,8530	5,8650

Na segunda rodada, verifica-se a diferença entre o maior *prev_exec* e o menor *temp_corr*. Se o menor inteiro da diferença for maior que um, remove-se uma unidade de D, da linha com o maior valor de *prev_exec* e adiciona-se uma unidade ao D na linha com menor *temp_corr*. Recalcula-se *prev_exec* e *temp_corr* das linhas modificadas, calcula-se novamente o maior *prev_exec* e o menor *temp_corr*. O processo se repete até que o menor inteiro seja zero.

O algoritmo 3.4.3 apresenta o algoritmo menor_MAIOR . Nas linhas 2 a 6 ocorre o processo de inicialização dos valores, apresentados na Tabela 3.1a. A primeira rodada do algoritmo 3.4.3, linhas 7 a 13, redistribui os *chunks* e faz as correções do número de *chunks*. A segunda rodada do algoritmo, linhas 14 a 17, as máquinas lentas que comprometem o tempo de execução de todo o *job* são excluídos do processamento. Nas linhas 18 a 21 do algoritmo 3.4.3, após ajustados os valores para D_j, identifica-se as máquinas que tenham o mesmo D_j para formar os agrupamentos (*g_dist_bruta*), que serão utilizados para o gerenciamento dos demais algoritmos.

Algoritmo 3.4.3 Alg. menor_MAIOR

1. **for** $j \leftarrow 1$ to Num_hosts **do**
2. $\quad prev\_exec_j = t_e Map_j * D_j$
3. $\quad temp\_corr_j = t_e Map_j + prev\_exec_j$
4. $\quad Maior\_prev\_exec = \text{Max}(prev\_exec_j)$
5. $\quad menor\_temp\_corr = \text{Min}(temp\_corr_j)$
6. $Soma\_dist\_bruta = \sum_j D_j$
7. **while** $Soma\_dist\_bruta \neq Num\_chunks$ **do**
8. $\quad$ **if** $Soma\_dist\_bruta > Num\_chunks$ **then**
9. $\quad\quad D_{Host\_Maior\_prev\_exec} = D_{Host\_Maior\_prev\_exec} - 1$
10. $\quad$ Recalcula o Valor $Maior\_prev\_exec$ e D
11. $\quad$ **if** $Soma\_dist\_bruta < Num\_chunks$ **then**
12. $\quad\quad D_{Host\_menor\_temp\_corr} = D_{Host\_menor\_temp\_corr} + 1$
13. $\quad$ Recalcula o Valor $Maior\_prev\_exec$ e D
14. **while** $\lfloor Maior\_prev\_exec - menor\_temp\_corr \rfloor > 0$ **do**
15. $\quad D_{Host\_Maior\_prev\_exec} = D_{Host\_Maior\_prev\_exec} - 1$
16. $\quad D_{Host\_menor\_temp\_corr} = D_{Host\_menor\_temp\_corr} + 1$
17. $\quad$ Recalcula o Valor $Maior\_prev\_exec$ e $menor\_temp\_corr$
18. **for** $j \leftarrow 1$ to Num_hosts **do**
19. $\quad$ **while** $i < Num\_hosts$ **do**
20. $\quad\quad$ **if** $D_{Host_j} = D_{Host_i}$ **then**
21. $\quad\quad\quad g(D_k) = \{host_j, host_i, host_{i+1}, ..., host_n\}$ {Cria Grupos g_dist_bruta}

3.4.4 Algoritmo da Divisão de Dados para o *Reduce*

No novo algoritmo, a divisão de dados é adequada à capacidade computacional das máquinas *Reduces*. A diminuição da granularidade dos dados intermediários cria uma maior quantidade de pequenas tarefas. Assim, a maior quantidade de pequenas tarefas aumenta o paralelismo das execuções e diminui o tempo de processamento do *job*.

A menor granularidade dos dados intermediários tem por objetivo ajustar o tempo de execução das tarefas, aumentar o desempenho do *Reduce* no ambiente heterogêneo e minimizar os efeitos do desbalanceamento de dados intermediários produzidos durante a fase de *Map*.

As chaves intermediárias produzidas nas tarefas *Map* são independentes e tem resultados intermediários $I \subseteq \mathbb{K}\text{x}\mathbb{V}$. Os resultados I contêm todos os pares intermediários (chave/valor) produzidos em uma função. As chaves são formadas por conjunto de dados onde $C(k) = (k, v) \in I$. Os resultados intermediários são divididos em partições P de chaves por uma função de *Hash*. As partições tem um ou mais conjuntos de dados com chaves de mesmo índice, como na equação 3.2 (GUFLER et al., 2011).

$$P(j) = \biguplus_{k \in \mathbb{K}: Hash(k) = j} C(k) \tag{3.2}$$

A abordagem para balancear a carga de processamento na fase de *Reduce* é semelhante a técnica de partição fina proposta por Gufler, porém adaptada ao ambiente heterogêneo. As diferenças estão na forma como os dados são disponibilizados para as máquinas e na metodologia de cálculo para a divisão dos dados. As informações dos dados de cada partição são colocadas em uma fila FIFO global de tarefas. Os dados intermediários são divididos pelo algoritmo 3.4.4 através da soma dos fatores de granularidade Fg_j de cada máquina representado na equação 3.3. Assim, para cada máquina é obtido um valor da divisão entre o tempo de execução de cada tarefa *Reduce* ($t_e Reduce_j$) pela tarefa *Reduce* mais rápida (tr_e), coletados durante a execução da tarefa de medição.

$$Fg_j = \lceil \frac{t_e Reduce_j}{tr_e} \rceil \tag{3.3}$$

Diferente dos algoritmos do modelo original em que a quantidade de *Reduces* só pode ser definida pelo programador, no novo algoritmo da divisão dos dados do *Reduce* a soma dos fatores de granularidade define o número de tarefas *Reduces* (chamada de $Q\_reduces$). Uma função de *Hash* é aplicada à saída dos pares intermediários em cada máquina, chamados de $P_{interm_{Host_j}}$. Os dados são particionados por $Q\_reduces$ e colocados na fila FIFO, linhas 6 a 7 do algoritmo 3.4.4. Quando uma máquina ficar livre, ela recebe a localização dos dados que foram armazenados na fila para serem executados.

Algoritmo 3.4.4 Divisão dos dados *Reduce*

Require: $tr_e = 100$; $t_e Reduce$;
1. **while** $tr_e > t_e Reduce_i$ **do**
2. $\quad tr_e = t_e Reduce_i$ {Procura o menor tr_e}
3. **for** $j \leftarrow 1$ to Num_hosts **do**
4. $\quad Fg_j = \lceil \frac{t_e Reduce_j}{tr_e} \rceil$ {Calcula o fator de correção para cada máquina.}
5. $Q\_reduces = \sum_1^{Num\_hosts} Fg_j$
6. Executa $funcHash(P_{interm_{Host_j}}\{Key, Value\}$ mod $Q\_reduce)$ {Divisão dos dados.}
7. $Fila() \leftarrow (taskReduce, Host_j)$ {As tarefas são colocadas em uma fila com localização dos dados}

A Figura 3.6 ilustra o modelo proposta para a distribuição de dados intermediários do *Map*. A distribuição dos dados intermediários está diretamente relacionada com a criação de tarefas *Reduce*. A quantidade de chaves intermediárias distintas distribuídas por tarefa é menor do que no modelo original. Por exemplo, no modelo original cada tarefa *Reduce* é composta de vários pares diferentes de chaves, que são todas enviadas para o mesma máquina, como nas tarefas 1, 2 e 3 da Figura 3.6. No modelo proposto, os mesmos dados intermediários originam mais tarefas com uma menor quantidade de chaves diferentes por tarefa, como ilustra a Figura 3.6.

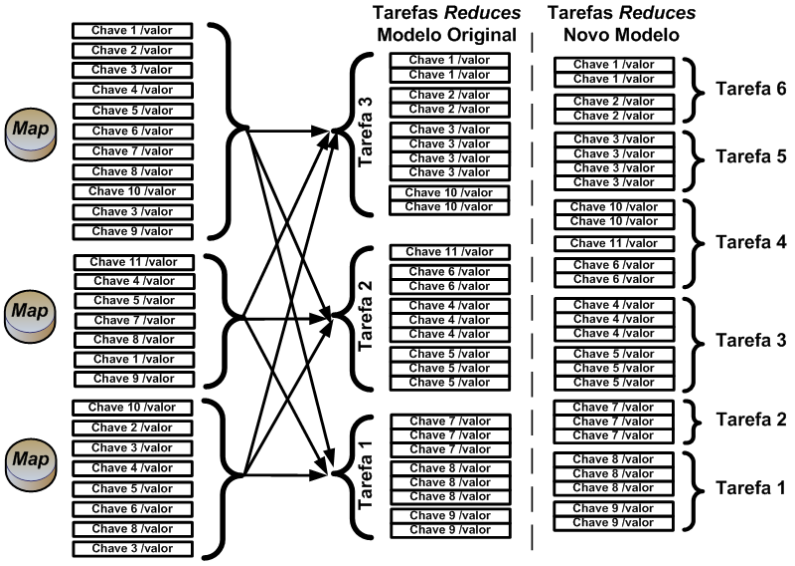


Figura 3.6: Modelo proposto para a distribuição de dados intermediários do *Map*

As informações contém a localização dos dados que compõem as chaves intermediárias e a função *Reduce* que deve ser processada pela máquina. Cada tarefa *Reduce* irá

processar todos os pares intermediários de uma mesma chave. Embora, não se possa garantir uma distribuição totalmente balanceada, na média, as máquinas que tiverem menor capacidade de processamento irão processar menor quantidade de tarefas, enquanto as máquinas mais rápidas processam uma maior quantidade de tarefas.

Por exemplo, sejam 3 máquinas selecionadas num *cluster* para executar uma determinada função *Reduce* sobre 15 chaves intermediárias produzidas na fase de *Map*. Uma máquina executa as tarefas na metade do tempo das outras duas máquinas. A rede e os tempos de transferência de dados são abstraídos por questões de simplificação do exemplo. Cada chave/valor é representada simplesmente por CH e tempo de execução é um valor genérico T. As Figuras 3.7 e 3.8 mostram o fluxo da execução das tarefas *reduce* para o algoritmo original e para o *MR-A++*, respectivamente.

No algoritmo original, os dados das chaves intermediárias são particionadas em 3 tarefas *Reduce*, P1 ={CH1, CH1, CH4, CH7, CH7}, P2= {CH2, CH2, CH5, CH8, CH8} e P3 = {CH3, CH3, CH6, CH6, CH9}, e distribuídos às máquinas 1, 2 e 3 respectivamente. O objetivo é distribuir uma quantidade homogênea de chaves para cada máquina. Assim, na Figura 3.7, para as máquinas mais lentas (Máquinas 2 e 3), o tempo de execução da tarefa *Reduce* é de 2,4 T e o custo de execução de cada chave é 0,48 T. Enquanto, na máquina mais rápida (Máquina 1), o tempo da tarefa *Reduce* é de 1,2 T e o custo de execução de cada chave é de 0,24 T. Portanto, o escalonador de tarefas deverá lançar uma tarefa especulativa para a Máquina 1 de qualquer uma das outras tarefas das máquinas mais lentas, já que ela está livre e não existem tarefas pendentes.

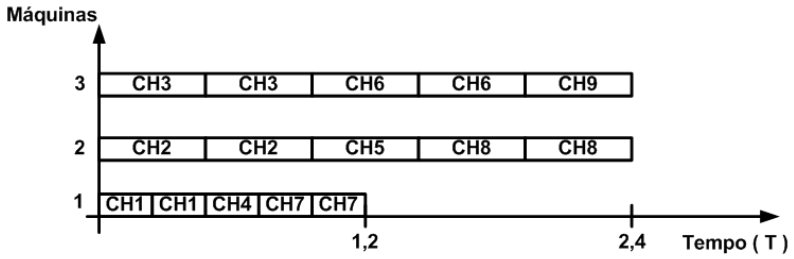


Figura 3.7: Fluxo da execução das tarefas de *Reduce* no algoritmo original

Na distribuição dos dados para as tarefas *Reduce* no *MR-A++*, o cálculo de $Q\_reduces$ resulta em 5 tarefas ao invés de 3 no modelo original. Assim, as chaves intermediárias podem ser particionas por uma função de *Hash* como, por exemplo, em P1 ={CH1, CH1, CH6, CH6}, P2= {CH2, CH2, CH7, CH7}, P3 = {CH3, CH3, CH8, CH8}, P4 = {CH4, CH9} e P5 = {CH5}, e após são colocadas em uma fila FIFO. As máquinas 1, 2 e 3 recebem as tarefas originadas das partições P1, P2 e P3 respetivamente, como mostra a Figura 3.8.

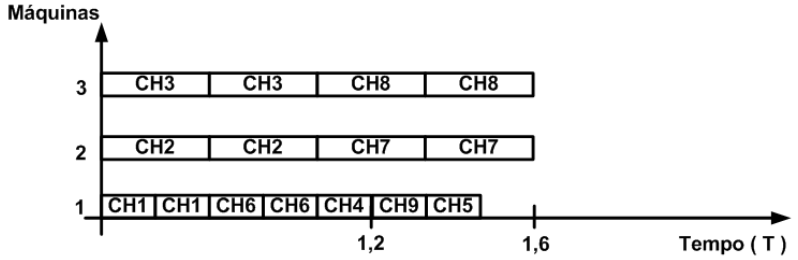


Figura 3.8: Fluxo da execução das tarefas de *Reduce* no *MR-A++*

Após a máquina mais rápida (Máquina 1) concluir suas tarefas, ela solicita ao escalonador mais tarefas para serem executadas e, então, o escalonador encaminha a tarefa 4, da partição 4. Quando for concluída a tarefa 4 a máquina novamente solicita mais tarefas ao escalonador, que envia a tarefa 5, correspondente à partição 5 para ser executada. Assim, o objetivo do algoritmo de distribuição de dados *Reduce* do *MR-A++* é de adequar o tamanho das tarefas ao tempo das execuções das máquinas, através da diminuição da granularidade dos dados intermediários. Quando a Máquina 1, terminar suas tarefas, será lançado uma tarefa especulativa de uma das tarefas das outras máquinas.

Com a estratégia adotada, o tempo total do *Reduce* passou de 2,4 T para 1,6 T, como pode ser observado no exemplo, da Figura 3.8. A execução das tarefas nas máquinas mais lentas e nas máquinas mais rápidas tendem a serem aproximadamente iguais.

Após os dados serem divididos são necessários distribuir as tarefas conforme a capacidade computacional das máquinas e gerenciar o progresso de suas execuções. A seção 3.5 detalha os algoritmos para a distribuição de tarefas no *MR-A++*.

3.5 Detalhamento dos Algoritmos de Distribuição de Tarefas

As tarefas de *MapReduce* são entregues preferencialmente às máquinas que contém os dados locais. O mestre utiliza um mecanismo de *heartbeat* para comunicar-se com as mauinas e fazer o gerenciamento das tarefas.

A solução adotada para a distribuição de tarefas foi particionar as máquinas em grupos conforme apresentado na subseção 3.4.3 e alterar os mecanismos de controle de tarefas remotas e especulativas. O gerenciamento de máquinas lentas (*stragglers*) foi separado em dois controles, um para o *Map* e outro para o *Reduce*.

O uso de controles distintos é justificado porque no *Map* não é conveniente que hajam lançamentos de tarefas remotas e a quantidade de tarefas especulativas deve ser a menor possível. Entretanto no *Reduce*, como é custoso determinar o tamanho dos pares intermediários, a estratégia foi de permitir com que as máquinas mais rápidas roubem tarefas das máquinas mais lentas.

3.5.1 Algoritmo de Distribuição de Tarefas *Map*

O algoritmo 3.4.3, apresentado na subseção 3.4.2, divide as máquinas em grupos de mesma capacidade computacional. Os grupos são utilizados no algoritmo de distribuição de tarefas do *Map* para controlar o lançamento de tarefas especulativas e remotas. As tarefas são colocadas em uma fila de tarefas e, à medida que as máquinas ficam livres, uma mensagem de *heartbeat* é enviada ao mestre para solicitar mais tarefas.

No novo algoritmo 3.5.1 para a distribuição de tarefas *Map*, diferentemente do algoritmo original, o mestre verifica se vale a pena lançar uma tarefa remota para uma máquina que não tem dados locais, considerando o tempo de execução e de transferência dos dados entre as máquinas envolvidas (linhas 7 a 9). Uma máquina é marcada como *straggler* (linhas 11 a 13) considerando a média de execução do grupo *g_dist_bruta*. Esta abordagem muda completamente o comportamento dos lançamentos especulativos adequando o sistema à heterogeneidade do ambiente, como poderá ser constatado nas seções 4.4 e 4.5

Algoritmo 3.5.1 Algoritmo para a distribuição de tarefas *Map*

1. $Receive() = heartbeat[Status\ Host()]$
2. **for** $h \leftarrow 1$ to Num_hosts **do**
3. **for** $t \leftarrow 1$ to Num_chunks **do**
4. **if** existe $Slot\_map$ livre e $D_{Host_h} > 0$ **then**
5. **if** $chunk_t = local$ **then**
6. $New\_taskId_{Host_h} \leftarrow f(map[chunk_t])$ {Tarefa Normal}
7. **else**
8. **if** $taskTime_{rest}(Host_h) + New\_taskTime(Host_h) > \varepsilon_{h \rightarrow r} + New\_taskTime(Host_r)$ **then**
9. Envia localização do $Chunk_t$ para o $Host_r$
10. $New\_taskId(Host_r) \leftarrow f(map[chunk_t])$ {Tarefa Remota}
11. **if** $taskTime > 60s$ e o Progresso $< average\_g(D_k)$ **then**
12. $Host_h \leftarrow straggler$
13. Lança tarefa especulativa para o $Host_{h+1}$ que tenha $chunk_t$ {Tarefa Especulativa}

3.5.2 Algoritmo de Distribuição de Tarefas *Reduce*

O algoritmo 3.5.2 distribui as tarefas *Reduce*. A abordagem aumenta a quantidade de tarefas executadas concorrentemente, porque existem menos máquinas marcadas como *stragglers*, devido ao ajuste feito para a distribuição dos dados.

Na fase de *Reduce* quando as tarefas são distribuídas é realizado uma cópia das chaves intermediárias para serem processadas nas máquinas (linhas 2 a 8). Os dados são distribuídos para as tarefas *Reduces* logo que pelo menos 5% das tarefas *Maps* estejam concluídas, através de um mecanismo de *prefetch*. A abordagem do *MR-A++* a quantidade de dados que executam o *prefetch* é menor que nos algoritmos originais. Porém, o tempo das execuções das tarefas *Reduce* são menores, porque tem menos dados para serem processados por tarefa. O resultado reflete-se no menor tempo do *job*. No algoritmo 3.5.2 (linhas 9 a 13) as máquinas lentas são definidas a partir da média dos tempos de execução de todas as máquinas que efetivamente participam do processamento. Assim as máquinas que tem uma *g_dist_bruta* igual a zero não irão participar da distribuição de tarefas, porque iriam ter tempos de execução muito lentos.

