MIGRAÇÃO DO CHIRON PARA AMBIENTE DE PROCESSAMENTO PARALELO COM MEMÓRIA DISTRIBUÍDA

João Luiz Reis Ferreira

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Alexandre de Assis Bento Lima

Rio de Janeiro

Setembro de 2014

MIGRAÇÃO DO CHIRON PARA AMBIENTE DE PROCESSAMENTO PARALELO COM MEMÓRIA DISTRIBUÍDA

João Luiz Reis Ferreira

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____
Prof. Alexandre de Assis Bento Lima, D.Sc.

_____
Profa. Marta Lima de Queirós Mattoso, D.Sc.

_____
Profa. Vanessa Braganholo Murta, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO DE 2014

Ferreira, João Luiz Reis

Migração do Chiron para Ambiente de Processamento Paralelo com Memória Distribuída / João Luiz Reis Ferreira. – Rio de Janeiro: UFRJ/COPPE, 2014.

XIV, 79 p.: il.; 29,7 cm.

Orientador: Alexandre de Assis Bento Lima

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2014.

Referências Bibliográficas: p. 67-69.

1. *Workflows* Científicos. 2. Memória Distribuída. 3. Processamento de Alto Desempenho. I. Lima, Alexandre de Assis Bento. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

MIGRAÇÃO DO CHIRON PARA AMBIENTE DE PROCESSAMENTO PARALELO COM MEMÓRIA DISTRIBUÍDA

João Luiz Reis Ferreira

Setembro/2014

Orientador: Alexandre de Assis Bento Lima
Programa: Engenharia de Sistemas e Computação

Experimentos científicos que manipulam dados em larga escala costumam ser representados como *workflows* científicos, de modo a facilitar o encadeamento de programas que desempenham funções específicas no processamento desses dados. Sistemas Gerenciadores de *Workflows* Científicos (SGWfC) apoiam a configuração, execução e monitoramento de *workflows* científicos. Para analisar grandes conjuntos de dados, é importante que o SGWfC ofereça suporte à execução paralela em ambientes de processamento de alto desempenho (PAD). Dentre os sistemas com essa característica, o Chiron se tornou o objeto de pesquisa desta dissertação, graças à sua coleta de proveniência e à sua álgebra, que viabiliza otimizações no plano de execução dos *workflows*. Outro aspecto do Chiron é a necessidade de disco compartilhado para seu pleno funcionamento. Para aproveitar a escalabilidade oferecida por ambientes de memória distribuída e ampliar o total de plataformas onde o Chiron pode ser utilizado, esta dissertação apresenta o processo de criação do ChironSN, uma versão modificada do Chiron apta a operar em tais ambientes. A fim de validar o protótipo, programas científicos de Map/Reduce, muito comuns em ambientes de memória distribuída, foram modelados como *workflows* e executados com o ChironSN.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

CHIRON MIGRATION TO SHARED-NOTHING PARALLEL PROCESSING ENVIRONMENTS

João Luiz Reis Ferreira

September/2014

Advisor: Alexandre de Assis Bento Lima
Department: Systems and Computing Engineering

Scientific experiments that handle large-scale datasets are typically represented as scientific workflows, which eases the chaining of programs that perform specific tasks to process the dataset. Scientific Workflow Management Systems (SWfMS) helps to configure, execute and monitor scientific workflows. Massive dataset analysis demands SWfMS to support parallel execution on High Performance Computing (HPC) environments. Among the systems with this feature, Chiron became the research object of this dissertation, thanks to its provenance gathering capabilities and its workflow algebra, which allows for optimizations on scientific workflows' execution plans. Another aspect of Chiron is the need for shared disk to its full operation. To take advantage of the scalability offered by shared-nothing environments and expand the amount of platforms where Chiron can be used, this dissertation presents the creation process of ChironSN, a modified version of Chiron which is able to work in such environments. In order to validate the prototype, Map/Reduce scientific programs, very popular in shared-nothing environments, were modeled as workflows and executed by ChironSN.

Dedicatória

À Maria Helena Craveiro de Amorim

(in memoriam)

Agradecimentos

Agradeço ao meu orientador Alexandre de Assis Bento Lima, pela orientação precisa, pela paciência, pela motivação transmitida, pelos e-mails respondidos quase que instantaneamente.

Agradeço à minha família, que me ajudou a chegar até aqui, e já está pensando no próximo destino.

Agradeço aos meus amigos, pela companhia, parceria, gargalhadas. Menção honrosa aos Meus Queridos: Monclar, Pivotto, Thiago, Gustavo, Renan, Roberta e Juliana. Segunda menção honrosa ao Gustavo, que partilhou dos momentos de bonança e agrura nesse período de Mestrado.

Agradeço aos professores e funcionários do Programa de Engenharia de Sistemas e Computação, pelo conhecimento recebido, pela dedicação, pela ajuda com os serviços burocráticos.

Agradeço ao Grid’5000 pela oportunidade de utilizar a infraestrutura computacional e conduzir os experimentos científicos necessários à validação da proposta desta dissertação.

Por fim, agradeço ao CNPq, cujo apoio financeiro foi muito importante durante o período do Mestrado.

Sumário

Índice de Figuras

Índice de Tabelas

Lista de Siglas

API *Application Programming Interface*

HDFS *Hadoop Distributed File System*

MPI *Message Passing Interface*

PAD Processamento de Alto Desempenho

SGBD Sistema Gerenciador de Banco de Dados

SGWfC Sistema Gerenciador de *Workflows* Científicos

Capítulo 1 - Introdução

Ao longo dos últimos anos, o volume de dados manipulados por experimentos científicos apresenta altas taxas de crescimento. Felizmente, as infraestruturas computacionais para processamento de alto desempenho (PAD) estão acompanhando esse crescimento e permitindo que os cientistas analisem amostras de quaisquer tamanhos. Para cada uma das três principais categorias de ambientes de PAD – *clusters*, grades e nuvens – há várias opções à disposição do cientista.

Se o volume de dados a ser analisado aumentou, a complexidade dos experimentos utilizados para essas análises também tem aumentado. O encadeamento de programas necessários para completar uma análise complexa pode ser uma barreira, principalmente se o cientista precisar gerenciar manualmente as execuções de cada programa, o que inclui a manipulação dos dados intermediários (GIL *et al.*, 2007). Os *workflows* científicos surgiram como uma abstração para representar esse encadeamento de programas, e diversos Sistemas Gerenciadores de *Workflows* Científicos (SGWfC) foram criados como ferramentas para apoiar a definição e a execução dos *workflows* científicos (DEELMAN *et al*., 2009).

Apesar da variedade de SGWfC existentes, nem todos foram planejados para coordenar a execução paralela de *workflows* científicos em ambientes de PAD. Dentre os sistemas concebidos com essa característica, a pesquisa descrita nesta dissertação centrou-se no Chiron (OGASAWARA *et al*., 2013), que foi desenvolvido como um motor de execução paralela para *workflows* científicos especificados segundo uma abordagem algébrica (OGASAWARA *et al*., 2011). Essa álgebra adiciona semântica às atividades através de operações algébricas, define um modelo de execução paralelo para as atividades, e utiliza transformações algébricas para otimizar o plano de execução dos *workflows*. O Chiron, além de implementar os conceitos dessa álgebra, possui coleta de proveniência nativa, que pode ser consultada pelos cientistas em tempo real, durante a execução dos experimentos.

Desde a sua criação, o Chiron já foi utilizado para executar *workflows* de diversas áreas da ciência, que incluem bioinformática (OCAÑA *et al.*, 2011), dinâmica de fluidos (GUERRA *et al.*, 2012), e *Big Data* (DIAS *et al*., 2013). Atualmente, o

SciCumulus (OLIVEIRA *et al*., 2012) utiliza o Chiron como motor de execução de *workflows* científicos em nuvens de computadores.

Durante o desenvolvimento desta dissertação, constatou-se que os SGWfC dão preferência a ambientes de processamento paralelo com disco compartilhado. Embora sistemas como o Chiron, o Pegasus (DEELMAN *et al*., 2007) e o Swift (ZHAO *et al*., 2007) sejam bem sucedidos nesses ambientes, há situações em que a única arquitetura disponível é a de memória distribuída. Esses sistemas até são capazes de operar em ambientes de memória distribuída, mas com funcionalidades limitadas e um desempenho muito aquém daquele obtido em ambientes onde todos os nós têm acesso a um único sistema de arquivos.

A escalabilidade e a heterogeneidade existentes em ambientes de processamento paralelo com memória distribuída fazem desses ambientes uma ótima opção de plataforma para a análise de grandes conjuntos de dados, considerando que a quantidade de nós computacionais pode aumentar conforme as demandas de processamento e armazenamento. É nesse ambiente escalável que ganhou popularidade o Apache Hadoop (2014), um arcabouço em código aberto para processamento paralelo que implementa o modelo Map/Reduce (DEAN & GHEMAWAT, 2008).