Algoritmo 3.5.2 Algoritmo para a distribuição de tarefas *Reduce*

1. **for** $j \leftarrow 1$ to $Num\_reduces$ **do**
2. $Receive() = heartbeat[StatusHost()]$
3. **if** $Slot\_Reduce > 0 \wedge D_k > 0$ **then**
4. $Local\_Reduce \leftarrow Read(Fila(task))$ {Lê as Informações dos dados}
5. $New\_taskId() \leftarrow f(reduce(map(key[n], value[n]))$
6. Wait $Host[f(Reduce)]$
7. $Send(taskReduce, Host_j)$ { Tarefa Normal}
8. $Send(locais(P_{interm}(key[n], value[n]), Host_j)$ { $Host_j$ Copia pares}
9. **if** $taskTime > 60s$ e Progresso $< Average\_taskReduce$ **then**
10. $Host(j) \leftarrow straggler$
11. $New\_taskId_{esp}() \leftarrow f(Reduce(Map(key[n], value[n]))$ {Cria Tarefa Especulativa}
12. $Send(taskReduce, Host_{j+1})$
13. $Send(locais(P_{interm}(key[n], value[n]), Host_{j+1})$ { $Host_{j+1}$ Copia pares}
14. Wait $Host[f(Reduce)]$ that $Job\_inProgress = complete$

A estratégia de diminuir o tamanho dos dados intermediários aumenta significativamente o desempenho do *MapReduce* em ambiente heterogêneo, como poderá ser constatado nos resultados que serão apresentados no capítulo 4.

4 AVALIAÇÃO DE RESULTADOS

Neste capítulo serão descritas a metodologia utilizada nos testes de validação dos algoritmos, a arquitetura e as configurações do experimentos. Assim como, serão discutidas as justificativas para as decisões tomadas na escolha dos parâmetros para testes.

Através da análise 2^k Fatorial será conhecido o comportamento dos *jobs* e das funções *Map* e *Reduce*, comparando o uso dos novos algoritmos com os algoritmos originais. Os resultados obtidos nos diversos experimentos realizados serão apresentados e analisados.

4.1 Metodologia

Atualmente, as três técnicas conhecidas para a avaliação de desempenho são os modelos analíticos, as simulações e as medições. A escolha apropriada depende de diversos fatores tais como estágio do projeto, tempo necessário, ferramentas, acurácia das informações, dificuldade da avaliação, custo e escalabilidade (JAIN, 1991).

A modelagem analítica e as simulações são consideradas ferramentas essenciais em muitas áreas da ciência para a predição e análise de sistemas complexos. Porém, em muitos problemas de otimização, a relação entre o desempenho e os parâmetros de interesse não é conhecida analiticamente. Os desafios do uso de simulações estão nas parametrizações corretas do ambiente e nas funções do simulador em relação ao sistema real (BHATNAGAR; HEMACHANDRA; MISHRA, 2011).

Experimentos distribuídos em larga escala consomem muito tempo. A criação de ambientes para medição depende de dispositivos de rede e software que tem vários padrões possíveis de implementação, lidando com diferentes ambientes e interações que são muito difíceis de controlar (VELHO; LEGRAND, 2009).

A análise de dados em larga escala foi iniciada por empresas de Internet que operam com milhares de servidores. Construir ambientes heterogêneos reais em larga escala para efetuar medições com milhares de máquinas é uma tarefa extremamente complexa devido à escassez de recursos. Assim, para minimizar estes problemas e comprovar a escalabilidade dos novos algoritmos criados, foi adotado o uso do simulador MRSG (detalhado na seção 4.3), baseado no SimGrid (SIMGRID, 2011).

Um problema chave do uso de modelos de simulação é a depuração de variáveis aleatórias. O uso de simuladores determinísticos permite definir as saídas especificando-se variáveis. Desta forma a comparação entre os resultados das simulações e as características dos simuladores é mais simplificada (JAIN, 1991). Entretanto, na simulação de um ambiente, o uso de variáveis constantes gera resultados iguais na saída. Portanto, o uso de simuladores permite avaliar modelos e determinar uma tendência de comportamento do que é simulado.

As simulações foram executadas em uma máquina com processador $Core\ i7$-2820QM

@ 2,3 GHz de 8 núcleos, com 8 GB de memória RAM e sistema Linux Kernel 2.6.32. Os experimentos realizados testam o desempenho do *MapReduce* com o uso dos novos algoritmos em ambientes heterogêneos.

Nos experimentos utiliza-se desde uma baixa escala de máquinas, com *grids* de 32 máquinas, até uma alta escala com *grids* de até 7.000 máquinas *multi-core*. Ao todo foram realizados 160 experimentos e por ser utilizado um simulador determinístico, cada teste foi repetido 10 vezes, para constatar que realmente não haveriam variações nos resultados das simulações. A quantidade de repetições são devido ao fato do simulador apresentar os mesmos resultados para toda a sequência de testes.

Duas abordagens diferentes de testes foram adotadas. Na primeira abordagem foi analisado o comportamento dos algoritmos em relação ao *job* e às funções de *Map* e *Reduce*. As *grids* eram formadas de 32 a 2.048 máquinas heterogêneas. Dois extremos de bandas de rede foram utilizadas de 10 Mbps e 1 Gpbs. A rede é homogênea para todas as máquinas. Nesta abordagem a quantidade de dados da função *Reduce* é a mesma da entrada de dados na função *Map*.

Outra abordagem analisa a escalabilidade dos algoritmos com alta densidade de máquinas heterogêneas. O comportamento dos *jobs* é avaliado com uma saída de dados intermediários de 50% ou 100% da entrada do *Map*. As *grids* heterogêneas nesta avaliação têm uma quantidade de máquinas que varia de 1.000 a 7.000, com uma rede de 1 Gbps.

O critério para a escolha da banda de 10 Mbps é que essa banda representa um padrão mundial a ser adotado para acesso via Internet (BORGNAT et al., 2009),(BARBOSA, 2011). Assim deseja-se saber se a adequação dos algoritmos seria escalável para *desktop grids* com esta configuração de banda. A banda de 1 Gbps é normalmente utilizada para formar os *clusters* em ambientes reais no *MapReduce*.

Diversos fatores podem alterar o desempenho do *MapReduce*. No simulador assim como no *MapReduce* são possíveis muitas configurações. Portanto, é difícil determinar qual tipo de teste representa um estresse adequado para validar os algoritmos. Muitas vezes um fator é unidirecional, isto é, o desempenho ou continuamente decrementa ou continuamente incrementa. Assim, para resolver este problema utilizou-se a técnica chamada de Análise 2^k Fatorial (JAIN, 1991) para estudar a tendência do comportamento dos *jobs* com os novos algoritmos. Para determinar os fatores que influenciam nos novos algoritmos foram realizados ao todo 144 testes. A avaliação é apresentada na seção 4.4.

4.2 Arquitetura dos Experimentos

Na criação da plataforma, o MRSG permite definir a capacidade computacional e o número de *cores* de cada máquina, a banda em bits e a latência da rede. As configurações de cada plataforma avaliada nos experimentos são apresentadas na Tabela 4.1.

O sistema 1, apresentado na Tabela 4.1.a, será utilizado na análise do comportamento dos algoritmos em relação ao *job* e às funções de *Map* e *Reduce* e o sistema 2, apresentado na Tabela 4.1.c, será utilizado para avaliar o comportamento dos *jobs* com a saída dos dados intermediários do *Map* de 50% ou 100% em relação a sua entrada.

As máquinas são de capacidade heterogênea, com 2 núcleos por máquina. A capacidade computacional em *flops* foi obtida através do *benchmark* MaxxPI[2] (BICAK, 2011). No sistema 1, Tabela 4.1.a a quantidade de máquinas varia de 32 a 2048. Cada máquina foi configurada com dois *slots* para executar tarefas *Map* e dois para executar tarefas *Reduce*. No sistema 2, Tabela 4.1.c, a quantidade de máquinas varia de 1000 a 7000.

As máquinas foram geradas de forma que a capacidade computacional média de processamento em *flops*, como apresentada na Tabela 4.1(a), fosse a mesma para os experimentos poderem ser comparados. Embora, cada máquina tenha uma capacidade computacional diferente, a média computacional de todas é proporcional em cada ambiente.

O número de *chunks* processados foram configurados de forma a representarem uma carga de trabalho média a qual foi definida como sendo 4 tarefas por *slot* para cada máquina. Assim, os dados crescem proporcionalmente à quantidade de máquinas na *grid*, e são apresentados na Tabela 4.1.b para o sistema 1 e na Tabela 4.1.d para o sistema 2.

Tabela 4.1: Configuração do ambiente

(a) Sistema - 1

Máquinas	***Flops/máquina***
32	3,5 E+09
64	3,4 E+09
128	3,5 E+09
256	3,5 E+09
512	3,5 E+09
1024	3,5 E+09
2048	3,5 E+09

(b) Carga de trabalho -1

Máquinas	***Chunks***	**Dados**
32	256	16 GB
64	512	32 GB
128	1.024	64 GB
256	2.048	128 GB
512	4.096	256 GB
1024	8.196	512 GB
2048	16.384	1 TB

(c) Sistema - 2

Máquinas	***Flops/máquina***
1000	3,5 E+09
2000	3,5 E+09
3000	3,5 E+09
4000	3,5 E+09
5000	3,5 E+09
6000	3,5 E+09
7000	3,5 E+09

(d) Carga de trabalho - 2

Máquinas	***Chunks***	**Dados**
1000	8.000	0,5 TB
2000	16.000	1,0 TB
3000	24.000	1,5 TB
4000	32.000	2 TB
5000	40.000	2,5 TB
6000	48.000	3,0 TB
7000	56.000	3,5 TB

O custo computacional de cada tarefa *Map* e *Reduce* é igual a 1000 *flops*/byte. O tamanho da tarefa representa aplicações de tarefas curtas. Tarefas curtas são execuções típicas realizadas pelo *Google* (DEAN; GHEMAWAT, 2010).

4.2.1 Configurações dos Hardwares Simulados

O MRSG tem um *script* próprio para gerar os arquivos de plataforma, originários do Simgrid, onde as configurações são criadas a partir de parâmetros definidos. As capacidades computacionais dos equipamentos são geradas a partir de uma distribuição normal, como a técnica utilizada no trabalho de Javadi (JAVADI et al., 2009). As plataformas foram criadas variando a capacidade computacional de 2E+09 até 5E+09 *flops*. Cada máquina tem uma capacidade diferente, dentro desta variação.

A Tabela 4.2 mostra um exemplo de descrição de equipamentos compatíveis com a capacidade computacional de cada máquina gerada pelos *scripts* de criação de plataforma. Mais exemplos das configurações dos equipamentos são encontradas no anexo A.1 neste trabalho. No exemplo, tem-se 25 equipamentos de diferentes capacidades, o que representa 78,12% da quantidade total das máquinas.

Tabela 4.2: Configuração de estações para experimento com 32 máquinas

Configuração do Equipamento	Quantidade
Intel Xeon 3050 (2130 MHz) – 2GB RAM	1
Intel Core2 Duo E6540 (2330 MHz) – 2GB RAM	1
Intel Core2 Duo E6540 (2330 MHz) – 4GB RAM	1
Intel Pentium Dual Core E5300 (2616MHz) – 2GB RAM	1
Intel Core 2 Duo E6550 (3002MHz) – 3GB RAM	1
AMD Phenom II X2 550 (3100MHz) – 4GB RAM	2
Intel Core 2 Duo E8200 (3200MHz) – 4GB RAM	2
AMD Athlon II X2 255 (3410MHz) – 2GB RAM	1
AMD Phenom II 550 (3567MHz) – 2GB RAM	1
AMD Phenom II 550 (3567MHz) – 4GB RAM	1
Pentium Dual-Core E5400 (2700 MHz) – 4GB RAM	1
Intel Core2 Duo E8135 (2660 MHz) – 4GB RAM	1
Intel Pentium Dual Core E6500 (3520MHz) – 4GB RAM	1
AMD Athlon II X2 250 (3750MHz) – 4 GM RAM	1
Intel Core 2 Duo E6600 (3700MHz) – 2GB RAM	2
Intel Core 2 Duo E6600 (3700MHz) – 4GB RAM	3
AMD Phenom II X3 720 (2809MHz) – 4GB RAM	2
AMD Phenom II X3 720 (3200MHz) – 4GB RAM	1
AMD Phenom II X3 720 (3335MHz) – 4GB RAM	1
AMD Phenom II X3 720 (3600MHz) – 2GB RAM	2
Intel Xeon L5320 (1800MHz) – 4GB RAM	1
Intel Xeon E5335 (2000MHz) – 4GB RAM	2
Intel Xeon X3210 (2130MHz) – 4GB RAM	1
AMD Phenom II X3 720 (3816MHz) – 4GB RAM	1
TOTAL	32

4.3 Simulador MRSG - *MapReduce over SimGrid*

O simulador MRSG foi construído sobre o SimGrid para simular o ambiente de execução do *MapReduce*. O SimGrid foi escolhido pela sua escalabilidade e funcionalidades para simular diferentes tipos de aplicações distribuídas (DONASSOLO et al., 2010).

O objetivo do MRSG é de facilitar a pesquisa do comportamento do *MapReduce* em diferentes plataformas, possibilitando o estudo de novos modelos de algoritmos de modo simplificado e a construção de plataformas em larga escala, sem a necessidade de uma implementação de uma infra-estrutura de *hardware* necessária.

A Figura 4.1, ilustra os módulos necessários para a simulação do *MapReduce*. A plataforma de *hardware* e o comportamento das máquinas são simulados pelo SimGrid. O módulo DFS simula o sistema de arquivos distribuído, onde foram criados os novos algoritmos de distribuição dos dados e o novo mecanismo de agrupamentos.

As informações sobre a rede e as definições de *hardware* são feitas sobre arquivos de plataforma especificados pelo SimGrid. A submissão do *job* é definida por um arquivo de configuração do MRSG onde se especifica a quantidade de tarefas, o custo de cada tarefa, o número de *slots* disponíveis e a entrada de dados. O gerenciamento das execuções do *MapReduce* e o escalonamento de tarefas são feitos no módulos do MRSG. A computação das máquinas e a simulação da *grid* são feitas pelo Simgrid.

Os novos algoritmos foram alterados no módulo *MapReduce*, mais especificamente foram alterados o controle do progresso, os algoritmos do lançamento de tarefas *Map* e *Reduce*, os algoritmos de lançamentos especulativo e remoto, os algoritmos de divisão de dados e o controle dos agrupamentos. No lado cliente no módulo *TaskTracker* foram simulados o envio de informações da tarefa de medição.

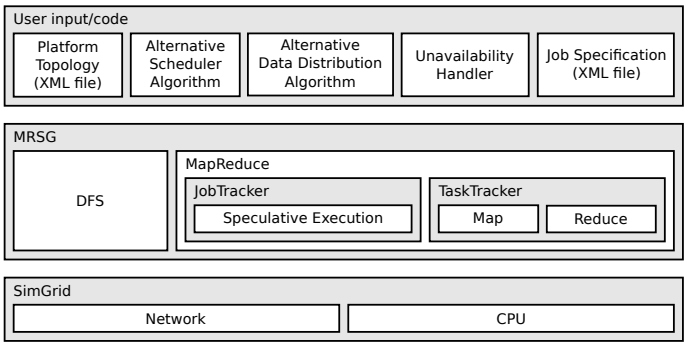


Figura 4.1: Módulos do MRSG

Dois arquivos, escritos em formato XML, definem a estrutura da *grid*. Um arquivo de plataforma determina as características das máquinas e da rede, como é apresentado no *script* 1. Um segundo arquivo de *deployment* descreve a função de cada uma das máquinas na *grid*.

Script 1 Arquivo de plataformas no Simgrid para o MRSG

```
<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
<platform version="3">
  <AS id="AS0" routing="Full">
        <host id="Host 0" power="3500000000.0" core="2" />
        <host id="Host 1" power="3500000000.0" core="2" />
        ......
        <link id="l1" bandwidth="125000000.0" latency="1e-4" />
        <link id="l2" bandwidth="125000000.0" latency="1e-4" />
        .......
<route src="Host 0" dst="Host 1">
                <link_ctn id="l1"/>
        </route>
        ......
        <route src="Host 1" dst="Host 2">
                <link_ctn id="l1"/>
                <link_ctn id="l2"/>
               </route>
         .......
```

Outro aquivo utilizado pelo MRSG é o que define as especificações do *job*, chamado de *mrsg.conf*. As especificações são o total de tarefas *Reduces*, o tamanho do *chunk*, o tamanho da entrada de dados, o número de réplicas, o percentual de saída de dados da função *Map*, o custo de cada tarefa *Map* e *Reduce* e o números de *slots* de *Map* e *Reduce*.

O MRSG usa uma descrição simplificada da entrada de trabalho e dos custos da tarefa. Para a entrada, o usuário deve informar apenas seu tamanho. Portanto, não há dados reais para armazenar e processar na máquina de simulação. Esta abordagem está diretamente ligada ao modo como o SimGrid simula a computação de tarefas. No SimGrid, o poder de computação das máquinas simuladas é medida em *flops* por segundo. Esta descrição simplificada permite aos usuários alterar o tamanho da tarefa, redefinindo os seus custos.

4.3.1 Validação do Simulador

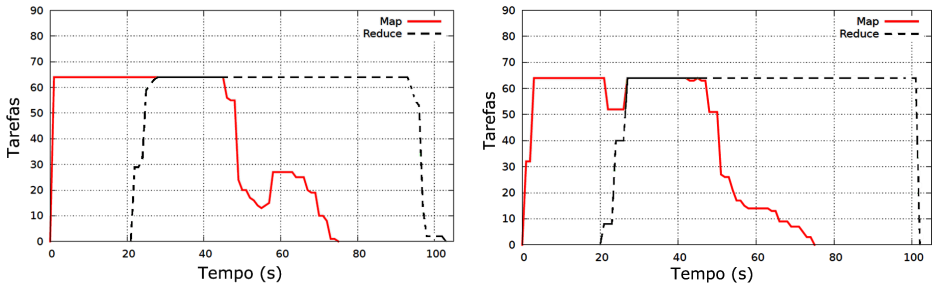
Para validar o simulador MRSG foram executadas simulações reais na *Grid 5000*, repetidas 30 vezes. Para ilustrar a similaridade entre a execução de tarefas no MRSG e

na *Grid 5000* é apresentado na Figura 4.2 um experimento de Filtro de Log. A execução da aplicação filtra os *logs* de traços de disponibilidade de máquinas usando uma entrada de dados de 8,8 GB e 17,6 GB. O *Map* processa cada linha e emite um par (chave, valor) toda vez que há um valor maior que 300 segundos. A chave representa o identificador da máquina e o valor o tempo de disponibilidade. O *Reduce* então calcula a média dos valores ocorridos em cada chave e emite uma nova (chave,valor) com o valor da média de cada máquina. A Tabela 4.3 apresenta a configuração do ambiente real na *Grid 5000* que foi reproduzido no simulador para os testes da aplicação Filtro de Log.

Tabela 4.3: Configuração do ambiente para o Filtro de *Log*

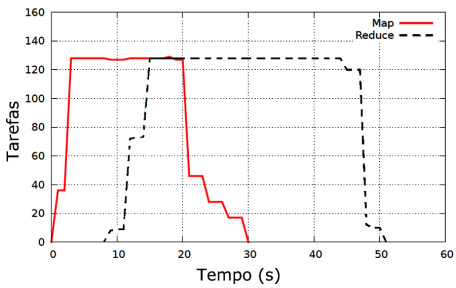
Descrição	**Config. 1**	**Config. 2**
Nós	32	64
Cores	64	512
CPU	3.0 GHz	2.5 GHz
Memória	2 GB	16 GB
Banda	10 Gb/s	20 Gb/s
Entrada de Dados	8,8 GB	17,6 GB
Número de *Maps*	141	282
Número de Reduces	64	128

A Figura 4.2 apresenta o comportamento das execuções de *jobs* em ambiente real Figura 4.2.a e Figura 4.2.c, e simulado Figura 4.2.b e Figura 4.2.d para a aplicação Log Filter, para cada configuração.

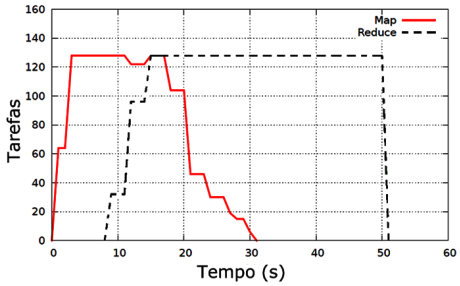


(a) Grid'5000 - Config. 1

(b) MRSG - Config. 1



(c) Grid'5000 - Config. 2



(d) MRSG - Config. 2

Figura 4.2: Comparação da execução entre Grid'5000 e MRSG para aplicação Filtro de *Log*

O comportamento das execuções é semelhante com uma pequena variação a menor (de 1,91% à 4,83%) no tempo total de execução do ambiente real em relação ao simulado. Esta variação é aceitável e menor que a identificada nos simuladores da seção 2.8. O

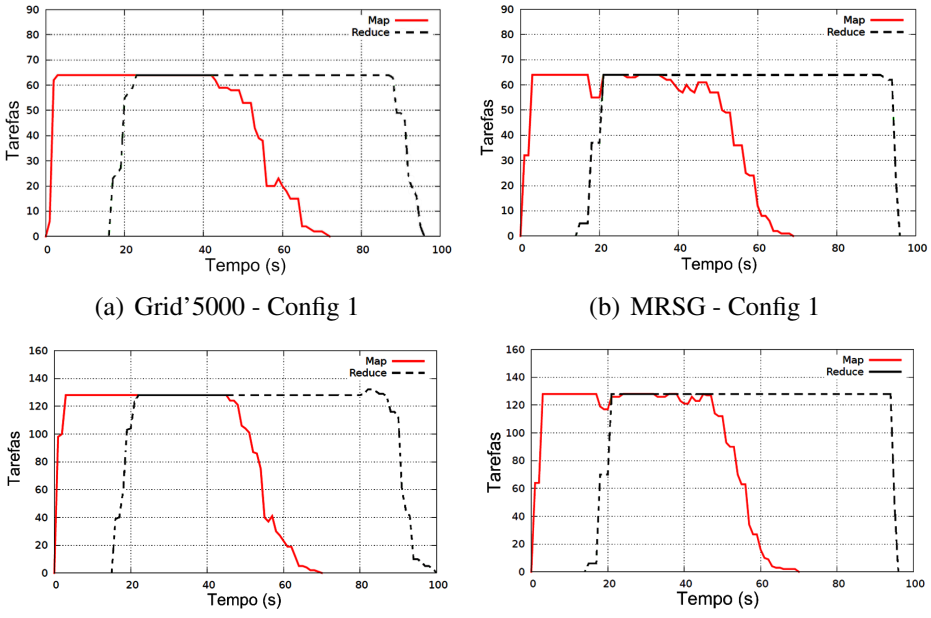
processamento do Filtro de Log representa as características de parte das aplicações executadas pelo *MapReduce*, como no algoritmo *Word Count*. A entrada de dados no *Map* é maior que a saída de dados no *Reduce*. .

Outra aplicação testada foi o *Tera Sort* e a comparação é apresentada na Figura 4.3. A entrada de dados é um arquivo com números para serem ordenados e o tamanho da entrada de dados é igual à saída. A Tabela 4.4 apresenta a configuração do ambiente real na *Grid 5000* que foi reproduzido no simulador para os testes da aplicação *Tera Sort*.

Tabela 4.4: Configuração utilizada na Grid'5000 para a execução do Tera Sort

Descrição	**Config. 1**	**Config. 2**
Nós	32	64
Núcleos	128	128
CPU	2.0 GHz	2.0 GHz
Memória	2 GB	2 GB
Banda	2 Gb/s	10 Gb/s
Entrada	12 GB	24 GB
Número de *Maps*	192	384
Número de *Reduces*	64	128

No *Tera Sort* a função de *Map* recebe um *chunk* de dados e emite uma chave para cada entrada. O valor da chave é um número. As chaves são ordenadas na etapa de partição e, após, o *Reduce* emite os números ordenados, do menor ao maior número.



(a) Grid'5000 - Config 1 (b) MRSG - Config 1

(c) Grid'5000 - Config 2 (d) MRSG - Config 2

Figura 4.3: Comparação entre Grid'5000 e MRSG para aplicação Tera Sort

Observa-se que na Figura 4.2.b e Figura 4.2.c existe uma pequena queda no número de tarefas no ambiente simulado, assim como na Figura 4.3.b e Figura 4.3.d . Esta variação ocorre quando inicia-se a fase de cópia de dados do *Reduce*. Porém esta pequena variação do comportamento não compromete o funcionamento do simulador e não invalida o andamento da simulação.

4.4 Análise 2^k Fatorial

A abordagem da Análise 2^k Fatorial determina, para um dado experimento, o percentual do efeito de k fatores, com dois níveis de incerteza (JAIN, 1991). O uso da técnica objetiva definir os fatores com maior impacto sobre o desempenho dos novos algoritmos. Todos os testes comparam o efeito do uso dos algoritmos através do fator Tipo de Algoritmo e Número de Tarefas *Reduce*. Assim, avalia-se a influência do uso dos algoritmos modificados no *job* e, isoladamente, no *Map* e no *Reduce*. Cada carga de trabalho origina um *job* utilizando os algoritmos originais e modificados. O *job* simula uma tarefa de *word count* qualquer.

A configuração do ambiente dos testes é definida na Tabela 4.5. Três cargas de trabalho diferentes foram aplicadas alterando a quantidade da entrada de dados. A carga de trabalho é determinada conforme a quantidade de máquinas, assim a entrada de dados varia de 8 GB a 1,2 TB, conforme é apresentado na Tabela 4.5.d. Foram tomadas as médias dos valores da influência dos 4 fatores em cada teste.

Tabela 4.5: Fatores da Análise 2^k Fatorial

a. Influência da banda

Fator	**Variação**	**Legenda**
Tipo de algoritmo	Original ou Modificado	A
Número de máquinas (W)	32 ou 512	B
Banda de rede	1Gbps ou 10Mbps	C
Número de tarefas *Reduces*	2 x W ou 6 x W	D

b. Influência do tamanho da saída do *Map* em 1 Gbps

Fator	**Variação**	**Legenda**
Tipo de algoritmo	Original ou Modificado	A
Número de máquinas (W)	32 ou 512	B
Tamanho da saída do *Map*	50% ou 100%	C
Número de tarefas *Reduces*	2 x W ou 6 x W	D

c. Influência do tamanho da saída do *Map* em 10 Mbps

Fator	**Variação**	**Legenda**
Tipo de algoritmo	Original ou Modificado	A
Número de máquinas (W)	32 ou 512	B
Tamanho da saída do *Map*	50% ou 100%	C
Número de tarefas *Reduces*	2 x W ou 6 x W	D

d. Carga de trabalho

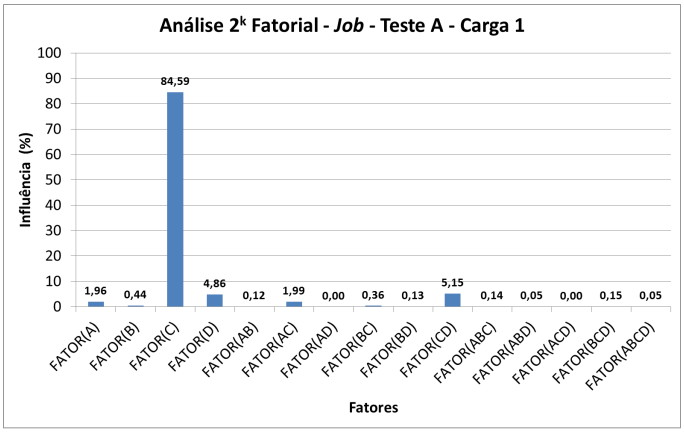
Baixa	4 x W x 64 MB	Teste 1
Média	16 x W x 64 MB	Teste 2
Alta	40 x W x 64 MB	Teste 3

Na Tabela 4.5.a, os fatores A (Tipo de Algoritmo), B (Número de Máquinas) , C (Banda de Rede) e D (Número *Reduces*) determinam a influência da banda sobre a execução de todo o *Job* e sobre a execução do *Map* e do *Reduce*. Nas Tabelas 4.5.b e 4.5.c, o fator C foi trocado para o tamanho da saída do *Map* com objetivo de determinar a influência da menor densidade de chaves na entrada dos *Reduces* em função da Banda de 10Mbps e 1 Gbps.

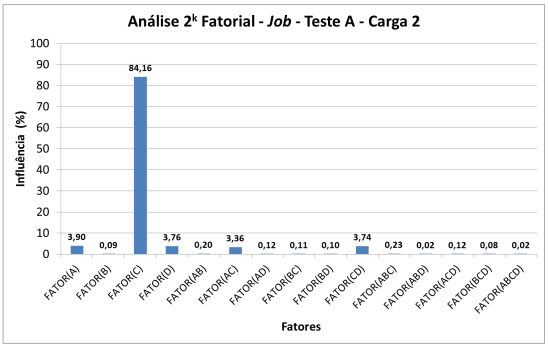
4.4.1 Avaliação da Influência da Banda

Os testes das Figuras 4.4, 4.5 e 4.6 avaliam a influência da banda sobre os fatores A, B, C e D conforme a Tabela 4.5.a. A Figura 4.4 apresenta os testes realizados para avaliar a influência da banda para o *job*.

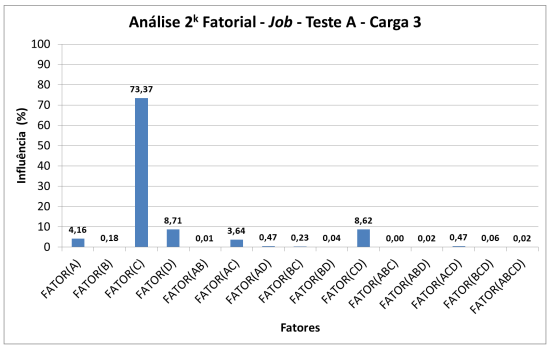
Na (Figura 4.4.a), com baixa carga de trabalho, as variações da banda da rede (fator C) representam 84,59% de toda a variação. A justificativa para este impacto é o tempo para a cópia de dados intermediários da fase *Map* para a fase *Reduce*. O tempo de execução do *Reduce* devido à cópia de dados intermediários, aumenta em média 10 vezes com a diminuição da *banda* e, nesse caso, o tempo de execução representa de 89,42% a 93,32% do tempo total do *job*. Portanto, a influência do tempo da execução das tarefas *Reduce* sobre o tempo do *job* será proporcionalmente maior.



(a) Carga de trabalho baixa



(b) Carga de trabalho média



(c) Carga de trabalho alta

Figura 4.4: Influência da banda sobre os *jobs*

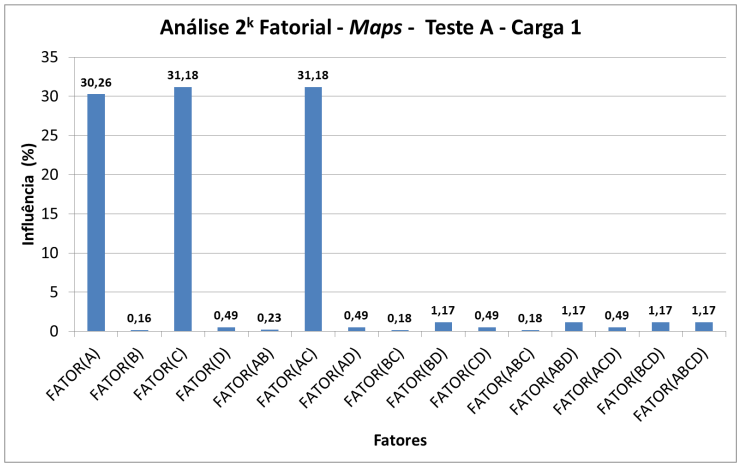
O uso dos novos algoritmos (fator A) representa um impacto de 1,96% e a combinação deste fator com a variação da banda (fator AC) da rede representa 1,99%. O número de tarefas *Reduces* (fator D) representa um impacto de 4,86% e a combinação deste fator com a variação da banda (fator CD) representa 5,15%. Estes fatores somados com a influência fator C (84,59%), são responsáveis por 98,55% das variações dos eventos.

A medida que a carga aumenta, nas Figuras 4.4.b e 4.4.c, a influência da banda de rede (Fator C) reduz, respectivamente, de 84,16% para 73,37%. O uso dos algoritmos (fator A), comprado nos três casos, aumenta a influência de 1,96%, com baixa carga de trabalho, para 4,16% com alta carga de trabalho.

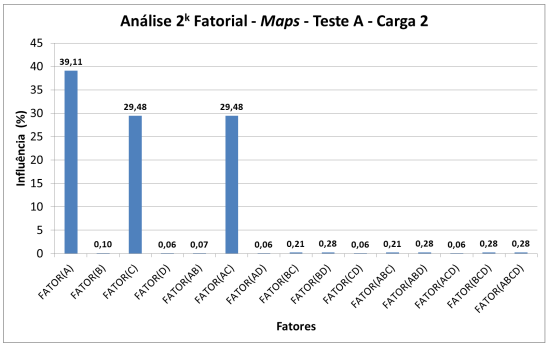
O número de tarefas *Reduce* (fator D), quase dobra de 4,86% para 8,71%. O fator CD, resultado da combinação do número de tarefas *Reduce* com a banda de rede, aumenta

de 5,15% para 8,62%. Portanto, todos os resultados obtidos indicam que a abordagem do uso de uma menor granularidade para os dados intermediários, tem maior impacto no comportamento do *job* a medida que a carga de trabalho aumenta. Também observa-se o comportamento do *job* não depende do número de máquinas (fator B).

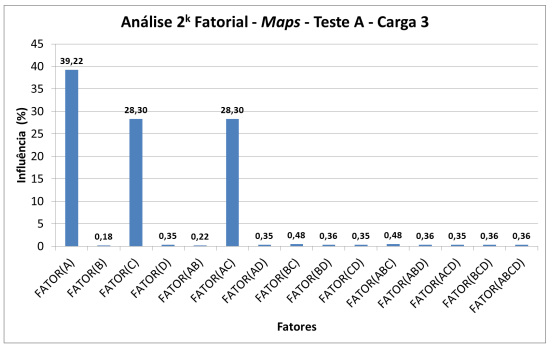
A Figura 4.5 apresenta os testes realizados para o *Map*. Na Figura 4.5.a, com baixa carga de trabalho, 30,26 % do comportamento do *Map* é influenciado pelo uso dos novos algoritmos (fator A). A mudança de banda (fator C) explica 31,18% do comportamento e a combinação do tipo de algoritmo e banda de rede (fator AC) representam 31,18% da influência sobre o comportamento das execuções das tarefas de *Map*. A soma da influência desses fatores representa 92,62% do comportamento das variações do *Map*.



(a) Carga de trabalho baixa



(b) Carga de trabalho média



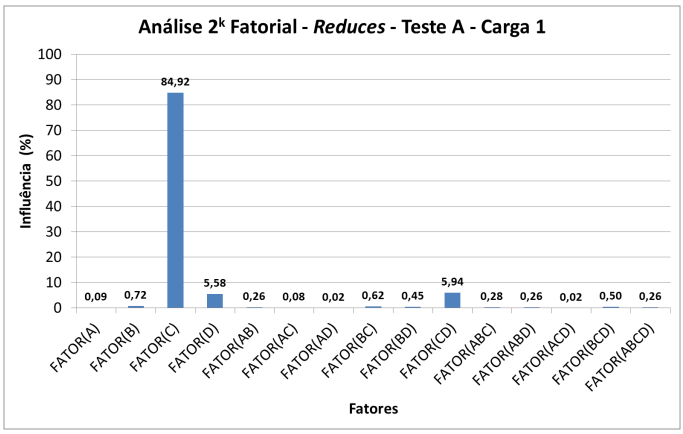
(c) Carga de trabalho alta

Figura 4.5: Influência da banda sobre o *Map*

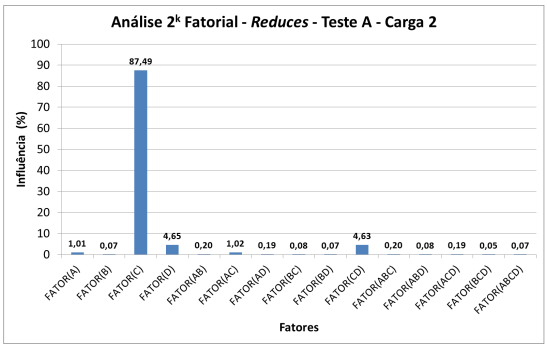
Nas Figuras 4.5.b e 4.5.c, a medida que a carga de trabalho aumenta, a influência do uso dos novos algoritmos estável com valores de 39,11% e 39,22%. De outra forma, a influência da banda (fator C) diminui de 31,18% com baixa carga de trabalho para 28,30% com alta carga de trabalho. Os fatores combinados do uso dos algoritmos e a influência da banda (fator AC) também diminuem de 31,18% com baixa carga de trabalho para 28,30% com alta carga de trabalho.

Portanto, os resultados indicam que o crescimento da carga de trabalho modifica o comportamento do *Map*. A media que o carga cresce, aumenta a influência do fator A e diminui a influência da troca da banda (fator C) e do efeito da combinação do uso dos algoritmos com a troca da banda (fator AC). A influência A quantidade de máquinas (fator B) e o número de tarefas *Reduce* tem influência irrelevante no comportamento do *Map*.

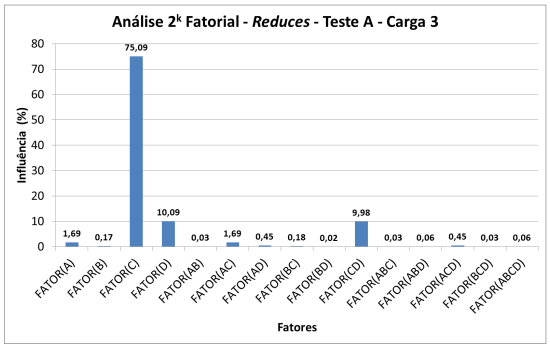
A Figura 4.6 apresenta os testes realizados para avaliar a influência da banda sobre as execuções das tarefas de *Reduce*. A Figura 4.6.a demonstra que o comportamento do *Reduce* com uma baixa carga de trabalho. O comportamento é influenciado pela mudança de banda (fator C) em 84,92%, pelo número de tarefas *Reduce* (fator D) em 5,58% e pela combinação dos dois fatores em conjunto (fator CD) em 5,94%. Os fatores representam 96,44% do comportamento do *Reduce*. Observa-se que o comportamento também não se altera com a varição da quantidade de máquinas e o uso dos algoritmos do *Map*.