Adaptar o Chiron à arquitetura de memória distribuída já bastaria como motivação para uma pesquisa, visto que aumentaria a quantidade de plataformas aptas à execução de experimentos científicos apoiados pela álgebra de descrição de *workflows* científicos. No entanto, esta adaptação abre outras possibilidades, especialmente para os cientistas que já utilizam memória distribuída para conduzir seus experimentos. É comum a crítica de que muitos experimentos são modelados como Map/Reduce apenas para aproveitar a popularidade do Hadoop e a disponibilidade de plataformas já configuradas, mesmo que o problema em si não possa ser eficientemente ou facilmente representado por meio de funções *map* e *reduce* (LIN, 2012). Com o Chiron, o cientista tem acesso a outros operadores, que permitem a especificação de um experimento (*workflow* científico) com maior flexibilidade.

Além da maior versatilidade na descrição dos experimentos, o Chiron também oferece ao cientista a capacidade de monitorar a execução em tempo real, através da coleta de dados de proveniência. O acesso à proveniência, especialmente durante a execução, pode ser considerado uma das características mais relevantes para *workflows*

que manipulam dados em larga escala e demoram muito tempo para serem executados. Enquanto um programa Map/Reduce seria recomeçado do zero após uma falha na fase de *Reduce*, o Chiron executaria novamente apenas a atividade de *Reduce* em um *workflow* equivalente.

Até um programa clássico de Map/Reduce como o de contagem de palavras pode ser beneficiado pela proveniência armazenada pelo Chiron. Com Hadoop, a contagem parcial de palavras em cada documento (produto da fase de *Map*) não é acessível; apenas a contagem total de cada palavra após a conclusão da fase de *Reduce*. Com o Chiron, é possível monitorar a execução e acessar os dados produzidos em qualquer atividade, independentemente do estado das atividades seguintes.

A partir do exposto, é possível formalizar o objetivo desta dissertação como sendo a migração do Chiron para um ambiente de processamento paralelo com memória distribuída. Adicionalmente, houve a preocupação em propor novas funcionalidades e avaliar como elas se integram à arquitetura do Chiron. Por fim, há uma avaliação experimental de como programas Map/Reduce se comportam quando modelados e executados através da versão modificada do Chiron, identificada por ChironSN (Chiron + *"Shared Nothing"*).

As principais contribuições desta dissertação são as seguintes:

- Desenvolvimento de uma versão modificada e funcional do Chiron para ambientes de memória distribuída – ChironSN – , que mantém compatibilidade com a versão atual;
- Obtenção de resultados experimentais que avaliam o ChironSN e os recursos adicionados durante o processo de migração;
- Exposição de funcionalidades ou modificações que podem ser adicionadas ao ChironSN em futuras extensões dessa pesquisa.

O conteúdo dessa dissertação está organizado em outros quatro capítulos além deste capítulo introdutório. O Capítulo 2 contextualiza a execução de *workflows* científicos em ambientes de memória distribuída e apresenta ferramentas que apoiam o processamento paralelo de dados em larga escala nesses ambientes. O Capítulo 3 apresenta o processo de migração do Chiron para ambiente de processamento paralelo com memória distribuída, o que inclui as modificações feitas na arquitetura atual e

como elas contribuem para o desenvolvimento do Chiron como motor de execução paralela de *workflows* científicos. O Capítulo 4 propõe uma avaliação experimental e discorre sobre os resultados obtidos em cenários variados de execução com o ChironSN. O Capítulo 5 conclui essa dissertação e propõe trabalhos futuros.

Capítulo 2 - *Workflows* Científicos e o Processamento Paralelo em Ambientes de Memória Distribuída

Este capítulo contextualiza a execução de *workflows* científicos em ambientes de memória distribuída (seções 2.1 e 2.2). Na sequência (seção 2.3) apresenta-se o Chiron, um motor de execução paralela de *workflows* científicos, elemento central desta dissertação. Para concluir, as seções 2.4, 2.5 e 2.6 apresentam o modelo de programação Map/Reduce e o arcabouço Hadoop, muito importantes para o processamento paralelo em ambientes de memória distribuída.

2.1 Ambientes de Memória Distribuída

Em um ambiente computacional de memória distribuída, cada nó é uma unidade independente e autossuficiente. Qualquer comunicação ou compartilhamento entre os nós é feito apenas através da rede que os conecta. Uma das principais características dessa arquitetura é a escalabilidade, visto que essa independência entre os nós permite que novos nós sejam adicionados conforme a necessidade, aumentando a capacidade de processamento do sistema. Outra característica é a heterogeneidade que pode existir nesses ambientes, com nós de especificações técnicas variadas (HOGAN, 2014).

Não obstante, uma aplicação distribuída só fará bom uso da escalabilidade se os dados a serem consumidos por ela também estiverem distribuídos pelos nós, e de forma eficiente. Esse, inclusive, é um dos principais desafios para quaisquer sistemas que se proponham a gerenciar o processamento paralelo de aplicações em ambientes de memória distribuída. Dentre esses sistemas, temos os Sistemas Gerenciadores de *Workflows* Científicos.

2.2 Sistemas Gerenciadores de *Workflows* Científicos

Workflows científicos podem ser tratados como uma abstração utilizada para modelar o fluxo de atividades e dados em um experimento científico. Tais experimentos podem estar associados a qualquer área da ciência. Um fator comum entre eles é a crescente demanda por poder computacional, reflexo da enorme quantidade de dados que precisa ser processada e analisada. (DEELMAN *et al*., 2009).

Sistemas Gerenciadores de *Workflows* Científicos (SGWfC) são *softwares* que apoiam a criação, a configuração, a execução e o monitoramento de *workflows* científicos. Dentre os vários SGWfC existentes, podemos citar: Kepler (ALTINTAS *et al*., 2004), VisTrails (CALLAHAN *et al.*, 2006), Pegasus (DEELMAN *et al*., 2007) e Swift (ZHAO *et al*., 2007). Apesar de terem um objetivo comum, os SGWfC utilizam formas diferentes de representação para os *workflows* e, muitas vezes, focam em tipos específicos de aplicações e domínios.

Kepler e VisTrails, por exemplo, são relevantes nos quesitos de usabilidade, semântica e proveniência, mas não são capazes de executar *workflows* de forma distribuída. Essa funcionalidade é cada vez mais exigida em um SGWfC. A capacidade de aproveitar os recursos de ambientes de Processamento de Alto Desempenho (PAD) para executar *workflows* científicos é imprescindível quando se está analisando grandes conjuntos de dados. Dentre os SGWfC que oferecem suporte nativo à execução distribuída, Pegasus e Swift merecem destaque.

O Pegasus é um dos SGWfC mais completos no que diz respeito à execução distribuída de *workflows* científicos. Ele possui interface gráfica para definição, submissão e execução de *workflows* científicos. A execução distribuída é transparente para o cientista, que tem à disposição ferramentas para monitorar a execução do *workflow*. Dados de proveniência são coletados, armazenados em bases de dados e podem ser acessados através de consultas ou utilitários fornecidos pelo próprio Pegasus.

O Swift consiste em uma linguagem de programação com paralelismo implícito. Durante a escrita do código que descreve o *workflow*, o cientista utiliza primitivas específicas da linguagem para sinalizar quais partes desse *workflow* devem ser paralelizadas. Todo o processo de distribuição da execução é transparente ao cientista. No entanto, a ausência de interface gráfica para definir os *workflows* e monitorar a sua execução pode dificultar o trabalho de alguns cientistas. O suporte à proveniência também deixa a desejar, pois a única opção é consultar arquivos de log que são gerados durante a execução do *workflow*.

Uma característica comum a Pegasus e Swift é que ambos podem ser executados em ambientes de memória distribuída, mas os dados de entrada devem ficar armazenados em um local único, seja em um dos nós ou em um servidor remoto. Não há distribuição dos dados entre os nós antes da execução do *workflow*; os dados são

copiados para os nós sob demanda, de acordo com o fragmento que será processado. Ao final da execução, os dados produzidos podem ser transferidos para um diretório pré-definido em um dos nós.

Se os dados estão em um servidor remoto, o gargalo da execução pode ser no tempo de transferência dos arquivos. Se os dados estão em um dos nós, o tempo gasto com transferências será menor, mas como essas transferências terão um dos nós como ponto de convergência, um desbalanceamento tende a ocorrer na rede que interconecta os nós, degradando o desempenho, especialmente se outras aplicações também estiverem sendo executadas ao mesmo tempo. Esse cenário ainda está longe de representar uma solução ótima, e há espaço para melhorar a execução de *workflows* científicos em ambientes de memória distribuída.

2.3 Chiron e a abordagem algébrica para *workflows* científicos

O Chiron (OGASAWARA *et al*., 2013) é um motor de execução paralela de *workflows* científicos, projetado para ser utilizado em ambientes de PAD no processamento de grandes conjuntos de dados. Os *workflows* executados pelo Chiron são especificados segundo uma álgebra (seção 2.3.1) cujos operadores regem o consumo e a produção de dados pelas atividades que compõem esses *workflows*. O modelo arquitetural (seção 2.3.2) do Chiron garante o bom aproveitamento de todos os recursos disponíveis no ambiente de PAD. O suporte a consultas de proveniência (seção 2.3.3) em tempo real é um dos diferenciais do Chiron em relação a outros SGWfC.