(a) Carga de trabalho baixa



(b) Carga de trabalho média



(c) Carga de trabalho ata

Figura 4.6: Influência da banda sobre o *Reduce*

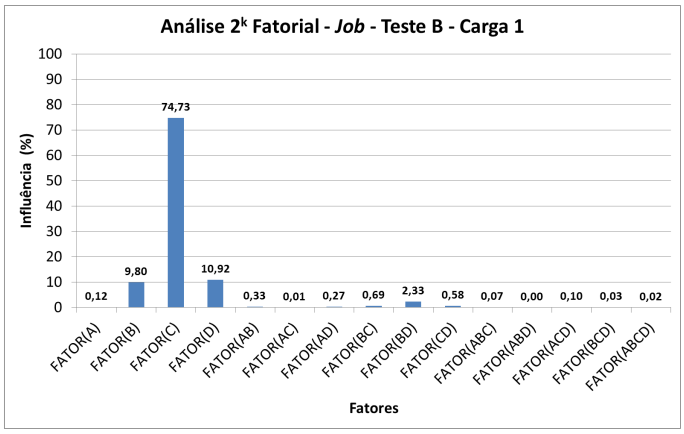
A influência do número de *Reduces* aumenta de 5,58% para 10,09% com o aumento da carga de trabalho. Os fatores combinados da mudança de banda com o número de tarefas *Reduce* (fator AC) também aumentam a influência sobre o comportamento de 5,94%, com baixa carga de trabalho (Figura 4.6.a), para 9,98% com alta carga de trabalho (Figura 4.6.c). Por outro lado, a influência da banda sobre o comportamento do *Reduce* diminui de 84,92% para 75,09%, Figuras 4.6.a e 4.6.c respectivamente.

O resultado obtido para o comportamento relaciona-se com o tempo da cópia dos dados intermediários do *Map* para a execução das tarefas *Reduce*. Assim, o tempo de cópia dos dados intermediários influenciado pela mudança de banda representam um fator importante no comportamento das execuções das tarefas *Reduce*. A abordagem de diminuir a *granularidade* tem maior influência no comportamento do *Reduce* a medida que a carga de trabalho aumenta.

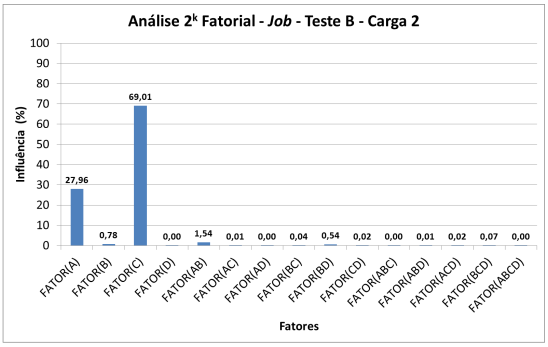
4.4.2 Avaliação da Influência do Tamanho da Saída do *Map* em 1 Gbps

Nas Figuras 4.7, 4.8 e 4.9 avalia-se o comportamento do *Job*, do *Map* e do *Reduce* para uma rede de 1 Gbps, considerando a densidade de dados da saída de dados intermediárias do *Map* como 50% ou 100% da quantidade de dados da entrada. Os testes correspondem à Tabela 4.5.b, apresentada anteriormente, e avaliam a influência do tamanho da saída do *Map* sobre os fatores A (Tipo de Algoritmo), B (Número de Máquinas) , C (Tamanho da Saída do *Map*) e D (Número de Tarefas *Reduces*).

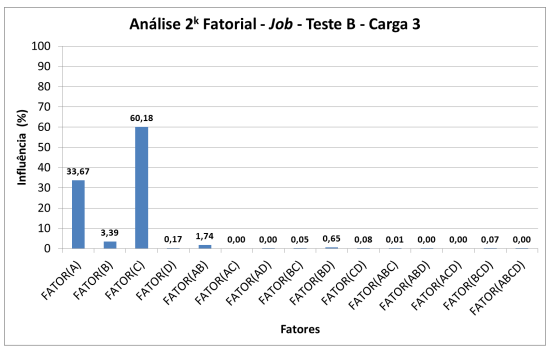
A Figura 4.7 apresenta os testes realizados para o *job*. Na Figura 4.7.a o teste mostra os resultados para uma carga de trabalho baixa. O tamanho da saída do *Map* (fator C) influencia o comportamento em 74,73%. O número de tarefas *Reduces* (fator D) representa um impacto de 10,92% e a combinação deste fator com a variação da banda (fator BD) representa uma influência de 2,33%. Estes fatores somados são responsáveis por 97,78% das variações dos eventos. O resultado é previsível, pois mostra que uma alteração na entrada do *Reduce* influencia o comportamento das execuções.



(a) Carga de trabalho baixa



(b) Carga de trabalho média



(c) Carga de trabalho alta

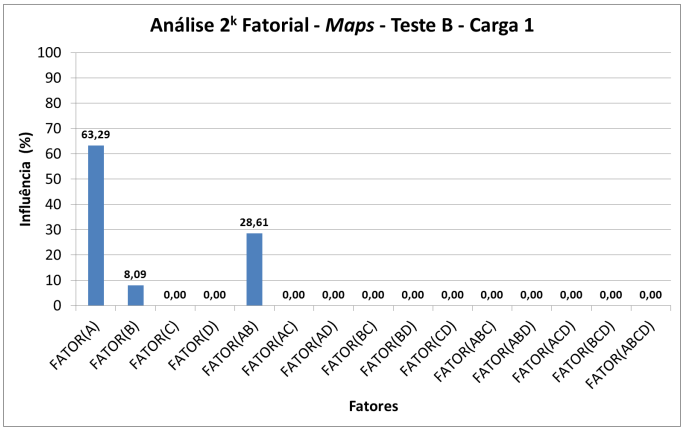
Figura 4.7: Influência do tamanho da saída do *Map* sobre o *job* @ 1 Gbps

Os novos algoritmos influenciam o comportamento do *job* com o aumento da carga de trabalho, como pode ser observado na Figura 4.7. Com baixa carga de trabalho a influência é de apenas 0,12%, na Figura 4.7.a, o valor aumenta para 27,96% com carga de trabalho média, Figura 4.7.b, e para 33,67% com alta carga de trabalho, Figura 4.7.c.

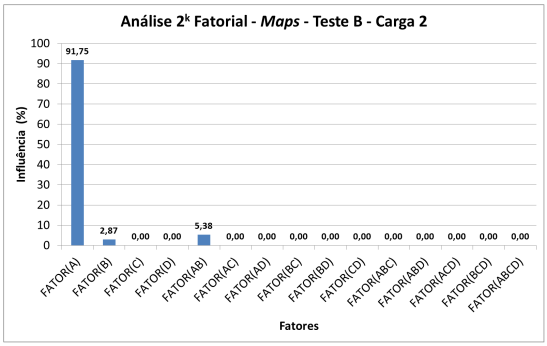
A influência do tamanho da saída do *Map* (fator C) é de 69,01% com carga de trabalho média, Figura 4.7.b, e 60,18% com alta carga de trabalho, Figura 4.7.c. O número de máquinas (fator B) tem uma influência desprezível no comportamento do *job* com cargas de trabalho médias Figura 4.7.b.

Ao contrário, com baixa carga de trabalho, Figura 4.7.a, a influência sobre o comportamento do *job* é de 9,80% e, com altas cargas de trabalho, Figura 4.7.c, representa 3,39%. Os testes indicam que o comportamento do *job* é sensível à carga de trabalho quando existem variações no tamanho dos dados da saída do *Map*. Portanto, pode-se concluir que o uso dos algoritmos modificados do *MR-A++* (fator A) tem maior influência no comportamento do *job* com o aumento da carga de trabalho e com a variação do tamanho da saída do *Map* (fator C). Também pode-se supor que o número de máquinas (fator B) influencia mais o comportamento do *job* em baixas cargas de trabalho do que em altas cargas de trabalho.

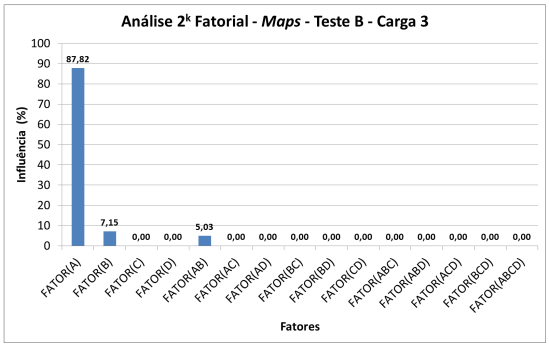
A Figura 4.8 apresenta o comportamento das execuções das tarefas de *Map* em relação à variação do tamanho de sua saída de dados. Na Figura 4.8.a é observado que o uso dos novos algoritmos (fator A) influencia o comportamento do *Map* em 63,29 % . O número de máquinas (fator B) influencia 8,09% o comportamento do *Map* e a combinação destes dois fatores (fator AB) representam 29,61% da influência sobre o comportamento da execução de tarefas de *Map*. A soma da influência de todos os fatores apresentados representam 99,99% do comportamento da execução das tarefas do *Map*. O tamanho da saída do *Map* (fator C) e o número de tarefas *Reduces* (fator D) não tem qualquer influência no *Map*.



(a) Carga de trabalho baixa



(b) Carga de trabalho média



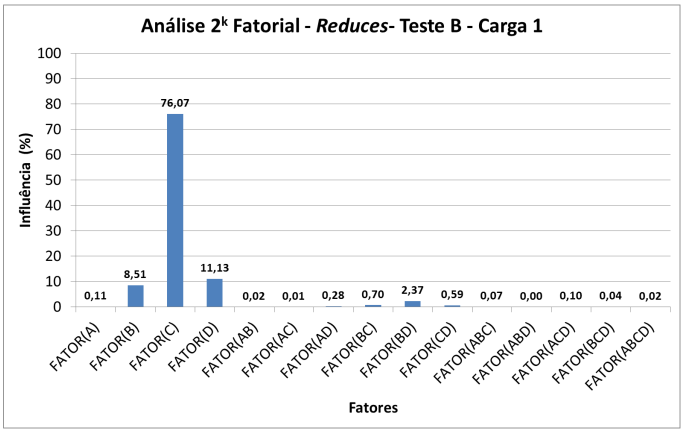
(c) Carga de trabalho alta

Figura 4.8: Influência do tamanho da saída do *Map* sobre o *Map* @ 1 Gbps

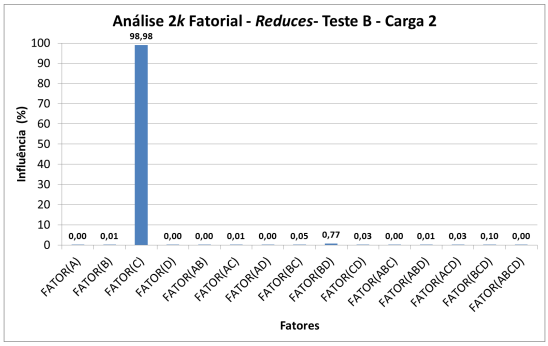
Nas Figuras 4.8.b e 4.8.c observa-se que, o aumento da carga de trabalho, faz o fator AB (combinação das influencias do uso dos algoritmos com o número de máquinas) diminuir seu percentual de influência sobre o comportamento da execução de tarefas de *Map* de 28,61% com abaixa carga de trabalho para 5,03% com alta carga de trabalho.

Os testes da Figura 4.9 apresentam a influência do tamanho da saída do *Map* sobre o *Reduce*. Na Figura 4.9.a, o comportamento do *Reduce* é influenciado pela variação da saída do *Map* (fator C) 76,07% e pelo número de tarefas *Reduces* (fator D) 11,13 %.

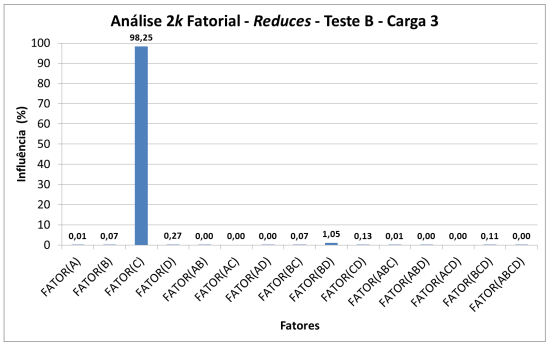
O número de máquinas (fator B) influencia 8,51% e a combinação deste fator com o número de tarefas *Reduces* (fator BD) 2,37%. Os fatores representam 98,78% do comportamento do *Reduce*. Pode-se observar que a dependência da saída (fator C) é maior quando aumenta-se carga de trabalho, isto indica em redes de alta banda disponível o comportamento do *Reduce* é muito depende do tamanho da saída do *Map*.



(a) Carga de trabalho baixa



(b) Carga de trabalho média



(c) Carga de trabalho alta

Figura 4.9: Influência do tamanho da saída do *Map* sobre o *Reduce* @ 1 Gbps

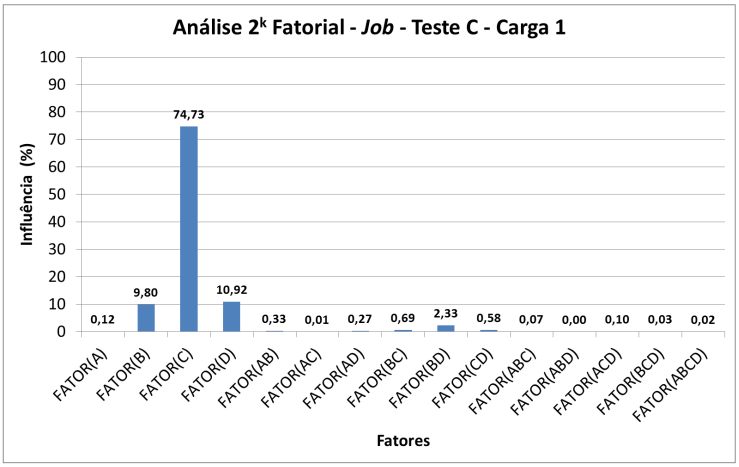
4.4.3 Avaliação da Influência do Tamanho da Saída do *Map* em 10 Mbps

O teste c da Tabela 4.5.c avalia a influência do tamanho da saída do *Map* sobre os fatores A, B, C e D, considerando uma banda lenta de 10 Mbps. A Figura 4.10 e a Tabela 4.6 apresentam os resultados dos testes realizados para avaliar o comportamento do *job*, do *Map* e do *Reduce*.

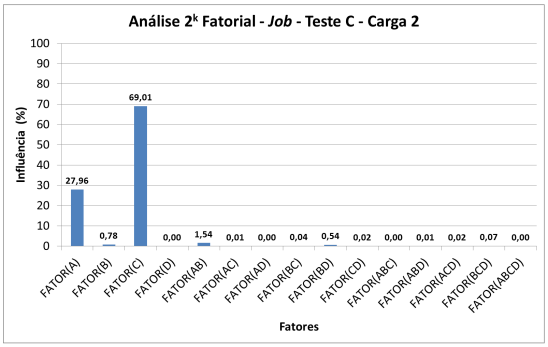
A Figura 4.10.a apresenta o comportamento do *job* para uma carga de trabalho baixa. O comportamento é influenciado pela variação da saída do *Map* (fator C) em 74,73%. A quantidade de máquinas (fator B) influencia em 9,80% o comportamento do *job* e o número de tarefas *Reduces* (fator D) representa um impacto de 10,92% e a combinação deste fator com a variação da banda (fator BD) representa 2,33%. Os fatores somados são responsáveis por 97,78% das variações dos eventos. Este resultado é previsível, pois mostra que uma alteração na entrada do *Reduce* influencia o comportamento das execuções. O comportamento se justifica em função da redução da saída dos dados intermediários das

tarefas *Maps* implicar em menores tempos para transferências dos dados para as máquinas que executarão as tarefas de *Reduce*.

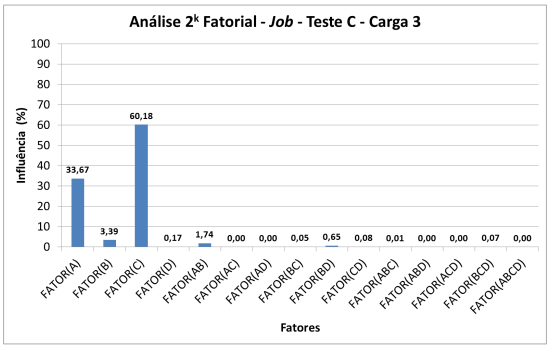
Porém, o comportamento do *job* se altera conforme a carga de trabalho em redes de 10 Mbps, como pode ser observado na Figura 4.10.b e Figura 4.10.c. Com carga de trabalho média as variações são influenciadas 69,01% pela variação da saída do *Map* (fator C) e, com alta carga de trabalho, passa para 60,18%.



(a) Carga de trabalho baixa



(b) Carga de trabalho média



(c) Carga de trabalho alta

Figura 4.10: Influência do tamanho da saída do *Map* sobre o *job* @ 10 Mbps

Entretanto, o uso dos algoritmos (fator A) que com baixa carga de trabalho tinha uma influência de 0,12% passa para 27,96% com carga de trabalho média e para 33,67% com alta carga de trabalho. O número de máquinas (fator B) deixa de influenciar significativamente o comportamento do *job* como pode ser observado na Figura 4.10. O comportamento do *job* da Figura 4.10, em redes de baixa banda disponível, é igual ao comportamento já apresentado na Figura 4.7, com redes de alta capacidade de banda.

A Tabela 4.6 apresenta o resultado dos testes realizados para avaliar a influência do tamanho da saída do *Map* em redes de 10 Mbps, no comportamento das execuções das tarefas de *Map* e de *Reduce*. Ao comparar os dados verifica-se que os resultado são iguais ao comportamento em redes de 1 Gbps, já avaliado nas Figuras 4.8 e 4.9. Assim, demonstra-se que o comportamento do *Map* e do *Reduce* independem da influência da banda de rede quando altera-se o tamanho da saída do *Map*.

Tabela 4.6: Influência do tamanho da saída do *Map* @ 10 Mbps

	% Influência *Map*			% Influência *Reduce*		
Fator \ Carga	Baixa	Média	Alta	Baixa	Média	Alta
A	63,29	27,96	33,67	63,29	91,75	87,82
B	8,09	0,78	3,39	8,09	2,87	7,15
C	0	69,01	60,18	0	0	0
D	0	0	0,17	0	0	6
AB	28,61	1,54	1,74	28,61	5,38	0
AC	0	0,01	0	0	0	5,03
AD	0	0	0	0	0	0
BC	0	0,04	0,05	0	0	0
BD	0	0,54	0,65	0	0	0
CD	0	0,02	0,08	0	0	0
ABC	0	0	0,01	0	0	0
ABD	0	0,01	0	0	0	0
ACD	0	0,02	0	0	0	0
BCD	0	0,07	0,07	0	0	0
ABCD	0	0	0	0	0	0

4.4.4 Conclusões da Análise 2^k Fatorial

Os testes demonstram que a influência da banda sobre o *job* se altera à medida que a carga de trabalho aumenta. O fator banda de rede, tem um peso maior sobre o comportamento *job* em função do tempo gasto para serem executadas as cópias de dados intermediários da fase de *Map* para a fase de *Reduce*.

O comportamento do *Map* é influenciado pelo uso dos novos algoritmos, pela troca de banda e pela combinação destes dois fatores na proporção média de 1/3, com baixas cargas de trabalho. À medida que a carga de trabalho aumenta, o uso dos novos algoritmos tendem a influenciar mais no comportamento do *Map* em 39,22%. De outra forma, as influências da banda e de fatores combinados, como uso dos novos algoritmos e a troca da banda, diminuem com o aumento da carga de trabalho.

O comportamento do *Reduce* é influenciado principalmente pela troca de banda. A alta carga de trabalho aumenta a influência do número de tarefas *Reduces* no comportamento do *job* em 10,09%. Portanto, o comportamento é afetado com a mudança da granularidade dos dados.

A avaliação da influência do tamanho da saída do *Map*, demonstra que o comportamento do *Map* independe da banda de rede, mas é sensível à carga de trabalho. O fator combinado do número de máquinas com o uso dos novos algoritmos, diminui de 29,61%, com baixa carga de trabalho, para 5,03% com altas cargas de trabalho. Assim, a redução 83,01% demonstra-se quanto o *Map* é sensível à variação da carga de trabalho .

O comportamento do *Reduce* em *links* de 1 Gbps é influenciado basicamente pelo tamanho da saída do *Map*, principalmente com altas cargas de trabalho. Porém, com baixas cargas de trabalho o número de máquinas influencia o comportamento em 8,51% e o número de *Reduces* em 11,13%, ao contrário do que ocorre com altas cargas de trabalho.

O uso da técnica da Análise 2^k Fatorial permitiu avaliar os efeitos dos novos algoritmos sobre o *MapReduce*. Na próxima seção 4.5 serão analisados os resultados dos novos algoritmos quanto ao desempenho do *MapReduce* e à alta escala das máquinas.

4.5 Testes e Comparativos

Os testes apresentados nesta seção foram todos simulados em ambientes heterogêneos. A Figura 4.11 apresenta o tempo das tarefas de *Map* e *Reduce*, considerando o uso de uma rede de 1 Gbps. Os *jobs* demonstram tempos obtidos com os algoritmos originais e com o uso dos novos algoritmos.

O tempo das execuções dos *jobs* com os novos algoritmos é 22,00% mais rápido, no pior caso, e 35,76% mais rápido, no melhor caso, se comparado com o uso dos algoritmos originais. O resultado não é maior porque o desempenho do algoritmo original é influenciado pela maior disponibilidade de banda, que esconde o impacto da transferência de dados entre as máquinas em tempo de execução. Assim, as tarefas distribuídas pelo algoritmo original podem ser compensadas com o lançamento de tarefas remotas e especulativas.

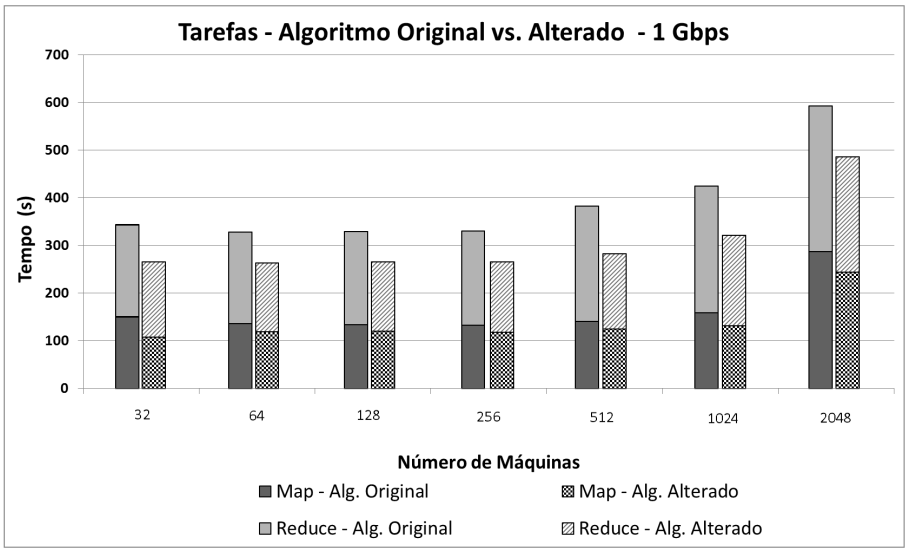


Figura 4.11: Tarefas - Algoritmo original vs. alterado @ 1 Gbps

Na função *Map* o tempo da tarefa, com o uso dos novos algoritmos, fica mais rápida 10,27%, no pior caso, e 27,96%, no melhor caso. O tempo das tarefas *Reduce* representam de 51,47% a 63,10% do tempo gasto pelo *job*. Neste caso, uma redução nos tempos do *Reduce* tem maior impacto sobre o *job*. O tempo das tarefas no *Reduce* fica mais rápido de 17,76%, no pior caso, a 34,89% no melhor caso.

A Tabela 4.7 compara o número de lançamento de tarefas remotas e especulativas, entre o algoritmo original e os novos algoritmos para uma rede de 1 Gbps. O escalonador, no algoritmo original, lança diversas tarefas remotas para melhorar o desempenho do *Map*, gerando cópias de dados durante o tempo de execução.

O lançamento das tarefas remotas melhora o desempenho dos *jobs*, mas tem o efeito de causar uma pequena lentidão no sistema. A lentidão é escondida pela alta disponibilidade de banda entre as máquinas, típica de grandes *data centers*. Porém, este modelo gera maiores tempos de execução em redes lentas e as cópias acabam influenciado negativamente no desempenho.

O novo modelo para o lançamento de tarefas especulativas e remotas, apresentado no capítulo 3, é evidenciado na Tabela 4.7. O escalonador antes de lançar uma tarefa remota, verifica primeiro se vale a pena lançá-la em outra máquina disponível ou então,

Tabela 4.7: Lançamentos especulativos e remotos @ 1 Gbps

Máquinas	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Tarefas remotas *Map* *	24	60	91	175	449	1168	2499
Tarefas remotas *Map* **	0	0	0	0	0	0	0
Tarefas especulativas *Reduce* *	11	9	24	47	136	439	1017
Tarefas especulativas *Reduce* **	0	3	7	11	7	52	0

* Algoritmo original, ** Novos algoritmos

se é melhor aguardar a conclusão da execução de uma tarefa corrente. Assim, um menor lançamento de tarefas remotas ocorre naturalmente.

O menor lançamento de tarefas especulativas deve-se ao novo modelo de agrupamentos, que permite a classificação de uma máquina lenta conforme a média de execuções das máquinas de uma mesma *g_dist_bruta*, e a uma maior quantidade de tarefas curtas na fase de *Reduce*. Portanto, o número de falsos positivos diminui e o resultado é um melhor desempenho do sistema em ambientes heterogêneos.

A Figura 4.12, apresenta o comportamento do *job* para uma rede de 10 Mbps. A execução do modelo *MapReduce* com máquinas heterogêneas em redes lentas apresenta um baixo desempenho porque, em aplicações intensivas, como o *MapReduce*, há uma grande quantidade de dados para serem copiados entre as máquinas durante a fase de *Reduce*.

Pode-se verificar que, com os novos algoritmos, o tempo total do *job* ficou 52,04% mais rápido, para o pior caso, e 72,08%, no melhor caso, que equivale a um ganho de 2,1 a 3,5 vezes. O impacto do tempo da execução das tarefas de *Map* é menor se comparado com o das tarefas *Reduce*. A proporcionalidade do tempo da execução da função *Reduce* varia de 73,98% a 95,76% em relação ao tempo total do *job*.

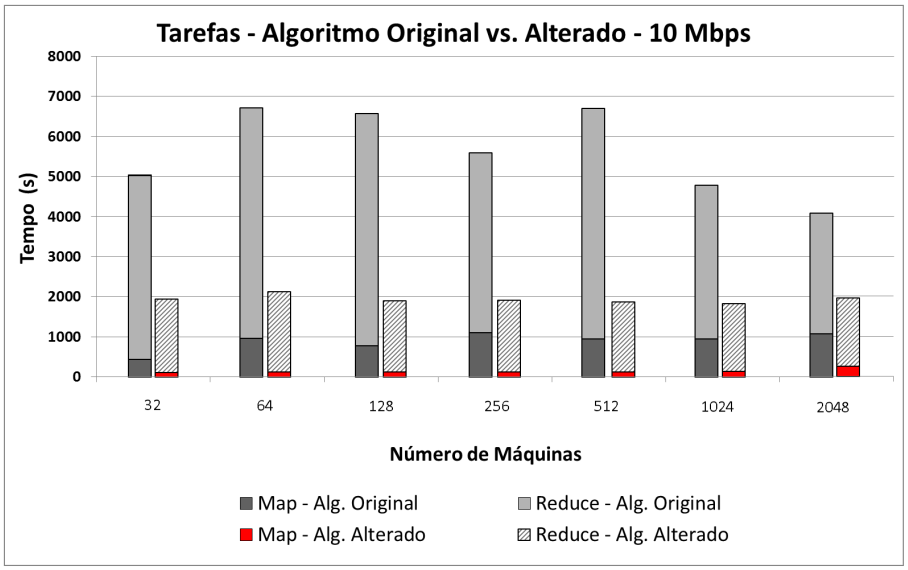


Figura 4.12: Tarefas - Algoritmo original vs. alterado @ 10 Mbps

As tarefas de *Map* apresentam uma melhora no tempo de 75,06% a 89,27% com o uso dos novos algoritmos, isto representa um ganho de 4 a 9,3 vezes respectivamente. Para as tarefas *Reduce* a melhora dos tempos de execução foram de 43,27%, para o pior caso,

e 69,73% para o melhor caso, correspondendo a um ganho de 1,8 e 3,3 vezes respectivamente. Os ganhos obtidos com o *Reduce*, embora menores, impactam mais sobre os ganhos de desempenho dos novos algoritmos.

A Tabela 4.8 apresenta o número de lançamentos remotos e especulativos obtidos nos testes em uma rede de 10 Mbps. Observa-se um grande número de lançamentos remotos, semelhante ao que acontece nos testes da Tabela 4.7 com uma rede de 1 Gbps. Portanto, pode-se concluir que o resultado está relacionado ao modelo do *MapReduce* e não a menor disponibilidade de banda da rede. Assim, as mudanças nos algoritmos do *MR-A++* alteram o comportamento do *MapReduce* e equacionam o problema de desempenho em ambientes heterogêneos com baixa disponibilidade de banda.

Tabela 4.8: Lançamentos especulativos e remotos @ 10 Mbps

Máquinas	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**
Tarefas remotas *Map* *	24	59	91	175	444	1143	2438
Tarefas remotas *Map* **	0	0	0	0	0	0	0
Tarefas especulativas *Reduce* *	11	9	24	47	136	439	1017
Tarefas especulativas *Reduce* **	0	0	0	0	0	0	0

* Algoritmo original, ** Novos algoritmos

No comportamento do *MapReduce* com algoritmos originais, em ambientes homogêneos, espera-se ter um número menor de lançamentos remotos e especulativos. O motivo está relacionado com o mesmo poder computacional das máquinas, com os dados balanceados e com os tempos iguais de execução de tarefas. Assim, os resultados obtidos na Tabela 4.8 para os novos algoritmos em ambiente heterogêneo, mostram que o comportamento dos lançamentos remotos e especulativos é adequado.

Na Figura 4.13 é apresentado a influência da granularidade dos dados da saída do *Map* para os tempos de execução do *Reduce* em uma rede de 1 Gbps. O melhor tempo de execução é 34,89% mais rápido, obtido com um grão de 1/4 da saída do *Map*, o tamanho equivale a 16 MB de dados. No pior caso a melhora do tempo de execução é de 19,32%, com um grão de 1/2 da saída de dados ou seja 32 MB de dados.

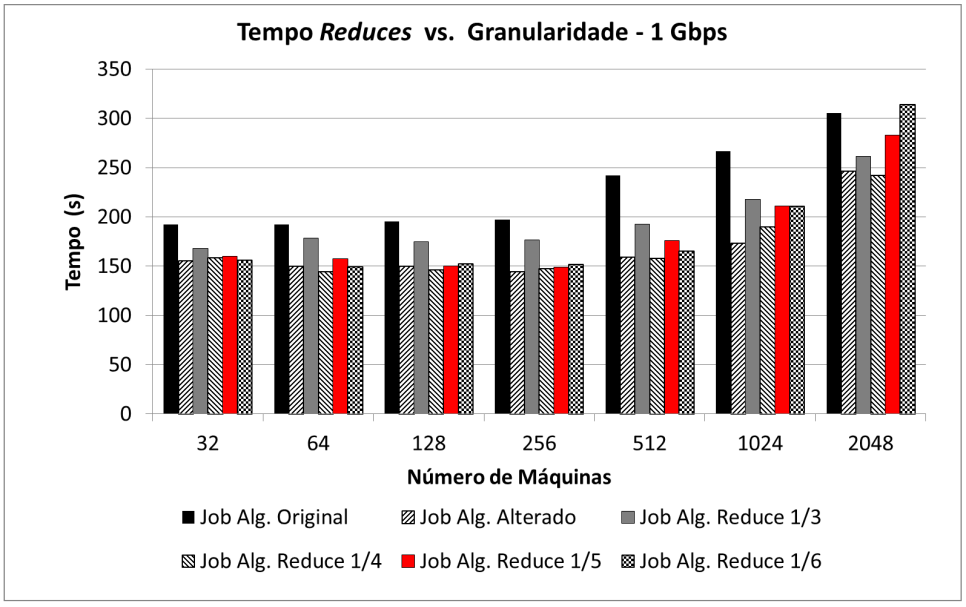


Figura 4.13: Influência da granularidade do *Reduce* @ 1 Gbps

Uma menor granularidade de dados implica em menor tempo de cópia de dados intermediários entre as máquinas que executarão as tarefas *Reduces*. Assim, o tempo de conexão entre cada máquina para transferir as chaves intermediárias é menor. O resultado é um maior paralelismo das tarefas de cópia entre as máquinas, embora a quantidade de dados total seja a mesma.

Observa-se que a abordagem de reduzir o tamanho dos dados resulta em menores tempos na execução das tarefas de *Reduce*. A menor granularidade permite um maior paralelismo das tarefas. Assim, uma máquina mais rápida conclui uma maior quantidade de tarefas menores no mesmo tempo da execução de uma tarefa em uma máquina mais lenta.

Porém, a abordagem pode resultar em tempos de execução inadequados se a granularidade vir a sobrecarregar o barramento de rede com muitas tarefas pequenas, resultando num efeito contrário ao desejado. No teste apresentado na Figura 4.13, um grão maior que 1/4 o sistema irá gerar tempos de execução mais lentos que o desejado.

A Figura 4.14 apresenta a influência da granularidade dos dados da saída do *Map* para os tempos de execução do *Reduce* com uma rede de 10 Mbps. O melhor tempo de execução com os novos algoritmos é 69,73% mais rápido que a execução com os algoritmos originais. O resultado é obtido com um grão de 1/6 do valor da saída de dados do *Map*, o que equivale a 11 MB de dados.

No pior caso, o uso dos novos algoritmos gera um tempo de 16,33% mais rápido que a execução com os algoritmos originais, obtido com um grão de 1/2 do valor da saída. Para os casos do teste da Figura 4.14 a menor granularidade que não afetar o desempenho do sistema é obtida com um grão 1/6, acima deste valor o tempo de execução será maior que o desejado.

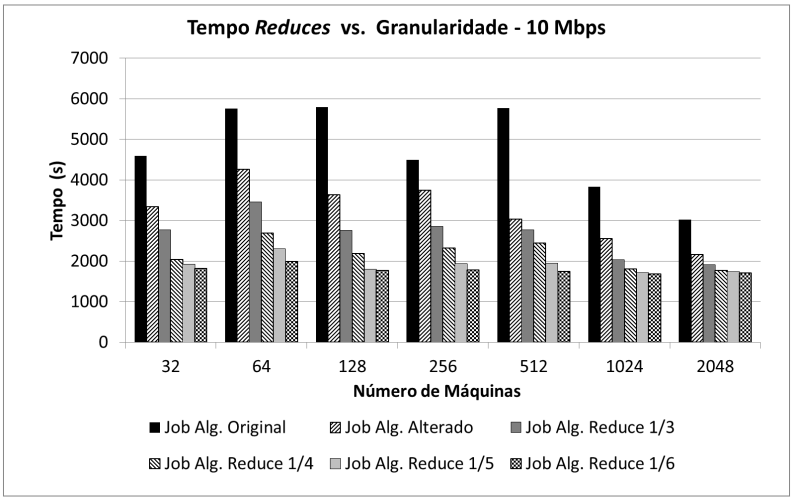


Figura 4.14: Influência da granularidade do *Reduce* @ 10 Mbps

No modelo original, a cópia de dados intermediários em redes de baixa disponibilidade de banda origina desempenhos pobres. Assim, a menor granularidade dos dados da entrada do *Reduce* permite a obtenção de tempos de execução aceitáveis para este tipo de rede.

Para o lançamento de tarefas estar adequado à capacidade computacional de cada máquina foi necessário implementar alterações no escalonamento de tarefas remotas e especulativas. A Figura 4.15 apresenta um comparativo entre o uso ou não de cada um dos algoritmos de lançamento de tarefas em uma rede de 1Gbps.

A curva tracejada superior representa o comportamento dos algoritmos do modelo original. A linha contínua mais abaixo no gráfico, representa o comportamento dos novos algoritmos utilizando os controles de lançamento remoto e especulativo de tarefas. Observa-se que a adequação do lançamento de tarefas remotas e especulativas à capacidade computacional das máquinas resulta no maior desempenho do sistema.

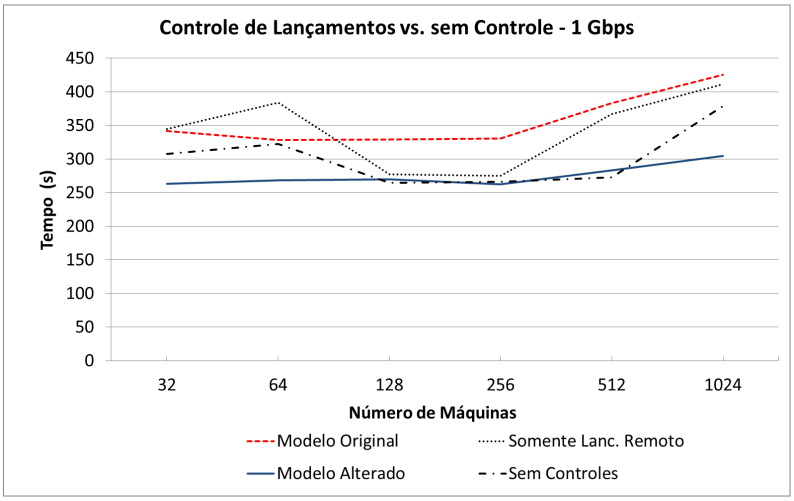


Figura 4.15: Influência do controle de lançamentos de tarefas @ 1 Gbps

Na linha pontilhada observa-se o efeito de utilizar somente o controle de tarefas remotas sem a adequação do lançamento especulativo de tarefas. A Figura 4.15 mostra que os algoritmos não conseguem manter o mesmo comportamento. O mesmo comportamento aparece na linha tracejada intercalada com ponto, onde não há controle algum, nem de lançamentos remotos e nem de lançamentos especulativos.