O Chiron tem suportado a execução de experimentos em diversas áreas da ciência, desde bioinformática (OCAÑA *et al.*, 2011) até dinâmica de fluidos (GUERRA *et al.*, 2012). Há também o SciCumulus (OLIVEIRA, 2012), um ambiente de execução paralela de *workflows* científicos em nuvens de computadores, que utiliza o Chiron como motor de execução dos *workflows*.

2.3.1 A abordagem algébrica para definição de *workflows* científicos

A abordagem algébrica criada para representar *workflows* científicos (OGASAWARA *et al*., 2011) foi baseada no modelo de álgebra relacional. Nessa abordagem, as atividades de um *workflow* são mapeadas para operadores algébricos dotados de semântica quanto ao consumo e à produção de dados. Os dados, por sua vez, são representados uniformemente como relações (de entrada e saída).

A semântica contida nos operadores viabiliza a reescrita de um *workflow* através de transformações algébricas, criando expressões equivalentes que podem tornar o plano de execução desse *workflow* mais eficiente (OGASAWARA, 2011). Essa possiblidade de otimização também é uma das justificativas em se basear a abordagem algébrica no modelo relacional, onde há um conjunto de técnicas já estabelecidas para processamento e otimização de consultas.

Ainda de acordo com a álgebra, cada conjunto de parâmetros de entrada e saída de um *workflow* constitui uma unidade de dados, similar ao que ocorre com as tuplas no modelo relacional. Dessa maneira, pode-se assumir que atividades consomem e produzem relações, e cada uma dessas relações possui atributos e é composta por um conjunto de tuplas. Os tipos dos atributos podem ser primitivos (inteiro, real, texto, etc.) ou complexos (por exemplo, referências a arquivos).

Os operadores incluídos na álgebra estão listados na Tabela 1, acrescidos de detalhes específicos à semântica de cada um deles. Uma atividade deve consumir e produzir tuplas de acordo com o que é previsto pelo operador que a caracteriza. Esse contrato entre atividade e operador é o que garante que eventuais transformações algébricas não produzam um *workflow* cujo resultado seja diferente daquele esperado pela execução do *workflow* original.

Parte dos operadores (*Map*, *SplitMap*, *Reduce*, *Filter*) prevê a execução de um programa externo para consumir a relação de entrada e produzir a relação de saída. Esse programa deve respeitar a estrutura das relações de entrada e saída definidas pela atividade vinculada ao operador. Os operadores restantes, *SRQuery* e *MRQuery*, por outro lado, executam uma expressão de álgebra relacional sobre a relação de entrada (relações, no caso da *MRQuery*), sendo úteis para filtragem e transformação de dados.

Tabela 1 - Operadores algébricos. Adaptado de (OGASAWARA *et al.*, 2011).

Operador	Tipo de execução	Operandos	Razão de consumo/produção de tuplas
Map	Programa	Relação	1:1
SplitMap	Programa	Relação	1:m
Reduce	Programa	Relação, Atributo(s) de agrupamento	n:1
Filter	Programa	Relação	1:(0-1)
SRQuery	Expressão de Álgebra Relacional	Relação	n:m
MRQuery	Expressão de Álgebra Relacional	Conjunto de relações	n:m

Uma atividade regida pelo operador *Map* consome individualmente as tuplas de entrada, produzindo uma tupla como resultado a cada iteração. Normalmente é utilizada para processar alguma transformação complexa no conteúdo da tupla de entrada. Quando a tupla de entrada contém referências a arquivos, é comum que o programa externo manipule esses arquivos e inclua na tupla de saída o resultado desse processamento, ou referências a novos arquivos criados.

Uma atividade regida pelo operador *SplitMap* efetua fragmentação nos dados contidos na tupla de entrada, produzindo uma quantidade variável de tuplas. Se a entrada é uma linha de texto, as tuplas de saída podem conter cada uma das palavras que formam aquela linha. Outra opção é a tupla de entrada referenciar um arquivo grande. Nessa situação, as tuplas de saída armazenarão referências para os arquivos menores, gerados a partir da fragmentação do arquivo original.

Em uma atividade regida pelo operador *Reduce*, espera-se a produção de uma tupla para cada conjunto de tuplas de entrada, que são agrupadas através de um valor comum. Uma aplicação usual é o oposto da fragmentação: se as tuplas que contêm referências para os arquivos têm algum atributo com valor igual, elas podem ser agrupadas através desse atributo e o programa externo irá produzir um novo arquivo baseado nas referências de arquivo contidas nas tuplas de entrada.

Uma atividade regida pelo operador *Filter* avalia se cada uma das tuplas de entrada deve ser propagada como tupla de saída. O programa externo implementa a lógica necessária para verificar se o conteúdo da tupla é relevante para as próximas atividades. Uma propriedade desse operador é a exigência de que as relações de entrada e saída tenham a mesma estrutura.

O operador *SRQuery* dispensa a execução de um programa externo. Em atividades regidas por esse operador, uma expressão de álgebra relacional é aplicada à relação de entrada, onde é possível efetuar projeções e transformações nas tuplas que compõem essa relação. O produto dessa expressão é tratado como a relação de saída da atividade.

O operador *MRQuery* é semelhante ao *SRQuery* no que diz respeito à execução de uma expressão de álgebra relacional sobre os dados de entrada de uma atividade. A diferença está no fato desse operador aceitar múltiplas relações de entrada, o que permite que uma atividade manipule dados de outras atividades já executadas. O operador se faz útil, por exemplo, quando é necessário efetuar uma junção em dados produzidos por duas atividades.

Essa abordagem algébrica, além de apresentar operadores algébricos que viabilizam otimizações no plano de execução do *workflow*, introduziu também o conceito de ativação de atividade, que é essencial para o modelo de execução paralela do *workflow*.

Uma ativação de atividade (ou apenas ativação) é uma estrutura autocontida que armazena todas as informações necessárias para a execução de uma atividade de um *workflow* em um processador ou núcleo no ambiente de PAD. Uma ativação, embora possa consumir e produzir uma quantidade variável de tuplas, contém apenas o mínimo de tuplas necessárias para que a execução seja realizada. O conjunto de tuplas de saída de uma atividade é composto por todas as tuplas produzidas pela execução das ativações vinculadas a essa atividade. As ativações de uma atividade devem respeitar a razão entre consumo e produção de tuplas especificada pelo operador que rege a atividade.

A execução de uma ativação compreende três etapas: instrumentação de entrada, invocação de programa e extração de saída. Instrumentação de entrada refere-se à extração dos valores da tupla de entrada e preparação destes para o formato esperado pelo programa que será executado. Invocação de programa é a etapa em que o programa

externo é executado. No momento da invocação do programa, os valores instrumentados são passados como parâmetro, normalmente via linha de comando. Extração de saída refere-se a coletar os dados gerados pelo programa e transformá-los no formato esperado pelas tuplas de saída.

A divisão de uma atividade em ativações permite um melhor aproveitamento de um ambiente computacional com muitos nós/processadores/núcleos. Outro fator que aumenta o alcance da álgebra é não impor restrições ao programa externo que será usado. Este pode ser escrito em qualquer linguagem de programação, podendo até ser *software* legado de projetos anteriores.

O ciclo de vida de uma ativação compreende quatro estados:

- *Enfileirada*: todos os dados de entrada estão prontos, faltando apenas a ativação ser enviada para execução em um nó computacional;
- *Esperando*: a ativação precisa esperar que outras ativações terminem de executar, para então mudar para o estado Enfileirada;
- *Executando*: a ativação já foi enviada para execução;
- *Executada*: a ativação já foi executada, e os dados de saída já foram extraídos.

Quando duas atividades têm relação de dependência entre si, a atividade dependente precisa que sua relação de entrada tenha esquema semelhante à relação de saída da atividade referenciada.

Para a execução dos *workflows*, foram apresentadas duas estratégias de fluxo de dados, que coordenam diferentemente a execução das ativações. Na estratégia *Primeira-Atividade-Primeiro*, a prioridade é executar todas as ativações de uma atividade, antes de passar para a execução da próxima atividade. Na estratégia *Primeira-Tupla-Primeiro* o objetivo é executar um conjunto de ativações de cada atividade, sempre propagando os dados para um conjunto de ativações da atividade seguinte. Quando um conjunto de ativações da última atividade do *workflow* é executado, o processo recomeça na primeira atividade, com outro conjunto de ativações. Essa estratégia depende muito da linearidade do *workflow*, ou da ausência de atividades bloqueantes, que precisam de todos os dados para poder operar.

A execução de um *workflow* conta ainda com duas opções para estratégias de despacho. No despacho estático, múltiplas ativações são enviadas para cada uma das unidades de processamento. No despacho dinâmico, as ativações são enviadas uma a uma às unidades de processamento (sob demanda).

2.3.2 Arquitetura do Chiron

O Chiron foi o motor de execução desenvolvido para coordenar a execução distribuída em ambientes PAD de *workflows* especificados segundo a abordagem algébrica proposta por OGASAWARA *et al*. (2011). O Chiron é uma aplicação escrita em Java, e utiliza MPI (GEIST *et al.*, 1996) para troca de mensagens entre as suas instâncias que operam nos nós computacionais.