A Figura 4.16 apresenta a influência do uso dos algoritmos de controle de lançamento de tarefas especulativas e remotas sobre uma rede de 10 Mbps. A linha tracejada representa a execução com os algoritmos originais. A falta do controle dos lançamentos remotos e especulativos é minimizada pela banda da rede. O comportamento da Figura 4.16 comparado com os testes da Figura 4.15 é ligeiramente diferente.

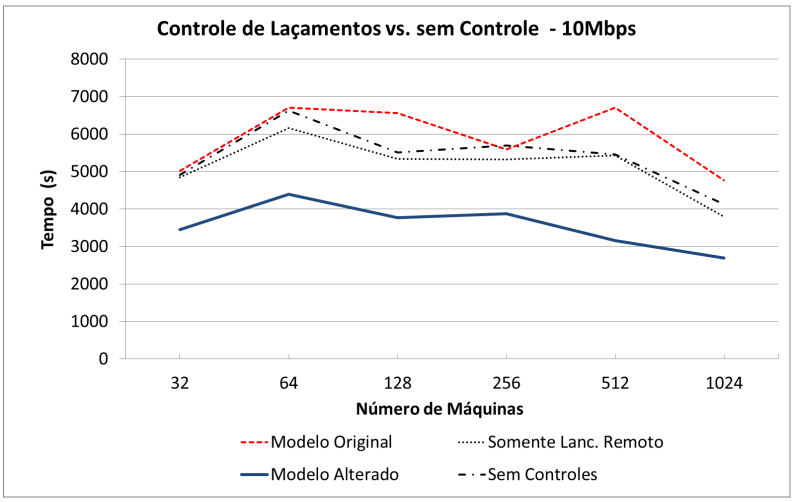


Figura 4.16: Influência do controle de lançamentos de tarefas @ 10 Mbps

Os resultados vistos na linha pontilhada (com uso do controle de tarefas remotas sem a adequação do lançamento especulativo de tarefas) e na linha tracejada intercalada com ponto (sem controles) mostram a importância de utilizar o gerenciamento dos lançamentos especulativos e remotos adequados ao ambiente heterogêneo, para a obtenção de bons resultados de desempenho em relação ao modelo original.

Na Figura 4.17 observa-se a escalabilidade dos algoritmos modificados em ambiente heterogêneo, com alta densidade de máquinas. As execuções dos algoritmos originais e dos modificados apresentam um crescimento linear do tempo de execução.

Duas apresentam o algoritmo original, em uma o *Reduce* processa 100% da entrada do *Map* (linha tracejada) e na outra a saída de dados do *Map* equivale a 50% dos dados da entrada (linha tracejada maior). As outras duas são execuções com os algoritmos modificados, uma a granularidade do *Reduce* é de 1/2, ou seja, o tamanho das tarefas de *Reduce* são menores, mas o *Reduce* processa 100% da entrada do *Map*, porque tem o dobro de tarefas para executar (linha pontilhada). Na outra, a saída do *Map* equivale a 50% da sua entrada (linha cheia).

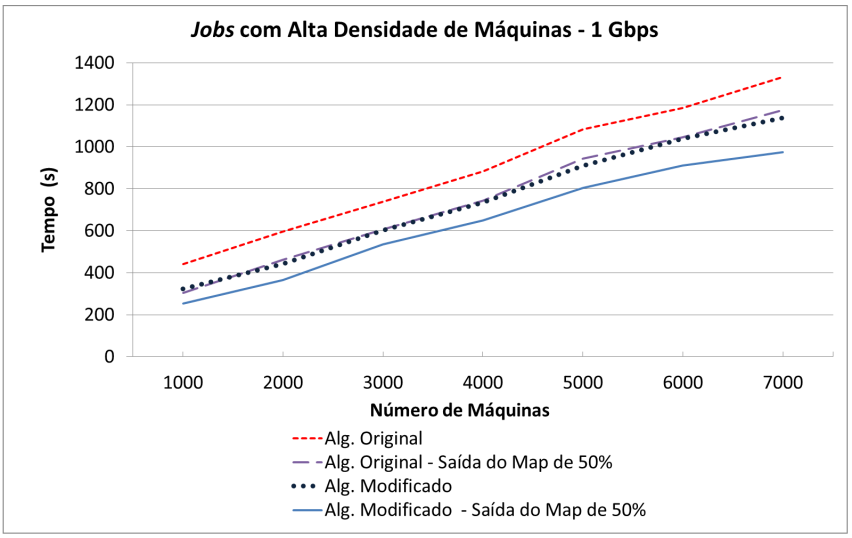


Figura 4.17: Execução de *jobs* com alta densidade de máquinas

Observa-se que o tempo de execução do *job* com os *Reduces* de granularidade igual a 1/2 é aproximadamente semelhante ao tempo da execução do algoritmo original com o *Reduce* processando 50% dos dados da entrada do *Map*. Portanto, a abordagem de redução da granularidade permite um maior paralelismo das execuções de tarefas entre as máquinas, tornando o tempo de execução equivalente ao tempo da execução do *Reduce* com *Map* com a saída de 50% dos dados da entrada.

Outro fator importante que pode ser observado nas Tabelas 4.9.a e 4.9.b é a quantidade de tarefas especulativas. Mesmo com o dobro de tarefas *Reduces*, o número de lançamentos especulativos e remotos ainda é menor, porque nos novos algoritmos o progresso das tarefas está atrelado à capacidade computacional das máquinas.

Na Tabela 4.9.a o número de lançamentos remotos e especulativos é o mesmo para uma saída do *Map* à 50% da sua entrada. Este resultado é devido ao fato de que muitas máquinas são marcadas como *stragglers* devido a heterogeneidade da rede. O mesmo não ocorre com o uso dos algoritmos alterados, porque estão adequados ao ambiente heterogêneo.

Tabela 4.9: Tarefas especulativas e remotas com alta densidade de máquinas

a. Algoritmo original

Máquinas	**1000**	**2000**	**3000**	**4000**	**5000**	**6000**	**7000**
Tarefas remotas *Map*[1]	1099	2390	4966	5322	6794	10397	9234
Tarefas remotas *Map*[2]	1099	2390	4966	5322	6794	10397	9234
Tarefas especulativas *Reduce*[1]	396	982	1400	1992	2524	2991	3461
Tarefas especulativas *Reduce*[2]	396	982	1400	1992	2524	2991	3461

[1]Algoritmo original, [2]Algoritmo original - 50% *Map*

b. Algoritmo modificado

Máquinas	**1000**	**2000**	**3000**	**4000**	**5000**	**6000**	**7000**
Tarefas remotas *Map*[3]	0	0	0	0	0	0	0
Tarefas remotas *Map*[4]	0	0	0	0	0	0	0
Tarefas especulativas *Reduce*[3]	467	822	987	839	1529	746	1147
Tarefas especulativas *Reduce*[4]	21	13	29	19	28	14	20

[3]Novos algoritmos, [4]Novos algoritmos - 50% *Map*

4.5.1 Avaliação dos Custos das Tarefas de Medição

A Figura 4.18 mostra um comparativo do custo da tarefa de medição versus a execução dos *jobs* com os algoritmos originais e com os novos algoritmos, para uma banda de 1 Gbps. O custo total da tarefa de medição é composto do custo da cópia de 1 MB de dados para cada máquina mais o tempo da execução da tarefa de medição.

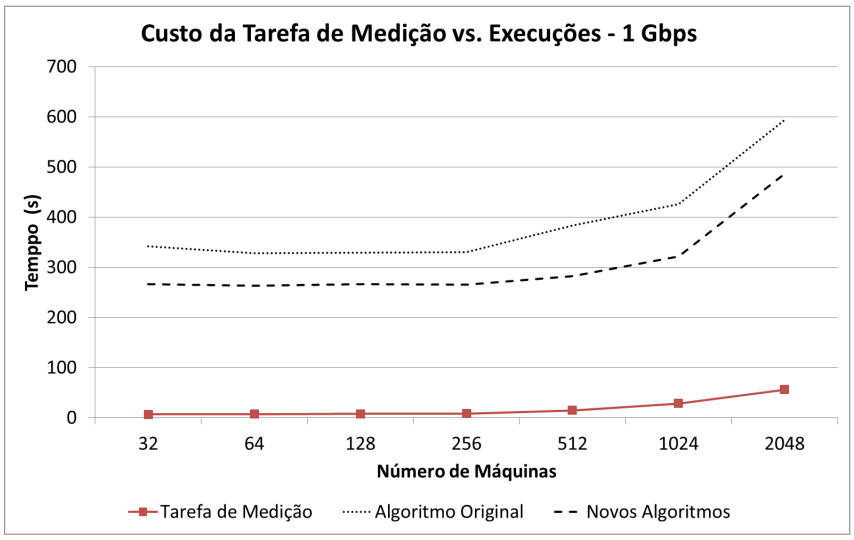


Figura 4.18: Comparativo do custo da tarefa de medição @ 1 Gbps

O custo de transferência t_t é o tempo em segundos necessário para copiar 1 MB de dados através da rede com 1 Gbps de banda. O cálculo do custo da transferência de 1 MB pelas máquinas é feita com base no algoritmo 3.4.1. O tempo para transferir 1 MB de dados para cada máquina é de $t_t = \frac{1024*1024*8}{1000000000}$ segundos. O mestre gasta um tempo de $2^{Md} * t_t$ segundos para transferir os dados para seus filhos e, em cada nível, gasta-se um tempo de $2 * t_t$ segundos para transferir os dados dos pais de cada nível para seus herdeiros.

O tempo de transferência total de 1 MB é o resultado do tempo gasto pelo mestre mais

o tempo gasto nos níveis intermediários, como pode ser verificado na Tabela 4.10. Assim, pelo paralelismo obtido nos subníveis, tem-se um tempo de transferência baixo comprado com o tempo necessário para transferir os dados a partir do mestre para cada uma das máquinas, em uma rede 1 Gbps.

Tabela 4.10: Custo da tarefa de medição @ 1 Gbps

Custos\ Máquinas	32	64	128	256	512	1024	2048
Tempo de *Map* (s)	0,53	0,69	0,91	1,26	2,13	3,20	8,52
Tempo de *Reduce* (s)	6,15	6,28	6,55	7,08	12,07	24,75	46,83
Tempo do *job* (s)	6,68	6,98	7,46	8,34	14,20	27,94	55,35
Tempo de transferência (s)	0,10	0,12	0,18	0,20	0,34	0,35	0,62
Custo total (s)	6,78	7,10	7,64	8,54	14,54	28,29	55,97
Dados transferidos (MB)	32	64	128	256	512	1024	2048

A Figura 4.19 mostra um comparativo do custo da tarefa de medição versus a execução dos *jobs* com os algoritmos originais e com os novos algoritmos, para uma banda de 10 Mbps. Os custos, neste caso, são maiores porque a banda disponível é menor. Entretanto, os tempos de execução do *MapReduce* em redes de 10 Mbps são maiores, assim como o benefício do uso dos novos algoritmos. O custo total da tarefa de medição é composto do custo da transferência de 1 MB de dados mais o tempo de execução da tarefa de medição, ambos verificados na Tabela 4.11.

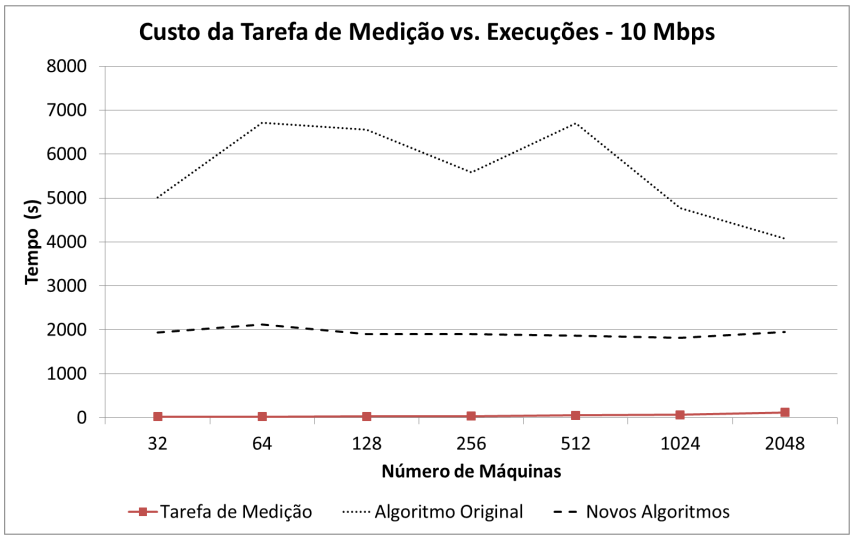


Figura 4.19: Comparativo do custo da tarefa de medição @ 10 Mbps

O tempo da tarefa de medição para 32 máquinas, por exemplo, será a soma do custo de transferência de 1 MB dados para 32 máquinas igual a 10,07 segundos mais o tempo de execução do *job* de medição de 13,04 segundos, isto equivale a um tempo total de 23,11 segundos. O valor é desprezível ao considerar-se que o tempo de execução de um *job* sem a implementação dos novos algoritmos gasta 84 minutos e com os novos algoritmos gasta pouco mais de 32 minutos de execução.

O custo de transferência de 1MB de dados para 2048 máquinas é de 62,08 segundos e o tempo de execução do *job* de medição é de 57,10 segundos, totalizando 1 minuto e 59 segundos para a execução da tarefa de medição. O *job* com os algoritmos originais gasta

68 minutos para ser executado enquanto que com os novos algoritmos gasta pouco mais de 33 minutos.

Tabela 4.11: Custo da tarefa de medição @ 10 Mbps

Custos\ Máquinas	32	64	128	256	512	1024	2048
Tempo de *Map* (s)	0,53	0,69	0,91	1,26	2,13	4,26	8,52
Tempo de *Reduce* (s)	12,50	11,76	11,31	12,90	16,54	25,45	48,58
Tempo do *job* (s)	13,04	12,45	12,22	14,16	18,67	29,71	57,10
Tempo de transferência (s)	10,07	11,74	18,45	20,13	33,55	35,23	62,08
Custo total (s)	23,11	24,19	30,67	34,29	52,22	64,94	119,18
Dados transferidos (MB)	32	64	128	256	512	1024	2048

Os custos iniciais podem ser considerados como custos de auto-configuração do ambiente, necessários para o melhor escalonamento das tarefas. Portanto, como pode ser observado, o custo da tarefa de medição é muito pequeno, praticamente desprezível se comparado com os ganhos obtidos.

Os ganhos ainda podem ser maiores, se for considerado a abordagem do uso do *Map-Reduce Write-Once-Read-Many*. Assim alguns tipos de aplicações podem se beneficiar do uso de *grids* heterogêneas para a execução de *jobs* intensivos em dados.

5 CONSIDERAÇÕES FINAIS

Neste capítulo, serão apresentadas as considerações finais e as conclusões deste trabalho, assim como, algumas ideias que não foram implementadas e que podem servir de fonte de inspiração para trabalhos futuros.

5.1 Conclusões

A Tabela 5.1 apresenta a comparação entre o *MR-A++* e os trabalhos relacionados mais significativos. O *MR-A++* adequa a quantidade de dados processada pelo *Map* à distribuição da capacidade computacional das máquinas. Os dados do *Reduce* são particionados em tamanhos menores segundo o somatório dos fatores de granularidade. Portanto, as máquinas mais rápidas processam mais dados que as mais lentas.

A diminuição do tamanho das partições dos dados intermediários, aumenta a granularidade dos dados e a quantidade de tarefas executadas. Assim, uma fila global de tarefas *Reduce* controla a execução e forcece a localização dos dados para as máquinas livres executarem o processamento. O resultado será um maior paralelismo das tarefas e um menor tempo de execução do *job*.

O controle do lançamento de tarefas especulativas e remotas, permite um controle efetivo da distribuição de tarefas. Assim, evita-se lançar tarefas remotas que não valem a pena serem lançadas. Caso o gerencimento não fosse realizado, os tempos de execução seriam maiores e resultariam em um número maior de tarefas especulativas na última onda de cada fase.

Tabela 5.1: MRA++ vs. Trabalhos Relacionados

Implementação	*MapReduce* 2004 Google	Hadoop 2006 Apache	Hadoop 2008 LATE[1]	Hadoop 2009 Moon	Hadoop 2010 Data Placement	MR-BitDew 2010	MRSG 2012 *MR-A++*
Tam. Dados *Map*	Tam. Def. Prog.	=	=	=	=	Cap. Rede	=
Tam. Dados *Reduce*	Def. Prog.	=	=	=	=	=	$\sum F_g$
Dist. Dados *Map*	Balanc.; Tam. HD	=	=	=	Cap. Proc.	Demanda	Cap. Proc.
Dist. Dados *Reduce*	Nº Maq. *Reduces*	=	=	=	=	=	Fila
Tarefas *Map*	Nº *chunks*	=	=	=	=	Fila	=
Tarefas *Reduce*	*Hash, Nº Maq.*	=	=	=	=	Fila	Granularidade
Replicação Dados	sim	sim	sim	Híbrido	não	sim	sim
Tarefas Especulativas	Prog. Exec.	=	LATE[1]	LATE[1]	não	não	Def. Agrup.
Sist. Arquivos	GFS	HDFS	HDFS	HDFS	HDFS	DHT	DFS

"=" **Mantém o modelo original;** [1] ***Longest Approximate Time to End*****; DHT** → ***Distributed Hash Table*** **sobre P2P e** ***Bittorrent***

O *Mapreduce* é um *framework* utilizado para o tratamento de aplicações intensivas em dados. A programação das aplicações não exige dos desenvolvedores um conhecimento prévio da arquitetura ou topologia da rede e, até mesmo, de como as tarefas devem ser paralelizadas. O conhecimento prévio da quantidade de dados ou do tempo de execução de cada tarefa não são necessários. Entretanto, o modelo criado para grandes *clusters* inviabiliza seu uso em ambientes heterogêneos de forma eficiente e, principalmente, na adoção de *links* com baixa capacidade de banda disponível.

O *MR-A++* não propõe somente a alteração de uma única parte do *MapReduce*, mas sim um conjunto de algoritmos adequados à capacidade computacional das máquinas. Assim, o *MapReduce* poderá ser utilizado em ambientes heterogêneos com *links* lentos e, em especial, na implementação do Hadoop. O *MR-A++* apresenta alterações na forma de dividir os dados, distribuir tarefas, criar agrupamentos e na maneira de controlar o progresso das tarefas.

A maioria dos trabalhos preocuparam-se somente com a fase de *Map* e com o lançamento especulativo de tarefas. As tarefas especulativas ocorrem somente na última onda de tarefas e têm um impacto pequeno no desempenho das execuções. Até o momento, ainda não haviam sido propostas tantas alterações simultaneamente para a maioria dos algoritmos do *MapReduce*, como foi proposto neste trabalho.