De acordo com a especificação do MPI, a cada instância deve ser atribuído um valor único (do inglês, *rank*) que a identificará nas trocas de mensagens. Assumindo uma configuração com *k* nós computacionais, e uma instância do Chiron sendo executada em cada nó, temos uma arquitetura como a ilustrada pela Figura 1.

Todos os nós computacionais compartilham a mesma área de armazenamento (disco compartilhado), o que é um dos requisitos para a atual versão do Chiron. Para a gerência das atividades, controle da execução das ativações e coleta de proveniência, o Chiron utiliza uma base de dados relacional. Todas as instâncias podem executar ativações, mas apenas a instância cujo *rank* é igual a 0 (instância coordenadora) manipula a base de dados. Essa base de dados não precisa ser mantida na mesma infraestrutura computacional onde o Chiron é executado, mas é importante minimizar o tempo de acesso aos dados ali armazenados. No cenário mais comum, a base de dados de proveniência é executada no mesmo nó que a instância coordenadora do Chiron.

Além do ambiente de execução do Java, a execução do Chiron requer a instalação do banco de dados PostgreSQL (2014). O Chiron utiliza ainda duas bibliotecas, que já estão incluídas no arquivo executável do Chiron. Uma delas é o PostgreSQL JDBC *driver* (2014) para interação com o banco de dados. A outra biblioteca é a MPJ (CARPENTER *et al.*, 2000), uma implementação em Java da interface MPI.

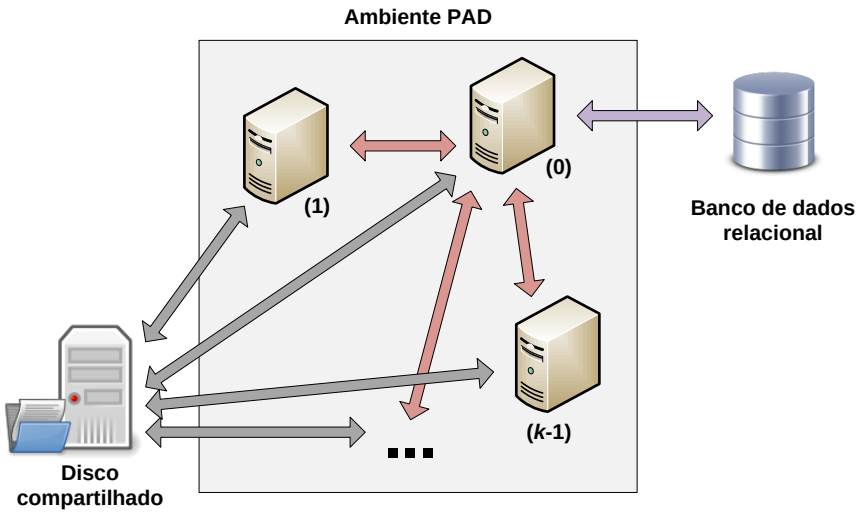

Figura 1 - Arquitetura do Chiron. Adaptado de (OGASAWARA, 2011).

Aproveitando-se do fato de que a quantidade de núcleos por processador está cada vez maior, o Chiron estabeleceu que cada instância pode conter múltiplas linhas de execução (do inglês, *threads)* responsáveis por processar ativações. Cada instância tem uma linha de execução responsável por distribuir ativações, e *n* linhas de execução aptas a processar ativações. Apenas as linhas de execução de distribuição (uma por instância) se comunicam com a instância coordenadora, enviando ativações concluídas e solicitando novas ativações.

A instância coordenadora, além de contar com as mesmas linhas de execução das outras instâncias, contém também uma linha de execução responsável por responder às requisições de ativações e coletar a proveniência. Essa linha de execução só gera mensagens de resposta às solicitações das linhas de execução de distribuição; nunca envia mensagens sem que tenha havido uma solicitação.

Por outro lado, as linhas de execução de distribuição operam sob o modelo de consulta periódica (do inglês, *polling)* (TANENBAUM, 2007), sempre enviando requisições quando não há ativações sendo processadas. Quando não há ativações prontas para execução, a mensagem recebida é de espera, e a linha de execução aguarda um intervalo de tempo predefinido antes de enviar outra requisição. Quando o *workflow* termina de executar, a mensagem recebida é de conclusão, o que leva a linha de execução de distribuição a encerrar aquela instância do Chiron.

A Figura 2 contém uma representação da arquitetura do Chiron do ponto de vista das linhas de execução. A imagem assume uma instância em cada um dos *k* nós e *n* linhas de execução que processam ativações por instância.

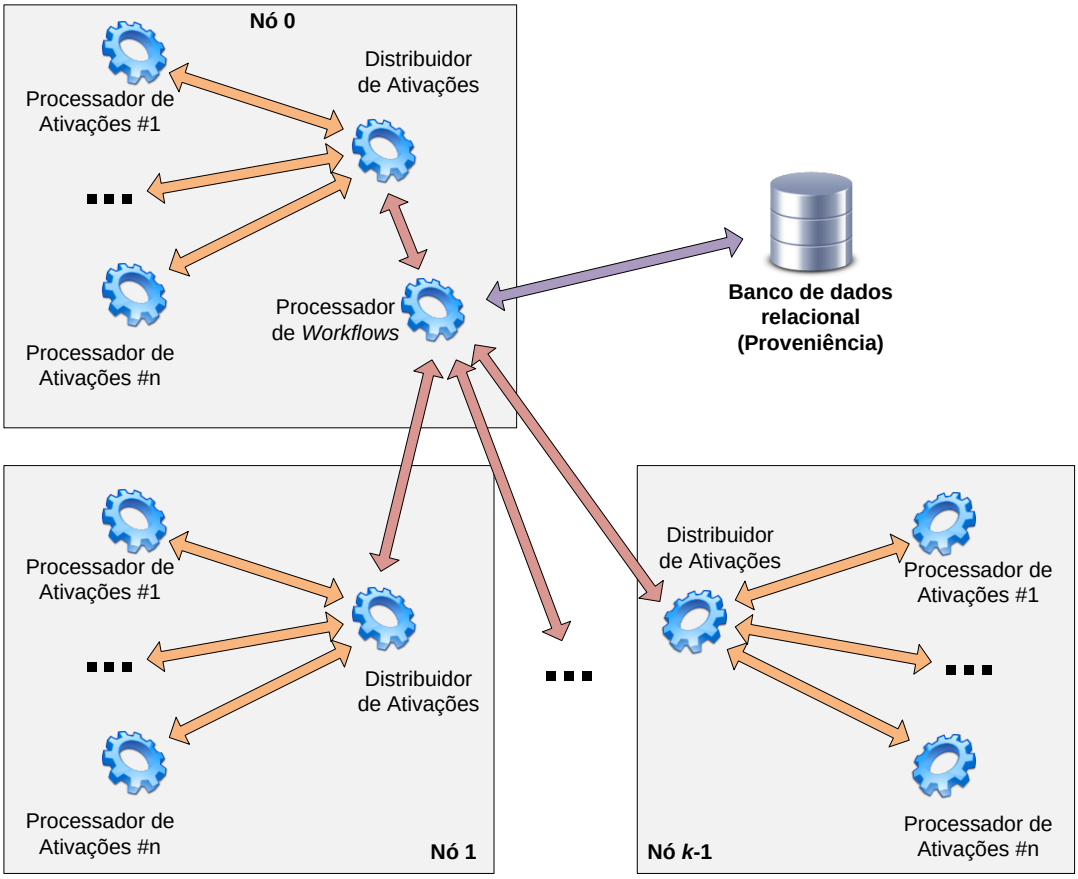

Figura 2 - Arquitetura do Chiron - Linhas de execução. Adaptado de (OGASAWARA, 2011).

Um *workflow* executado pelo Chiron é especificado através de um arquivo XML. A utilização desse formato permite que ferramentas de composição de *workflow* sejam utilizadas, como a GExpLine (OLIVEIRA *et al.*, 2010). Esse arquivo XML contém a descrição conceitual do *workflow*, que inclui, dentre outras coisas:

- *Tag* que identificará o *workflow* na base de proveniência;
- Comando que será executado pelas ativações (invocação de programa ou expressão de álgebra relacional);
- A definição das atividades, incluindo o operador algébrico relacionado;
- A definição das relações de entrada e saída de cada atividade;
- A definição dos atributos de cada relação.

A execução de um *workflow* necessita de outro arquivo XML, que descreve o experimento. Esse arquivo contém os parâmetros específicos para aquela execução. Dentre os parâmetros, destacam-se:

- *Tag* que indica qual *workflow* será executado;
- *Tag* que identifica a execução (deve ser única para um mesmo *workflow*);
- Diretório onde os arquivos de entrada e os arquivos de saída serão armazenados;
- Parâmetros opcionais para atividades e relações.

O portal[1] do Chiron na *web* disponibiliza vários exemplos de *workflows* que podem ser consultados, baixados e executados. Os arquivos XML de descrição conceitual e experimental utilizados nesta dissertação estão disponíveis no Apêndice A.