As implementações propostas no *MR-A++* preocupam-se com a adequação da carga de trabalho à capacidade computacional de cada máquina e com a redução das tarefas remotas, que ocorrem devido às máquinas ficarem livres para execução de tarefas e não terem dados locais para serem processados. Assim, são executadas cópias de dados de outras máquinas em tempo de execução. As cópias de dados em tempo de execução, principalmente em *links* lentos, afetam o tempo de execução de todo o *job*. Portanto, são uma das causas de baixos desempenhos em ambientes heterogêneos com baixa banda de rede disponível.

A Análise 2^k Fatorial, apresentada na seção 4.4, mostra que a banda influência o comportamento dos *jobs*, assim como, o tamanho dos dados intermediários produzidos pelo *Map*, independentemente da carga de trabalho e do número de máquinas. Por outro lado, com uma alta densidade de máquinas, observa-se um crescimento linear dos tempos de execuções tanto com os algoritmos originais como com os novos algoritmos. Portanto, há um indicativo de que a solução proposta não se esgota rapidamente e, assim, pode ser considerada uma boa solução para o problema da heterogeneidade dos recursos computacionais.

Em ambiente heterogêneo, o novo algoritmo para distribuição de dados, na fase de *Map*, aumenta do desempenho das execuções das tarefas em até 9,3 vezes. Na fase de *Reduce*, a abordagem de redução do tamanho da granularidade dos dados intermediários, produzidos no *Map*, resultou numa melhora de desempenho de até 69,73%. O resultado é um *job* 72,08% mais rápido com os novos algoritmos, comparado com o modelo original. Portanto, em uma rede de 10 Mbps, um *job* de 16 GB de dados, com 32 máquinass, que no modelo original é executado em 84 minutos, passa a ser executado em apenas 32 minutos.

Os experimentos realizados mostram que um *job* de 1 TB de dados, com 2048 máquinas interligadas por *links* de 10 Mbps, tem um tempo de execução de 33 minutos com os novos algoritmos, em contraste com o tempo de 68 minutos obtido no modelo original. Assim, sem considerar a volatilidade, os novos algoritmos apresentam um desempenho satisfatório para a execução das tarefas, viabilizando a implementação de aplicações intensivas em dados sobre *links* de baixa banda disponível, como tipicamente é a Internet. Portanto, o novo modelo permite o uso do *framework* em ambientes formados por máqui-

nas heterogêneas em *grids* como, por exemplo, *desktop grids*.

A política de executar uma tarefa de medição antes da distribuição de dados tem um custo baixo. Por exemplo, para 2048 máquinas, o custo é de apenas 56 segundos, em redes de 1 Gbps, e 57,1 segundos, em redes de 10 Mbps. Os custos desta ordem de grandeza podem ser considerados desprezíveis em face aos benefícios alcançados. Se existirem múltiplos *jobs* com as mesmas funções de *Map* e *Reduce*, sob uma mesma base de dados, também não serão necessárias múltiplas execuções de tarefas de medição.

As soluções propostas no *MR-A++* durante este trabalho permitem um maior balanceamento do processamento das tarefas para o *MapReduce*. O maior balanceamento de processamento ocorre com uma carga de processamento heterogênea, porém adequada à capacidade computacional de cada máquina na *grid*. A solução não se aplica somente a redes de baixa banda disponível, mas também pode ser implementada em ambientes heterogêneos com alta capacidade de banda, como foi demonstrado na seção 4.5. Os ganhos do desempenho do *job* são da ordem de 22% no pior caso e 35,7% no melhor caso, para links rápidos. Se os resultados forem comparados grosseiramente com outros trabalhos, ainda assim, apresentam um rendimento satisfatório, como nos ganhos de 27% a 31% obtidos no trabalho de Zaharia e de 29% obtidos no trabalho de Lin. Ao considerar *links* lentos, os ganhos de desempenho variam de 52,04%, no pior caso e 72,08%, no melhor caso.

Com os resultados obtidos neste trabalho, associados aos baixos custos da implementação do novo modelo, acredita-se que os objetivos da proposta inicial, de adequar o *MapReduce* ao uso em ambientes heterogêneos, tanto em *clusters* como em *desktop grids*, foram alcançados.

5.2 Trabalhos Futuros

Durante este trabalho foram levantadas diversas possibilidades de abordagens que não puderam ser implementadas ou testadas. Algumas ideias são descritas a seguir, porém as possibilidades de estudos certamente não se esgotam nestas observações.

1. **Volatilidade**: Os ambientes heterogêneos, como *desktop grid*, apresentam também características de volatilidade. O controle desta característica ainda encontra-se em aberto. Uma abordagem possível é aplicar preditores de disponibilidade baseados em comportamentos medidos por curvas de distribuição de probabilidade, como *Gamma* e *Weibull*;

2. **Influência de Outros Parâmetros de Desempenho**: Alguns parâmetros não foram implementados separadamente no simulador MRSG. Assim, os custos das tarefas tinham um valor global e estes custos não poderiam ser tratados isoladamente, como por exemplo:

    - Custos relacionados com o tempo de I/O de disco nas estações;
    - Custos de I/O de discos diferentes;
    - Quantidades de *slots*: este é um parâmetro global e poderia agregar valores diferentes conforme o número de núcleos dos processadores;
    - Uso de *links* de diferentes bandas entre as máquinas: este estudo não foi feito por questões de simplificação do modelo.

- Tarefas de medição: no *MR-A++* as tarefas de medição são executadas em intervalos periódicos, por questões de simplificação, utilizadas entre ondas de execução. Outra possibilidade a investigar é a criação de outros mecanismos de medição que possam ser aplicados;

3. **Outras Abordagens de Teste**: Outros testes podem medir a influência da adoção de uma tarefa de *Combiner* nas máquinas antes do envio dos dados intermediários. O simulador não tem um módulo para computar os custos de otimizações na máquina local. A abordagem implica na redução da quantidade de dados a serem enviados para as máquinas na fase de *Shuffle*. Assim, pode-se investigar qual a influência da abordagem da granularidade dos dados intermediários.

4. **Outras Abordagens para o Modelo do *MapReduce***: Poderiam ser consideradas outras formas de configuração, com por exemplo:

   - Topologia das máquinas em ambiente P2P sem um controlador mestre/escravo;
   - Uso de uma árvore de controladores para a distribuição e o gerenciamento de tarefas, para ter-se uma estrutura tolerante a falhas do mestre;
   - Influência dos mecanismos de replicação de dados em ambientes de grande escala de máquinas.

5. **Algoritmos de Divisão dos Dados para o *Reduce***: Pode-se investigar outros modelos de algoritmos além do *bin packing problem*, que possam não ter complexidade *NP-Hard* e que se apliquem a dados heterogêneos;

6. **Interface para a Criação de Aplicações**: A semântica do *MapReduce* dificulta a criação de aplicações complexas. Uma interface que pudesse abstrair melhor a construção de aplicações poderia ser interessante para a adoção deste *framework* nestes tipos de aplicações ou até mesmo uma nova linguagem de programação paralela a partir do *framework*.

REFERÊNCIAS

AL-FURAJH, I.; ALURU, S.; GOIL, S.; RANKA, S. Parallel construction of multidimensional binary search trees. **Parallel and Distributed Systems, IEEE Transactions on**, [S.l.], v.11, n.2, p.136–148, feb 2000.

ANDERSON, D. P. BOINC: a system for public-resource computing and storage. In: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, 2004, Washington, DC, USA. **Anais...** IEEE Computer Society, 2004. p.4–10. (GRID '04).

ANDERSON, D. P.; MCLEOD, J. Local Scheduling for Volunteer Computing. **Parallel and Distributed Processing Symposium, International**, Los Alamitos, CA, USA, v.0, p.477, 2007.

ANJOS, J. C. S.; KOLBERG, W.; ARANTES, L.; GEYER, C. F. R. Estratégias para Uso de MapReduce em Ambientes Heterogêneos. In: CLCAR Conferência Latinoamericana de Computación de Alto Rendimiento, 2010. **Anais...** Evangraf, 2010. v.1, n.1, p.322–325.

BARBOSA, A. F. **Pesquisa sobre o uso das tecnologias de informação e comunicação no Brasil : tic domicílios e tic empresas 2010**. 1.ed. [S.l.]: Comitê Gestor da Internet no Brasil, 2011. v.1, p.1–584.

BERKELEY, U. o. C. **BOINC - Seti@Home Statistics and leaderboards**. Disponível em: <http://setiathome.berkeley.edu/l>. Acesso em: agosto 2012.

BHATNAGAR, S.; HEMACHANDRA, N.; MISHRA, V. K. Stochastic approximation algorithms for constrained optimization via simulation. **ACM Trans. Model. Comput. Simul.**, New York, NY, USA, v.21, p.15:1–15:22, February 2011.

BICAK, M. **MaxxPI System Bench**. [S.l.]: MaxxPI.net, 2011. Disponível em http://www.maxxpi.net, acesso em Novembro 2011.

BORGNAT, P.; DEWAELE, G.; FUKUDA, K.; ABRY, P.; CHO, K. Seven Years and One Day: sketching the evolution of internet traffic. In: INFOCOM 2009, IEEE, 2009. **Anais...** [S.l.: s.n.], 2009. p.711–719.

BOYAR, J.; EPSTEIN, L.; FAVRHOLDT, L. M.; KOHRT, J. S.; LARSEN, K. S.; PEDERSEN, M. M.; WØHLK, S. The maximum resource bin packing problem. **Theor. Comput. Sci.**, Essex, UK, v.362, n.1, p.127–139, Oct. 2006.

BUYYA, R.; BAKER, M.; LAFORENZA, D. **The Grid - International Efforts in Global Computing**. [S.l.]: Grid Computing Info Centre, 2000.

CARDONA, K.; SECRETAN, J.; GEORGIOPOULOS, M.; ANAGNOSTOPOULOS, G. **A Grid Based System for Data Mining Using MapReduce**. [S.l.]: The AMALTHEA REU Program, 2007. Acesso em Março 2011. (Technical Report TR-2007-02).

CHEN, S.; SCHLOSSER, S. W. **Map-Reduce Meets Wider Varieties of Applications**. [S.l.]: Intel Research Pittsburgh, 2008. (IRP-TR-08-05).

CORMEN, T. H.; STEIN, C.; RIVEST, R. L.; LEISERSON, C. E. **Introduction to Algorithms**. 2nd.ed. [S.l.]: McGraw-Hill Higher Education, 2001.

COTTET, F.; DELACROIX, J.; KAISER, C.; MAMMERI, Z. **Scheduling in Real-Time Systems**. [S.l.]: John Wiley & Sons Ltd, 2002.

CROVELLA, M. E.; HARCHOL-BALTER, M.; MURTA, C. D. Task Assignment in a Distributed System - Improving Performance by Unbalancing Load. **SIGMETRICS Performance Evaluation Reveview**, New York, NY, USA, v.26, n.1, p.268–269, 1998.

DEAN, J.; GHEMAWAT, S. MapReduce - Simplified Data Processing on Large Clusters. In: OSDI, 2004. **Anais...** [S.l.: s.n.], 2004. p.137–150.

DEAN, J.; GHEMAWAT, S. MapReduce - A Flexible Data Processing Tool. **Communications of the ACM**, New York, NY, USA, v.53, n.1, p.72–77, 2010.

DONASSOLO, B.; CASANOVA, H.; LEGRAND, A.; VELHO, P. Fast and Scalable Simulation of Volunteer Computing Systems Using SimGrid. In: Workshop on Large-Scale System and Application Performance (LSAP), 2010. **Anais...** [S.l.: s.n.], 2010.

DONGARRA, J. J.; JEANNOT, E.; SAULE, E.; SHI, Z. Bi-objective Scheduling Algorithms for Optimizing Makespan and Reliability on Heterogeneous Systems. In: SPAA '07 - Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, 2007, New York, NY, USA. **Anais...** ACM, 2007. p.280–288.

FANG, W.; HE, B.; LUO, Q.; GOVINDARAJU, N. K. Mars: accelerating mapreduce with graphics processors. **IEEE Transactions on Parallel and Distributed Systems**, Los Alamitos, CA, USA, v.22, p.608–620, 2011.

FEDAK, G.; HE, H.; CAPPELLO, F. BitDew: a programmable environment for large-scale data management and distribution. In: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, 2008, Piscataway, NJ, USA. **Anais...** IEEE Press, 2008. p.45:1–45:12. (SC '08).

FLOYD, R. W.; RIVEST, R. L. Expected time bounds for selection. **Commun. ACM**, New York, NY, USA, v.18, p.165–172, March 1975.

FOSTER, I. T. Globus Toolkit Version 4 - Software for Service-Oriented Systems. In: NPC, 2005. **Anais...** Springer, 2005. p.2–13. (Lecture Notes in Computer Science, v.3779).

GRAEFE, G. Query Evaluation Techniques for Large Databases. **ACM Computer Survey**, [S.l.], v.25, n.2, p.73–170, 1993.

GUFLER, B.; AUGSTEN, N.; REISER, A.; KEMPER, A. Handling Data Skew in MapReduce. In: CLOSER, 2011. **Anais...** DBLP:conf/closer/2011, 2011. p.574–583.

HAMMOUD, S.; LI, M.; LIU, Y.; ALHAM, N. K.; LIU, Z. MRSim: a discrete event based mapreduce simulator. In: FSKD, 2010. **Anais...** IEEE, 2010. p.2993–2997.

HARCHOL-BALTER, M. Task Assignment with Unknown Duration. **Journal of the ACM**, [S.l.], v.49, p.260–288, 1999.

HILBERT, M.; LOPEZ, P.; VASQUEZ, C. Information Societies: measuring the digital information-processing capacity of a society in bits and bytes. **The Information Society**, [S.l.], v.26, p.157–178, May 2010.

INRIA; CNRS. **Grid 5000**. Disponível em https://www.grid5000.fr, acesso em Novembro 2011.

ISARD, M.; BUDIU, M.; YU, Y.; BIRRELL, A.; FETTERLY, D. Dryad - Distributed Data-parallel Programs from Sequential Building Blocks. In: EuroSys '07 - Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, 2007, New York, NY, USA. **Anais...** ACM, 2007. p.59–72.

JAIN, R. **The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling**. [S.l.]: Wiley, 1991. I–XXVII, 1–685p. (Wiley professional computing).

JAVADI, B.; KONDO, D.; VINCENT, J.-M.; ANDERSON, D. P. **Mining for Availability Models in Large-Scale Distributed Systems: A Case Study of SETI@home**. [S.l.]: INRIA, 2009. Research Report.

KAVULYA, S.; TAN, J.; GANDHI, R.; NARASIMHAN, P. An Analysis of Traces from a Production MapReduce Cluster. In: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010, Washington, DC, USA. **Anais...** IEEE Computer Society, 2010. p.94–103. (CCGRID '10).

KO, S. Y.; HOQUE, I.; CHO, B.; GUPTA, I. On Availability of Intermediate Data in Cloud Computations. **12th Workshop on Hot Topics in Operating Systems**, [S.l.], 2009.

KOLBERG, W.; ANJOS, J. C. S. **MRSG: a mapreduce simulator for desktop grids**. [S.l.]: Instituto de Informática, UFRGS, 2011. FL 5103. (364).

KONDO, D.; JAVADI, B.; MALECOT, P.; CAPPELLO, F.; ANDERSON, D. P. Cost-benefit Analysis of Cloud Computing versus Desktop Grids. In: IPDPS '09 - Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing, 2009, Washington, DC, USA. **Anais...** IEEE Computer Society, 2009. p.1–12.

LAM, C. **Hadoop in Action**. [S.l.]: Manning Publications, 2011. (Manning Pubs Co Series).

LIN, H.; ARCHULETA, J.; MA, X.; FENG, W.-c.; ZHANG, Z.; GARDNER, M. **MOON - MapReduce On Opportunistic eNvironments**. [S.l.]: Virginia Polytechnic Institute and State University, 2009.

MACKEY, G.; SEHRISH, S.; LOPEZ, J.; BENT, J.; HABIB, S.; WANG, J. Introducing Map-Reduce to High End Computing. In: Petascale Data Storage Workshop at SC08, 2008, Austin, Texas. **Anais...** [S.l.: s.n.], 2008.

MICELI, C.; MICELI, M.; JHA, S.; KAISER, H.; MERZKY, A. Programming Abstractions for Data Intensive Computing on Clouds and Grids. **9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009. CCGRID '09.**, [S.l.], v.1, n.9, p.478–483, May 2009.

MORETTI, C.; BUI, H.; HOLLINGSWORTH, K.; RICH, B.; FLYNN, P.; THAIN, D. All-Pairs - An Abstraction for Data-Intensive Computing on Campus Grids. **IEEE Trans. Parallel Distrib. Syst.**, Piscataway, NJ, USA, v.21, n.1, p.33–46, 2010.

OLSTON, C.; REED, B.; SRIVASTAVA, U.; KUMAR, R.; TOMKINS, A. Pig Latin - A Not-so-foreign Language for Data Processing. In: SIGMOD '08 - Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008, New York, NY, USA. **Anais...** ACM, 2008. p.1099–1110.

RANGER, C.; RAGHURAMAN, R.; PENMETSA, A.; BRADSKI, G.; KOZYRAKIS, C. Evaluating MapReduce for Multi-core and Multiprocessor Systems. In: HPCA '07: Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture, 2007, Washington, DC, USA. **Anais...** IEEE Computer Society, 2007. p.13–24.

ROSS, S. **Probabilidade: um curso moderno com aplicações**. 8ª.ed. [S.l.]: Bookman Editora, 2010. v.1.

SALBAROLI, F. **Fault Tolerant Hadoop Job Tracker**. [S.l.]: The Apache Software Foundation, 2009. (MAPREDUCE-225).

SIMGRID. **SimGrid Home Page**. Disponível em: <http://simgrid.gforge.inria.fr/>. Acesso em: agosto 2011.

SWIFT - Scalable Parallel Scripting for Scientific Computing, 2010. **Anais...** IOP Publishing, 2010. p.38–51. (Spring 2010, v.17).

TANG, B.; MOCA, M.; CHEVALIER, S.; HE, H.; FEDAK, G. Towards MapReduce for Desktop Grid Computing. **P2P, Parallel, Grid, Cloud, and Internet Computing, International Conference on**, Los Alamitos, CA, USA, v.0, p.193–200, 2010.

TANG, H. **Mumak: map-reduce simulator**. [S.l.]: Apache Software Foundation, 2009. [ONLINE], Disponível em https://issues.apache.org/jira/browse/MAPREDUCE-728. (Technical Report MAPREDUCE-728).

THAIN, D.; TANNENBAUM, T.; LIVNY, M. Condor and the Grid. In: Grid Computing - Making the Global Infrastructure a Reality, 2003. **Anais...** John Wiley, 2003.

UCAR, B.; AYKANAT, C.; KAYA, K.; IKINCI, M. Task Assignment in Heterogeneous Computing Systems. **Journal Parallel Distributed Computer**, Orlando, FL, USA, v.66, n.1, p.32–46, 2006.