2.3.3 Coleta de proveniência

O Chiron possui coleta nativa de proveniência, que é utilizada para armazenar a descrição conceitual dos *workflows* (extraída do XML) e, principalmente, gerenciar a execução das ativações. Como a base de proveniência é atualizada em tempo real com informações sobre os estados das ativações (em espera, em execução, etc.) e com os resultados produzidos por elas, os cientistas podem realizar consultas à base e monitorar a execução do experimento. É possível analisar o tempo de execução das ativações, quais ainda estão em espera ou enfileiradas, quais atividades já foram completamente executadas, etc.

A proveniência viabiliza a criação de pontos de checagem, o que permite que um experimento não recomece a execução do zero, em caso de falha. É possível detectar quais dados foram consumidos e quais atividades foram processadas corretamente antes da falha ocorrer. Uma segunda execução do mesmo experimento continuaria de onde a execução anterior parou. Outra opção é que o próprio cientista interrompa a execução ao detectar alguma inconsistência em algum resultado intermediário produzido. Com os dados de proveniência, o cientista pode corrigir o que for necessário no *workflow* e reiniciar a execução em algum ponto de checagem anterior, agora com as correções já

[1] Chiron - http://chironengine.sourceforge.net/

efetuadas. A coleta de proveniência é relevante à medida que oferece ao cientista a capacidade de conduzir e monitorar a execução de *workflows* científicos (GIL *et al.*, 2007).

MATTOSO *et al.* (2014) apresenta três aplicações reais de domínios diferentes que utilizam o Chiron e exploram os dados coletados pela proveniência para avaliar a configuração de parâmetros e dados de entrada, visualizar resultados parciais ou até otimizar/corrigir a execução de alguma atividade que esteja demorando demais para finalizar.

Todas as tuplas consumidas e produzidas ficam armazenadas na base de proveniência, que é centralizada. Devido a isso, é importante que as tuplas contenham apenas dados primitivos e referências a arquivos. Os arquivos reais ficam salvos no disco compartilhado. Esse comportamento é muito relevante quando o *workflow* está manipulando um grande volume de dados, pois minimiza o tempo de execução. Caso o cientista opte por não utilizar referências a arquivos, o armazenamento de todos os dados na base de proveniência pode implicar em maiores tempos de execução para o experimento.

2.3.4 O fluxo de execução do Chiron

A execução de um *workflow* pelo Chiron segue uma rotina simples: se as dependências de uma atividade estão prontas, esta atividade é submetida para execução. Essas dependências podem ser: um arquivo de texto fornecido pelo cientista, os dados produzidos por outra atividade, ou uma mescla de ambos. A eficiência e o diferencial do Chiron estão na forma como o consumo dos dados de entrada de uma atividade é fragmentado de modo que seja viável a paralelização da execução. Para cada atividade, o Chiron efetua as seguintes tarefas:

1) Importação dos dados de entrada: os dados que serão processados pela atividade são salvos na base de proveniência como a relação de entrada da atividade. Esses dados podem ter origem em um arquivo de texto ou serem copiados da relação de saída de outra atividade;

2) Criação das ativações: com os dados de entrada já salvos na base de dados, cada uma das tuplas da relação de entrada é vinculada a uma ativação. Essas ativações são as unidades de execução que serão enviadas aos nós trabalhadores;

3) Execução das ativações: as ativações são enviadas para execução nos nós trabalhadores. Quando todas as ativações de uma atividade são executadas, a atividade é marcada como concluída.

Como o Chiron foi desenvolvido para o processamento de grandes conjuntos de dados, a utilização de referências a arquivos nas tuplas de entrada e saída é de grande valia, pois evita que todos os dados consumidos e produzidos sejam armazenados na base de proveniência, que é centralizada. A Figura 3 ilustra a diferença entre utilizar (a) um arquivo único com todos os dados e (b) um arquivo menor que contenha apenas referências a arquivos menores.

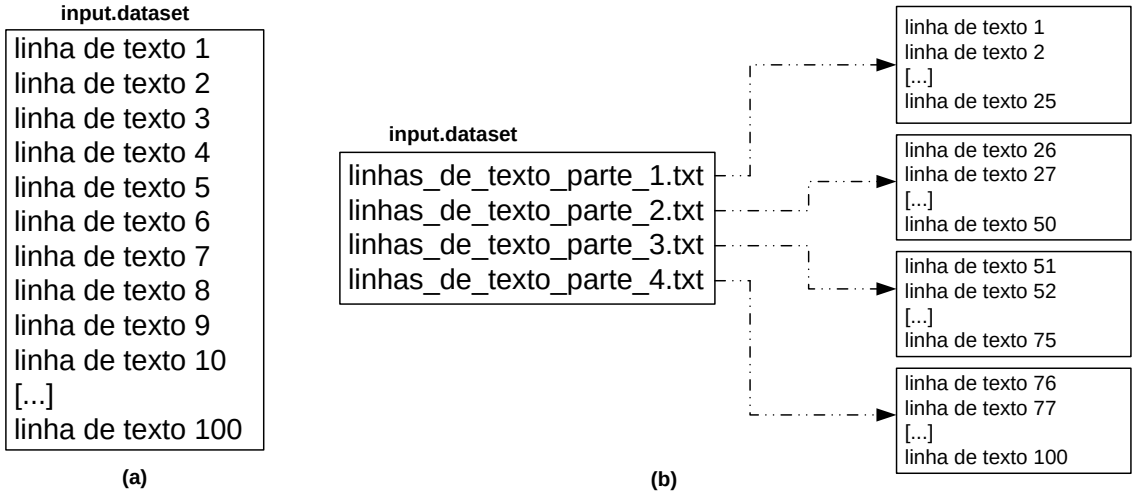

Figura 3 - Dados de entrada de uma atividade: (a) arquivo muito grande com todos os dados e (b) arquivo pequeno com referências a arquivos grandes

Para cada atividade, a quantidade de ativações é igual à quantidade de tuplas da relação de entrada, e essa situação interfere no planejamento do experimento de algumas maneiras:

- Menos ativações que instâncias executoras do Chiron disponíveis: pode ser mais vantajoso aumentar a fragmentação dos dados de entrada a fim de se obter mais ativações e assim aproveitar melhor os nós computacionais disponíveis, minimizando a ociosidade;
- Tempo de execução das ativações: a execução de uma ativação engloba a transmissão da ativação, a invocação do programa externo (uma chamada de linha de comando) e a extração das tuplas produzidas. É de se esperar que o tempo gasto com essas ações seja menor que o tempo gasto pelo programa

externo para processar a tupla. Pode ser considerado um desperdício gastar dois segundos com aquelas ações quando o processamento da tupla em si demora apenas 50 milissegundos.

Ainda sobre a Figura 3, é visível que a abordagem com um único arquivo geraria cem ativações, e o tempo de execução de cada uma dessas ativações seria bem menor que o tempo de execução de cada ativação da outra abordagem, onde cada uma das quatro ativações processaria 25 tuplas (extraídas do arquivo referenciado). Cabe ao cientista ponderar sobre as quantidades de fragmentos e tuplas por fragmento que serão utilizadas na execução do experimento.

A Figura 4 exibe o fluxo de dados durante a preparação para a execução de uma atividade onde os dados são importados para a base de proveniência a partir de um arquivo armazenado no disco compartilhado. Após a importação, são criadas as ativações que serão distribuídas para processar o conteúdo das tuplas.

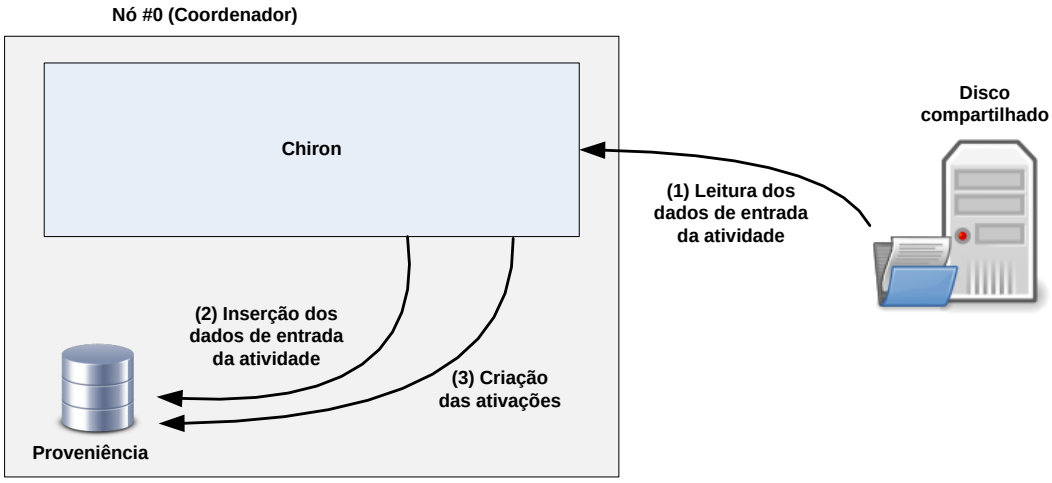

Figura 4 - Inserção dos dados de entrada de uma atividade e criação das ativações

No momento de execução das ativações de uma atividade, o nó coordenador do Chiron atua como agente passivo, apenas respondendo às requisições dos nós trabalhadores. Caso essas requisições já contenham ativações executadas, estas são salvas na base de dados antes que ativações não executadas sejam enviadas na mensagem de resposta. Normalmente apenas uma ativação é enviada por mensagem (despacho dinâmico), mas é possível também o envio de múltiplas ativações por mensagem (despacho estático).

A Figura 5 apresenta todas as etapas envolvidas na execução de uma única ativação, desde a solicitação até o armazenamento das tuplas produzidas. O nó coordenador recebe uma requisição de um nó trabalhador, recupera uma ativação pronta para execução do banco de dados e a envia como resposta. O nó trabalhador recebe a ativação, procede com a instrumentação dos dados da tupla de entrada e invoca o programa externo, passando como parâmetro os dados da tupla e o diretório onde os dados produzidos devem ser armazenados.

Se a tupla contém referências a arquivos, o Chiron já terá copiado tais arquivos para o diretório de execução do programa, permitindo que este acesse os arquivos referenciados sem manipular outros diretórios. As tuplas produzidas também devem ser salvas nesse diretório. Ao final da execução, o programa é encerrado e a instância trabalhadora do Chiron efetua a extração das tuplas produzidas através da leitura do arquivo criado em disco pelo programa. Quando todos os dados foram extraídos, eles são incluídos na ativação, que é marcada como concluída e enviada de volta para o nó coordenador. O nó coordenador irá então atualizar o estado daquela ativação na base de proveniência, incluindo dados como o tempo de execução e um código que identifica o nó que executou aquela ativação.

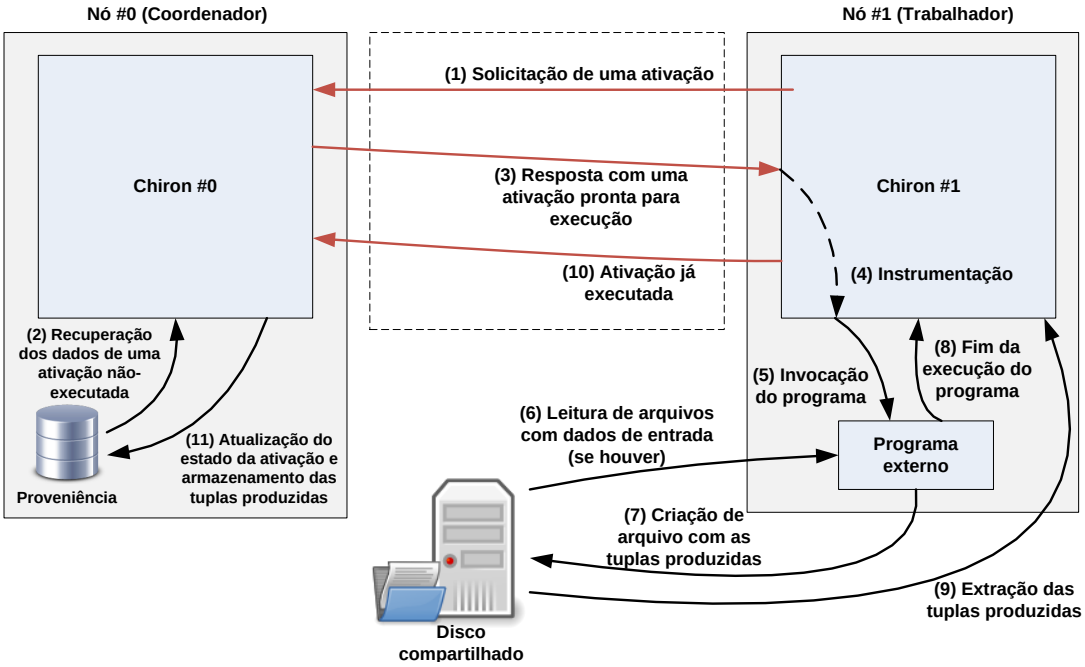

Figura 5 - Fluxo de execução de uma ativação pelo Chiron

Um detalhe sobre a arquitetura do Chiron é que é possível utilizá-lo sem um disco compartilhado caso as tuplas de entrada não contenham referências a arquivos,

mas os dados em si. Como as tuplas consumidas e produzidas são transmitidas via mensagens MPI entre os nós e cada ativação manipula um diretório único, não há a necessidade de um armazenamento compartilhado entre os nós trabalhadores. Quando se utiliza referências a arquivos, o disco compartilhado é mandatório.

2.4 Map/Reduce

Map/Reduce é um modelo de programação cujo objetivo é viabilizar a execução paralela de tarefas, que podem ser utilizadas, por exemplo, para o processamento de conjuntos de dados em larga escala. O arcabouço proprietário Google MapReduce (DEAN & GHEMAWAT, 2008) implementa esse modelo e inclui características como paralelização automática e tolerância a falhas. Programas desenvolvidos segundo esse modelo podem ser executados em variados ambientes computacionais de alto desempenho, tais como *clusters*, grades e nuvens computacionais.

Um programa Map/Reduce é composto por uma função de mapeamento (*Map*) e uma função de redução (*Reduce*), que são definidas pelo usuário. A função de mapeamento opera sobre o conjunto de dados de entrada e, para cada valor de entrada, produz pares no formato chave e valor. Esse conjunto intermediário de pares é coletado e é feito um agrupamento dos valores de acordo com as chaves iguais. O conjunto de chaves e seus conjuntos de valores associados são passados então para a função de redução, que itera sobre os valores e produzem uma nova lista de valores, que é o resultado do programa Map/Reduce.

O pseudocódigo abaixo ilustra a semântica das funções de mapeamento e redução. É comum que a função de mapeamento altere o domínio dos valores de entrada. A função de redução, por outro lado, tende a gerar valores de saída no mesmo domínio dos valores intermediários.

Map(chave_entrada, valor_entrada) → lista(chave_saída, valor_intermediário)

Reduce(chave_saída, lista(valor_intermediário)) → lista(valor_saída)

A execução das funções de mapeamento e redução é de responsabilidade do arcabouço que implementa o Map/Reduce, tornando transparente para o usuário todo o processo de paralelização da execução do programa. A implementação original do Map/Reduce é proprietária e foi desenvolvida pela empresa Google. Felizmente, há à

disposição de todos o Apache Hadoop (2014), um arcabouço que contém uma implementação em código aberto do modelo Map/Reduce.

A arquitetura de um arcabouço Map/Reduce abrange um conjunto de nós computacionais, normalmente um *cluster*, onde um desses nós é utilizado como coordenador, enquanto os outros agem como trabalhadores. Há ainda um sistema de arquivos distribuído, originalmente o Google *File System* – GFS (GHEMAWAT *et al.*, 2003), onde ficam armazenados os arquivos de entrada e saída dos programas Map/Reduce.

O nó coordenador é responsável por disparar e monitorar a execução das tarefas independentes de mapeamento e redução nos nós trabalhadores, sendo capaz de detectar falhas e realocar tarefas para outros nós que estejam ociosos ou que já tenham terminado com sucesso suas tarefas.

Em uma configuração tradicional do *cluster*, cada nó trabalhador é também parte do sistema de arquivos distribuído, o que permite que o nó coordenador aloque tarefas de mapeamento para operar sobre blocos de dados de entrada que estejam armazenados naquele mesmo nó, minimizando custos de transmissão através da rede. Os resultados intermediários gerados pelas tarefas de mapeamento são armazenados localmente nos nós em que elas foram executadas, para depois serem enviados para os nós que executarão as tarefas de redução.

A Figura 6 contém um esquema da execução de um programa Map/Reduce, que inclui a submissão do programa, a alocação dos nós que executarão as tarefas de mapeamento e redução, e o fluxo dos dados processados.