VELHO, P.; LEGRAND, A. Accuracy study and improvement of network simulation in the SimGrid framework. In: Proceedings of the 2nd International Conference on Simulation Tools and Techniques, 2009, ICST, Brussels, Belgium, Belgium. **Anais...** ICST (Institute for Computer Sciences: Social-Informatics and Telecommunications Engineering), 2009. p.13:1–13:10. (Simutools '09).

VENNER, J. **Pro Hadoop - Build Scalable, Distributed Applications in the Cloud**. 1.ed. [S.l.]: Apress, Inc., 2009. v.1.

WANG, G.; BUTT, A. R.; PANDEY, P.; GUPTA, K. Using Realistic Simulation for Performance Analysis of MapReduce Setups. In: LSAP '09: Proceedings of the 1st ACM workshop on Large-Scale System and Application Performance, 2009, New York, NY, USA. **Anais...** ACM, 2009. p.19–26.

WHITE, T. **Hadoop - The Definitive Guide**. 1.ed. [S.l.]: OReilly Media, Inc., 2009. v.1.

WILDE, M.; FOSTER, I.; ISKRA, K.; BECKMAN, P.; ZHANG, Z.; ESPINOSA, A.; HATEGAN, M.; CLIFFORD, B.; RAICU, I. Parallel Scripting for Applications at the Petascale and Beyond. **Computer**, Los Alamitos, CA, USA, v.42, p.50–60, 2009.

XIE, J.; YIN, S.; RUAN, X.; DING, Z.; TIAN, Y.; MAJORS, J.; MANZANARES, A.; QIN, X. Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters. In: IEEE International Symposium on Parallel and Distributed Processing, 2010. **Anais...** [S.l.: s.n.], 2010. p.1–9. (Workshops and Phd Forum (IPDPSW)).

YOU, H.-H.; YANG, C.-C.; HUANG, J.-L. A load-aware scheduler for MapReduce framework in heterogeneous cloud environments. In: Proceedings of the 2011 ACM Symposium on Applied Computing, 2011, New York, NY, USA. **Anais...** ACM, 2011. p.127–132. (SAC '11).

ZAHARIA, M.; KONWINSKI, A.; JOSEPH, A. D.; KATZ, Y.; STOICA, I. Improving MapReduce Performance in Heterogeneous Environments. **OSDI**, [S.l.], Dec 2008.

ANEXO A CONFIGURAÇÕES

A.1 Configurações dos Equipamentos

O Simgrid tem um *script* próprio para gerar os arquivos de plataforma com as configurações a partir de parâmetros definidos. As configurações de equipamentos foram gerados a partir de uma distribuição normal, como a técnica utilizada no trabalho de Javadi (JAVADI et al., 2009). As plataformas foram criadas com uma capacidade variando de $2e^9$ até $5e^9$ *flops*. As Tabelas A.1, A.2 , A.3 e A.4 mostram as descrições dos equipamentos compatíveis com a capacidade computacional de cada máquina gerada pelos *scripts* de criação de plataforma para os diversos experimentos.

Tabela A.1: Configuração de Estações para Experimento com 64 Máquinas

Configuração do Equipamento	Quantidade	Configuração do Equipamento	Quantidade
Intel Xeon 3040 (1860MHz) – 2GB RAM	1	Pentium Dual-Core E6300 (2800MHz) – 2GB RAM	2
Intel Celeron E1500 (2200MHz) – 2GB RAM	2	Pentium Dual-Core E6300 (2800MHz) – 4GB RAM	1
Dual-Core AMD Opteron 1214 (2200MHz) – 2GB RAM	1	AMD Phenom II X2 555 (3200MHZ) – 4GB RAM	3
Intel Core i5 430UM (1200MHz) – 4 GM RAM	1	Pentium Dual-Core E5700 (3000MHZ) – 2GB RAM	1
Intel Core2 Duo T7500 (2200MHz) – 2GB RAM	1	Intel Pentium G6950 (3010MHz) – 2GB RAM	3
AMD Athlon 64 X2 Dual Core 4800 (2500MHz) – 2GB RAM	1	Intel Core2 Duo P9700 (2800 MHz) – 2GB RAM	2
Intel Xeon 3050 (2130 MHz) – 2GB RAM	3	Intel Core2 Duo P9700 (2800 MHz) – 4GB RAM	2
Intel Core2 Duo E6540 (2330 MHz) – 2GB RAM	1	Intel Core i3 370M (2400MHz) – 4GB RAM	2
Intel Core2 Duo E4600 (2400MHz) – 2GB RAM	2	AMD Athlon X3 440 (3000MHz) – 4GB RAM	2
Intel Core2 Duo E4600 (2400MHz) – 4GB RAM	1	AMD Phenom II X3 720 (3200MHz) – 4GB RAM	2
Dual Core AMD Opteron 185 (2600MHz) – 2GB RAM	2	Intel Core2 Duo E8500 (3160 MHz) – 2GB RAM	1
Intel Pentium Dual Core E5300 (2616MHz) – 2GB RAM	1	Intel Celeron G540 (2.500 MHz) – 2GB RAM	3
Intel Pentium P6200 (2128MHz) – 2GB RAM	1	AMD Phenom II X3 720 (3335MHz) – 4GB RAM	1
Intel Core 2 Duo E6550 (3002MHz) – 2GB RAM	1	Intel Core i5-2537M (1400MHz) – 4GB RAM	1
Intel Xeon 3065 (2330MHz) – 2GB RAM	2	AMD Phenom II X3 720 (3600MHz) – 4GB RAM	1
Intel Xeon 3065 (2330MHz) – 4GB RAM	2	Intel Xeon E5335 (2000MHz) – 4GB RAM	1
Intel Core2 Duo E6750 (2660MH) – 2GB RAM	3	Intel Xeon X3210 (2130MHz) – 4GB RAM	2
AMD Phenom II N640 Dual-Core (2900MHz) – 4GB RAM	3	AMD Phenom II X3 720 (3816MHz) – 4GB RAM	1
Intel Core i7 640M (2800MHz) – 4GB RAM	3	Pentium Dual-Core E6300 (2660MHz) – 2GB RAM	1

Tabela A.2: Configuração de Estações para Experimento com 32 Máquinas

Configuração do Equipamento	Quantidade	Configuração do Equipamento	Quantidade
Intel Xeon 3050 (2130 MHz) – 2GB RAM	1	Intel Pentium Dual Core E6500 (3520MHz) – 4GB RAM	1
Intel Core2 Duo E6540 (2330 MHz) – 2GB RAM	1	AMD Athlon II X2 250 (3750MHz) – 4 GM RAM	1
Intel Core2 Duo E6540 (2330 MHz) – 4GB RAM	1	Intel Core 2 Duo E6600 (3700MHz) – 2GB RAM	2
Intel Pentium Dual Core E5300 (2616MHz) – 2GB RAM	1	Intel Core 2 Duo E6600 (3700MHz) – 4GB RAM	3
Intel Core 2 Duo E6550 (3002MHz) – 3GB RAM	1	AMD Phenom II X3 720 (2809MHz) – 4GB RAM	2
AMD Phenom II X2 550 (3100MHz) – 4GB RAM	2	AMD Phenom II X3 720 (3200MHz) – 4GB RAM	1
Intel Core 2 Duo E8200 (3200MHz) – 4GB RAM	2	AMD Phenom II X3 720 (3335MHz) – 4GB RAM	1
AMD Athlon II X2 255 (3410MHz) – 2GB RAM	1	AMD Phenom II X3 720 (3600MHz) – 2GB RAM	2
AMD Phenom II 550 (3567MHz) – 2GB RAM	1	Intel Xeon L5320 (1800MHz) – 4GB RAM	1
AMD Phenom II 550 (3567MHz) – 4GB RAM	1	Intel Xeon E5335 (2000MHz) – 4GB RAM	2
Pentium Dual-Core E5400 (2700 MHz) – 4GB RAM	1	Intel Xeon X3210 (2130MHz) – 4GB RAM	1
Intel Core2 Duo E8135 (2660 MHz) – 4GB RAM	1	AMD Phenom II X3 720 (3816MHz) – 4GB RAM	1

Tabela A.3: Configuração de Estações para Experimento com 128 Máquinas

Configuração do Equipamento	Quantidade	Configuração do Equipamento	Quantidade
Intel Core2 Duo T7300 (2000MHz) – 2GB RAM	1	Intel Core2 Duo E8200 (2660MHZ) – 4GB RAM	1
Intel Pentium Dual T3400 (2160MHz) – 4GB RAM	1	Intel Pentium B950 (2100MHZ) – 2GB RAM	2
AMD Sempron Dual Core 2300MHz – 2GB RAM	1	Pentium Dual-Core E5700 (3000MHZ) – 2GB RAM	2
AMD Turion II Neo K685 Dual-Core (1800MHz) – 2GB RAM	1	Intel Core i3 330M (2130MHz) – 2GB RAM	2
Intel Celeron E1500 (2200MHz) – 2GB RAM	2	AMD Phenom II X2 560 (3300MHz) – 2GB RAM	3
Dual-Core AMD Opteron 1214 (2200MHz) – 2GB RAM	1	Intel Core2 Duo T9600 (2800MHz) – 4GB RAM	2
Pentium Dual-Core E5800 (3200MHz) – 4GB RAM	1	Intel Core i7 640LM (2130MHz) – 4GB RAM	2
Intel Core2 Duo T7500 (2200MHz) – 2GB RAM	1	Intel Core i3 350M (2270MHz) – 2GB RAM	2
AMD Athlon 64 X2 Dual Core 4800 (2500MHz) – 2GB RAM	1	Intel Core2 Duo P9700 (2800MHz) – 4GB RAM	6
Intel Xeon 3050 (2130 MHz) – 2GB RAM	2	Intel Core2 Duo T9800 (2930MHz) – 4GB RAM	2
Intel Core2 Duo E6540 (2330 MHz) – 2GB RAM	2	Pentium Dual-Core E6700 (3200 MHz) – 4GB RAM	3
Intel Core2 Duo E4600 (2400MHz) – 4GB RAM	2	Intel Core 2 Duo E8400 (3000MHz) – 4GB RAM	3
Dual Core AMD Opteron 185 (2600MHz) – 2GB RAM	2	Intel Core2 T9550 (2660MHz) – 4GB RAM	1
Intel Pentium P6200 (2128MHz) – 2GB RAM	3	Intel Pentium G630T (3300MHz) – 2GB RAM	2
AMD Phenom II X2 550 (3100MHz) – 4GB RAM	5	Intel Pentium B960 (2200MHz) – 4GB RAM	2
Intel Core 2 Duo E6550 (3002MHz) – 2GB RAM	3	Intel Core i3 380M (2530MHz) – 4GB RAM	2
Intel Xeon 3065 (2330MHz) – 2GB RAM	5	Intel Xeon L5240 (3000 MHz) – 2GB RAM	3
Intel Core2 Duo E6750 (2660MH) – 2GB RAM	3	Intel Celeron G540 (2.500 MHz) – 2GB RAM	3
AMD Phenom II N640 Dual-Core (2900MHz) – 4GB RAM	7	Intel Core i5 560M (2670MHz) – 4GB RAM	1
Intel Xeon 3060 (2400MHz) – 2GB RAM	2	Intel Core i5-2537M (1400MHz) – 2GB RAM	2
Pentium Dual-Core E5200 (2500MHz) – 2GB RAM	3	Intel Xeon E5335 (2000MHz) – 4GB RAM	4
Pentium Dual-Core E6300 (2660MHz) – 2GB RAM	1	AMD Phenom II X3 B75 (3000MHz) – 4GB RAM	3
Pentium Dual-Core E5400 (2700MHz) – 2GB RAM	2	Intel Core i3-2310M (2100MHz) – 2GB RAM	2
Intel Core2 X6800 (2930MHz) – 2GB RAM	3	Intel Xeon X5260 (3330MHz) – 2GB RAM	3
AMD Phenom II N660 Dual-Core (3000MHz) – 2GB RAM	3	Intel Core i5-2557M (1700MHz) – 2GB RAM	3
AMD Phenom II X2 B59 (3400MHZ) – 4GB RAM	1	Intel Core i3-2330M (2200MHz) – 4GB RAM	4
Intel Core i7 640M (2800MHz) – 4GB RAM	4		

Tabela A.4: Configuração de Estações para Experimento com 256 Máquinas

Configuração do Equipamento	Quantidade	Configuração do Equipamento	Quantidade
Intel Core2 Duo T7300 (2000MHz) – 2GB RAM	4	AMD Turion II P560 Dual-Core (2500MHz) – 2GB RAM	3
AMD Sempron Dual Core 2300MHz – 2GB RAM	2	Intel Core i7 620LM (2000MHz) – 4GB RAM	3
AMD Turion II Neo K685 Dual-Core (1800MHz) – 2GB RAM	1	Intel Celeron P4600 (2000MHz)– 2GB RAM	4
Intel Celeron E1500 (2200MHz) – 2GB RAM	3	Intel Core2 Extreme X7800 (2600MHz) – 2GB RAM	2
Dual-Core AMD Opteron 1214 (2200MHz) – 2GB RAM	2	Intel Core2 Duo E6700 (2660MHz) – 2GB RAM	4
Dual-Core AMD Opteron 1214 (2200MHz) – 4GB RAM	2	AMD Athlon 64 X2 Dual Core 5800+ (3000MHz) – 2GB RAM	1
Intel Core2 Duo E4500 (2200MHz) – 2GB RAM	3	AMD Athlon II X2 B24 (3000MHz) – 2GB RAM	2
AMD Sempron X2 180 (2400MHz) – 2GB RAM	2	Intel Pentium P6200 (2130MHz) – 4GB RAM	2
AMD Athlon Dual Core 5000B (2600MHz) – 2GB RAM	2	Intel Xeon 5148 (2330MHz) – 4GB RAM	2
AMD Athlon 64 X2 Dual Core 4800 (2500MHz) – 2GB RAM	3	AMD Phenom II N620 Dual-Core (3000MHz) – 4GB RAM	2
Intel Xeon 3050 (2130 MHz) – 2GB RAM	1	Intel Core2 Duo T9300 (2500MHz) – 4GB RAM	2
Intel Core2 Duo E6420 (2130MHz) – 2GB RAM	2	Intel Core2 Duo E8135 (2400MHz) – 2GB RAM	2
Intel Core2 Duo E6540 (2330 MHz) – 2GB RAM	2	Intel Core2 Duo E7200 (2530MHz) – 2GB RAM	2
Intel Core2 Duo T6400 (2000MHz) – 4GB RAM	2	AMD Phenom II N640 Dual-Core (2900MHz) – 4GB RAM	5
Dual-Core AMD Opteron 2214 (2200MHz) – 4GB RAM	3	Intel Xeon 3060 (2400MHz) – 4GB RAM	3
Intel Core2 Duo E4600 (2400MHz) – 2GB RAM	3	Intel Core2 Duo P9600 (2530MHz) – 2GB RAM	3
Intel Pentium Dual E2220 (2400MHz) – 2GB RAM	3	Intel Core2 Duo P8700 (2530MHz) – 4GB RAM	2
Intel Core2 Duo P7350 (2000MHz) – 2GB RAM	1	AMD Phenom II P860 Triple-Core (2000MHz) – 4GB RAM	1
Dual Core AMD Opteron 185 (2600MHz) – 2GB RAM	3	Intel Core2 Duo T9500 (2600MHz) – 2GB RAM	4
Intel Pentium P6000 (1870MHz) – 2GB RAM	2	Intel Core i3-2367M (1400MHz) – 2GB RAM	4
Intel Core2 Duo E6550 (2330MHz) – 2GB RAM	3	AMD Athlon II X2 260 (3200MHz) – 2GB RAM	3
Intel Core2 Duo T7800 (2600MHz – 4GB RAM	4	Pentium Dual-Core E6300 (2800MHz) – 4GB RAM	2
Intel Core2 Duo E6600 (2400MHz) – 2GB RAM	4	Pentium Dual-Core E5500 (2800MHz) – 2GB RAM	2
AMD Athlon II X2 B22 (2800MHz) – 4GB RAM	2	AMD Athlon II X2 4450e (2300MHz) – 2GB RAM	2
AMD Phenom II N830 Triple-Core (2100MHz) – 2GB RAM	2	Intel Core i3 390M (2670MHz) – 2GB RAM	2
Intel Xeon 5160 (3000MHz) – 4GB RAM	2	Pentium Dual-Core E6800 (3330MHz) – 2GB RAM	4
Pentium Dual-Core E5700 (3000MHz) – 4GB RAM	2	Intel Celeron G540 (2.500 MHz) – 2GB RAM	6
AMD Phenom II N870 Triple-Core (2300MHz) – 2GB RAM	2	Intel Core i5 540M (2530MHz) – 2GB RAM	4
AMD Phenom 8750B Triple-Core (2400MHz) – 2GB RAM	3	Intel Core i5 560M (2670MHz) – 2GB RAM	3
Intel Pentium G6950 (2800MHz) – 4GB RAM	6	Intel Core i5 560M (2670MHz) – 4GB RAM	6
Intel Core i7 640LM (2130MHz) – 2GB RAM	3	AMD Phenom II X3 720 (3600MHz) – 4GB RAM	4
Intel Core2 Duo E8235 (2800MHz) – 2GB RAM	2	Intel Core i5-2537M (1400MHz) – 4GB RAM	4
Intel Core i3 350M (2270MHz) – 2GB RAM	2	AMD Athlon II X3 440 (3000MHz) – 2GB RAM	2
Intel Core2 Duo E7600 (3060MHz) – 4GB RAM	3	AMD Athlon II X3 440 (3000MHz) – 4GB RAM	3
Intel Core 2 Duo E8600 (4200MHz) – 4GB RAM	3	Intel Core i5 580M (2670MHz) – 4GB RAM	2
AMD Athlon II X3 415e (2500MHz) – 2GB RAM	1	Intel Xeon E5335 (2000MHz) – 2GB RAM	3
Intel Core2 Duo T9800 (2930MHz) – 4GB RAM	2	AMD Athlon II X3 435 (2900MHz) – 2GB RAM	2
Pentium Dual-Core E5800 (3200MHz) – 4GB RAM	5	Intel Xeon X3210 (2130MHz) – 4GB RAM	3
AMD Phenom 8850B Triple-Core (2500MHZ) – 2GB RAM	2	Intel Xeon X5260 (3330MHz) – 2GB RAM	2
Intel Xeon E5603 (1600MHz) – 4GB RAM	3	Intel Core i3 530 (2930MHz) – 2GB RAM	4
Intel Core i3 370M (2400MHz) – 2GB RAM	2	Intel Core i3 530 (2930MHz) – 4GB RAM	6
Intel Pentium G630T (2300MHz) – 4GB RAM	1	Intel Pentium G850 (2900MHz) – 2GB RAM	2
Intel Pentium B960 (2200MHz) – 4GB RAM	2	Intel Core i3-2330M (2200MHz) – 2GB RAM	2
Intel Core i5 520M (2400MHz) – 4GB RAM	3	Intel Core i7 640M (2800MHz) – 4GB RAM	3
AMD A8-3520M APU (1600MHz) – 2GB RAM	2	AMD Phenom II X3 740 (3000MHz) – 2GB RAM	5
Intel Core i5 480M (2670MHz) – 2GB RAM	5	Intel Xeon X5272 @ 3.40GHz (3400MHz) – 2GB RAM	2
Intel Core i3 380M (2530MHz) – 3GB RAM	3		