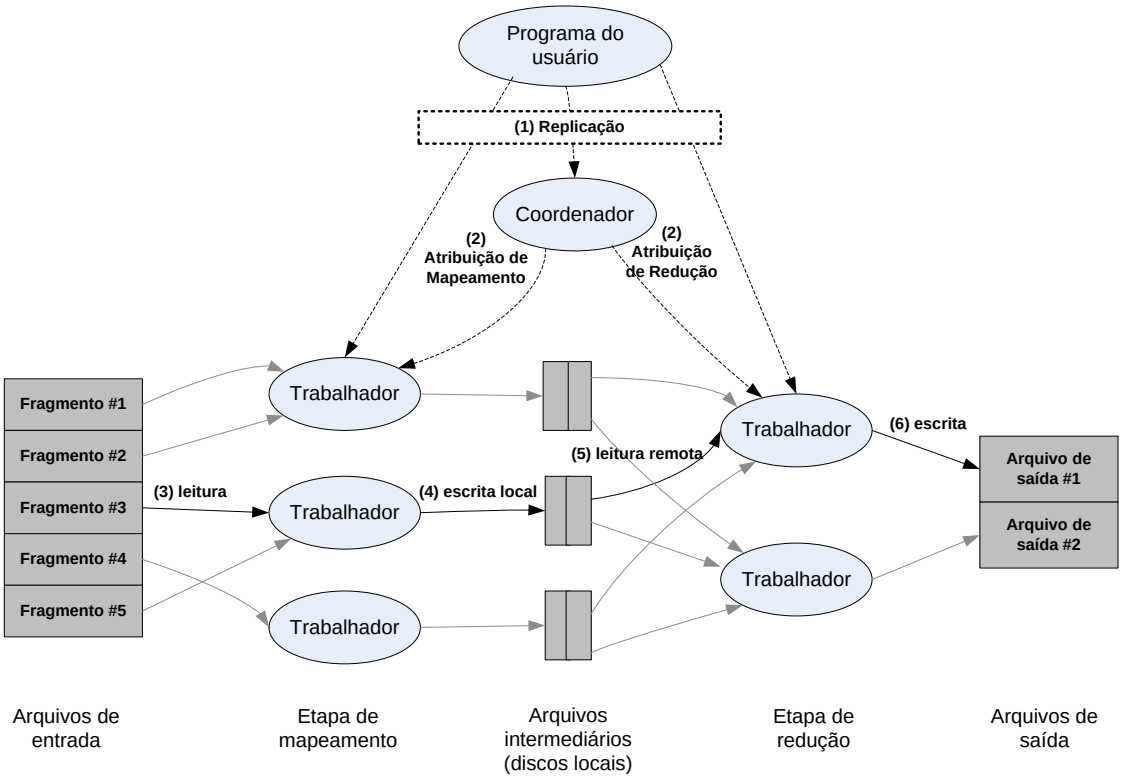

Figura 6 - Execução de um programa Map/Reduce. Adaptado de (DEAN & GHEMAWAT, 2008).

Além das etapas de mapeamento e redução, a execução de um programa Map/Reduce compreende outras etapas que também podem ser definidas pelo usuário, embora a configuração padrão delas já seja suficiente para garantir uma execução distribuída eficiente. Cada uma das oito etapas é detalhada no decorrer dessa seção, de acordo com a ordem em que elas são efetuadas.

A primeira etapa consiste na leitura dos dados de entrada. Para cada tarefa de mapeamento é alocado um fragmento do conjunto de dados de entrada. Devido à diversidade no formato desses dados, há um leitor responsável por extrair pares no formato chave e valor, que serão então passados à função de mapeamento definida pelo usuário. Para cada item lido do fragmento de entrada, é efetuada uma chamada à função de mapeamento. O caso mais comum é a leitura de uma linha de texto por iteração, de modo que a quantidade de chamadas à função de mapeamento será igual à quantidade de linhas de texto no fragmento sendo processado.

Na segunda etapa são feitos os mapeamentos. Conforme apresentado anteriormente, cada mapeamento consiste em processar uma série de pares no formato chave e valor, e para cada um desses pares, gerar zero, um ou vários novos pares, também no formato chave e valor. A estrutura dos pares de entrada e dos pares de saída

tende a ser diferente, considerando que a função de mapeamento é usada principalmente para transformar os dados de entrada, seja através de filtragem, fragmentação, ou outra operação de teor semelhante. Se considerarmos um programa de contagem de palavras, cada chamada à função de mapeamento processaria uma linha de texto, e os pares gerados conteriam uma palavra como chave e a contagem de ocorrências dessa palavra naquela linha como valor.

A etapa de fragmentação é aplicada aos dados produzidos pelos mapeamentos. Antes de as tarefas de mapeamento começarem a executar, elas já têm a informação de quantas tarefas de redução serão disparadas. Dessa maneira, conforme a função de mapeamento produz pares chave-valor, uma função de fragmentação é aplicada sobre as chaves a fim de se definir a tarefa de redução que irá processar esses pares intermediários. Conforme o mapeamento ocorre, as partições vão sendo salvas localmente no nó executor. Ao final da tarefa, tais partições são enviadas para o nó que executará a redução sobre aquela partição. A função de fragmentação é igual para todas as tarefas de mapeamento e é importante que ela distribua os dados intermediários de forma uniforme, equilibrando as transmissões através da rede entre os nós e também balanceando o volume de dados que cada tarefa de redução irá processar.

Os dados de entrada para as tarefas de redução têm a garantia de serem ordenados pelas chaves dos pares intermediários. A etapa de ordenação ocorre em dois momentos: sobre as partições criadas ao final de cada mapeamento; e depois que uma tarefa de redução recupera todas as partições que lhe foram alocadas, mas antes de começar a processá-las. Inicialmente, os pares são ordenados por chave conforme as partições vão sendo criadas em cada um dos nós que executam mapeamentos. Ao final de um mapeamento, cada uma das partições salvas localmente já está ordenada por chave e pronta para transmissão aos nós que efetuarão a redução. A segunda rodada de ordenação é feita no nó em que ocorrerá a redução, conforme são coletadas as partições oriundas das tarefas de mapeamento, e que estão espalhadas pelo *cluster*. As chamadas à função de redução só começam quando todas as partições constituem um único fragmento, com as chaves devidamente ordenadas. Quanto maior o volume de dados intermediários, maior tende a ser o tempo para ordená-los, o que representa um dos principais gargalos na execução de um programa Map/Reduce.

Dependendo da função de mapeamento, muitos dados intermediários podem ser gerados em cada nó, aumentando o volume das partições a serem enviadas para os redutores apropriados. Uma etapa de combinação pode ser aplicada para agregar valores com chaves iguais enquanto a partição ainda é criada, durante a execução do mapeamento. Um detalhe importante sobre a função de combinação é que ela não pode alterar a estrutura dos dados que são produzidos pelo mapeamento, uma vez que não há a garantia de que a combinação será aplicada sobre todos os dados produzidos. Essa etapa serve apenas para diminuir o volume de dados que será transmitido para Redução. A quantidade de vezes que a combinação é aplicada sobre os dados intermediários é determinada em tempo de execução por cada tarefa de mapeamento, e pode ser igual a 0 (zero).

A etapa de embaralhamento engloba o processo de transmissão das partições criadas pelas tarefas de mapeamento para os nós que executarão as tarefas de redução. No pior caso, todos os mapeamentos geram N partições, onde N é a quantidade de tarefas de redução alocadas. Nesse cenário, cada um dos nós de redução receberá uma partição de cada um dos nós de mapeamento. O tempo gasto com todas as transmissões entre os nós também representa um gargalo na execução de programa Map/Reduce. Para minimizar essa espera, conforme as tarefas de mapeamento terminam, suas partições já são encaminhadas para os nós designados para redução.

A etapa de redução consiste em coletar todas as partições que foram designadas àquela tarefa, montar um único fragmento já com as chaves ordenadas, e executar a função de redução para os dados contidos no fragmento. Cada chamada a essa função oferece acesso a uma chave e a todos os valores associados a ela. Os dados produzidos nessa etapa são tratados como o produto da execução do programa Map/Reduce.

A última etapa da execução de um programa Map/Reduce envolve a persistência dos dados produzidos pelas tarefas de redução em um armazenamento de destino, normalmente o sistema de arquivos distribuído. Nesse caso, é comum a criação de 1 (um) arquivo por tarefa de redução, numerados sequencialmente.

A Figura 7 representa o fluxo dos dados durante a execução de um programa Map/Reduce com três tarefas de mapeamento e duas tarefas de redução. O mais relevante nessa imagem são as etapas de fragmentação e ordenação dos dados

intermediários, e como a etapa de embaralhamento pode congestionar a rede que interconecta os nós.

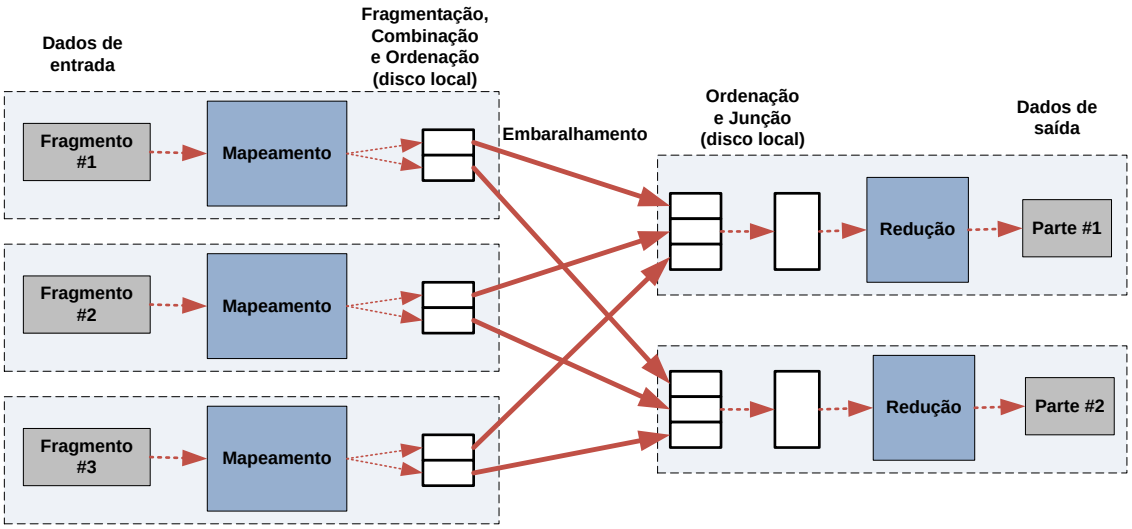

Figura 7 - Fluxo de dados e as etapas de execução de um programa Map/Reduce. Adaptado de (WHITE, 2012).

Na sequência, há a apresentação do arcabouço Apache Hadoop, que foi peça fundamental no desenvolvimento desta dissertação.

2.5 Apache Hadoop

O Apache Hadoop é um arcabouço em código aberto que implementa a arquitetura de execução distribuída proposta pelo modelo Map/Reduce. Para atingir tal objetivo, além de incluir uma implementação de Map/Reduce ("*Hadoop MapReduce*"), há também o HDFS ("*Hadoop Distributed File System*"), uma alternativa gratuita ao sistema de arquivos distribuído apresentado inicialmente pelo Google (GHEMAWAT *et al.*, 2003).

Além de oferecer suporte às características inerentes à arquitetura Map/Reduce, como execução distribuída de tarefas, tolerância a falhas, e balanceamento de carga, o Hadoop adiciona outros recursos, dentre eles:

- Interfaces amigáveis para gerenciamento do HDFS e monitoramento da execução de programas Map/Reduce;
- Interfaces de programação (API), para que as funções de mapeamento e redução sejam escritas em linguagens diferentes do Java, que é a linguagem de programação na qual o arcabouço é desenvolvido;

- Interfaces de entrada e saída, para que os dados consumidos e produzidos por uma aplicação não fiquem restritos ao HDFS. Essas interfaces também viabilizam a manipulação de dados tanto estruturados quanto não estruturados.

Na terminologia utilizada pelo Hadoop, a instância coordenadora, responsável por gerenciar a execução dos programas Map/Reduce, é chamado de *JobTracker*. Cada uma das instâncias trabalhadoras, onde as tarefas de Map/Reduce são executadas, é chamada de *TaskTracker*.

Um programa Map/Reduce pode ser submetido através de qualquer nó do *cluster*. No entanto, esse programa é sempre encaminhado ao nó onde o *JobTracker* está sendo executado. O *JobTracker* irá definir a estratégia de execução e coordenar a execução distribuída das tarefas.

A qualidade e a gama de recursos oferecidos pelo Hadoop (e pelo modelo Map/Reduce, por consequência) podem ser mensurados principalmente pelo grande número de empresas e instituições que o utilizam. Há uma lista[2] na página oficial do Hadoop com os nomes dessas organizações, e como cada uma delas faz uso do Hadoop. Uma análise dessa lista permite constatar como varia a quantidade de nós em *clusters* (desde unidades até milhares), e que a aplicação principal é o processamento de conjuntos de dados que comumente extrapolam 1 (um) *terabyte*.

Outro fator que conta pontos a favor do Hadoop é o "ecossistema" que tem se desenvolvido ao seu redor, com vários projetos de diferentes áreas aproveitando a arquitetura distribuída e escalável do Hadoop para construir aplicações de alto nível. A relevância do Map/Reduce para o processamento distribuído de dados em larga escala já foi discutida neste capítulo, faltando apenas apresentar como o sistema de arquivos distribuído do Hadoop contribui para esse cenário. Ademais, os detalhes da arquitetura do HDFS são essenciais ao processo de adaptação do Chiron para ambientes de memória distribuída.

[2] Organizações que utilizam o Hadoop - http://wiki.apache.org/hadoop/PoweredBy

2.6 HDFS - "*Hadoop Distributed File System*"

Assim como outros sistemas de arquivos distribuídos existentes, o HDFS (SHVACHKO *et al.*, 2010) cumpre os requisitos necessários para prover o armazenamento e o acesso a arquivos através de um conjunto de máquinas independentes e interconectadas através de uma rede. Dentre esses requisitos, destacam-se a transparência na manipulação dos arquivos, a escalabilidade, a confiabilidade, e a tolerância a falhas.

No entanto, como o Hadoop é um arcabouço centrado no processamento de grandes conjuntos de dados em *clusters*, a arquitetura do HDFS não foi planejada de forma a representar fielmente um sistema de arquivos convencional (WHITE, 2012). Dentre os objetivos que basearam a especificação do HDFS, é relevante citar:

- A falha de *hardware* deve ser considerada algo muito provável de acontecer e não deve interferir negativamente na execução das aplicações no *cluster*. Detecção de falhas e recuperação automática de tarefas interrompidas é essencial no HDFS;
- Considerando conjuntos de dados em larga escala, é comum que os arquivos armazenados no HDFS ultrapassem a barreira dos *gigabytes*. Como tal, há a necessidade em maximizar a largura de banda para acessar os arquivos, bem como tornar o sistema altamente escalável, de modo que esses arquivos possam ser distribuídos pelo *cluster* e a carga seja balanceada;
- Aplicações consomem e produzem dados em ciclos contínuos e iterativos. Mas esses dados raramente são modificados após serem criados; no pior caso, são apagados. O HDFS se aproveita dessa condição de imutabilidade para simplificar a gerência dos arquivos. Por exemplo, com o HDFS não é possível adicionar texto ao final de um arquivo previamente criado. Também é impossível solicitar o texto em uma posição específica de um arquivo; é necessário ler o arquivo desde o início até que se chegue à posição desejada;
- Quando se tem um grande volume de dados para processar, é mais eficiente mover a aplicação para perto desses dados do que trazer os dados até ela. Essa operação reduz o congestionamento da rede e aumenta a eficiência do sistema como um todo, incluindo outras aplicações que estejam executando

simultaneamente. Em vista disso, o HDFS oferece meios para a aplicação detectar a localização dos dados de que ela necessita, podendo escolher a melhor estratégia para a execução;

- Há facilidade em se portar e escalar o sistema para configurações diversas de computadores, o que favorece a adesão ao HDFS em vários ambientes.

A partir dos tópicos acima, fica mais fácil compreender a motivação por trás dos detalhes da arquitetura do HDFS que são apresentados na próxima seção.

2.6.1 Arquitetura do HDFS

Assim como o Map/Reduce, o HDFS também adota um esquema de execução com um nó coordenador (*NameNode*) e vários nós trabalhadores (*DataNodes*). Inclusive, a configuração padrão do Hadoop prevê que cada um dos nós trabalhadores atue como *DataNode* e execute um *TaskTracker*, permitindo assim que as tarefas de Map/Reduce sejam alocadas para execução em um nó onde os dados possam ser acessados localmente.

O *NameNode* tem como função gerenciar o espaço de nomes (*namespace*) do sistema de arquivos. Ele armazena a hierarquia de diretórios e arquivos do sistema, acompanhados dos respectivos metadados. Todos esses dados são mantidos na memória física (RAM), e uma representação desses dados é salva periodicamente em disco como cópia de segurança.

Os *DataNodes* armazenam e recuperam blocos de arquivos seguindo ordens do *NameNode* ou de uma aplicação cliente. Além disso, um *DataNode* está sempre notificando o *NameNode* acerca do seu estado e do conjunto de blocos de arquivos que ali estão salvos.

O *NameNode* também mantém em memória a localização dos blocos de todos os arquivos que estão armazenados nos *DataNodes*. Entretanto, esse conjunto de localizações não é incluído na cópia de segurança em disco. Quando o *NameNode* precisa ser reiniciado, ele carrega a cópia de segurança com a estrutura do sistema e os metadados; a localização dos blocos dos arquivos no *cluster* é obtida a partir das mensagens enviadas pelos *DataNodes*.

O grande diferencial do HDFS no apoio ao processamento de dados em larga escala é a forma como os blocos de arquivos são gerenciados e distribuídos dentro do *cluster*. Um arquivo armazenado no HDFS é automaticamente dividido em blocos de tamanho fixo (64MB, por padrão), e esses blocos são distribuídos pelos *DataNodes*. Além disso, cada um dos blocos de um arquivo é replicado em diferentes nós (o valor padrão são três cópias).

Essa divisão em blocos é muito bem aproveitada por aplicações Map/Reduce, uma vez que cada tarefa de mapeamento irá atuar apenas sobre um desses blocos. Várias tarefas executando em paralelo no *cluster* são capazes de processar o arquivo original em menor tempo, e com melhor aproveitamento dos recursos. A distribuição dos blocos e a replicação destes facilita também o processo de alocação das tarefas, aumentando as chances de que elas sejam executadas o mais perto possível do *DataNode* onde o bloco está armazenado.

Um parâmetro de configuração do HDFS determina a quantidade de réplicas por bloco, e o *NameNode* está sempre monitorando os relatórios enviados pelos *DataNodes* para garantir que todos os blocos estejam devidamente replicados. Se um *DataNode* para de se reportar, o *NameNode* assume falha e convoca outros *DataNodes* para armazenar as cópias do blocos que se perderam.

A política de replicação adotada pelo HDFS também é capaz de aproveitar especificidades da distribuição física dos nós. Por exemplo, uma configuração do HDFS permite que sejam declarados os armários (do inglês, *racks*) e como os nós do *cluster* são organizados. Nessa situação, é garantido que pelo menos um nó de cada armário conterá uma réplica de cada bloco (GOLDMAN *et al.*, 2012).

A Figura 8 ilustra um cenário em que dois arquivos são armazenados em um *cluster* que contém oito *DataNodes*, distribuídos em dois armários. O fator de replicação utilizado foi 3 (três).

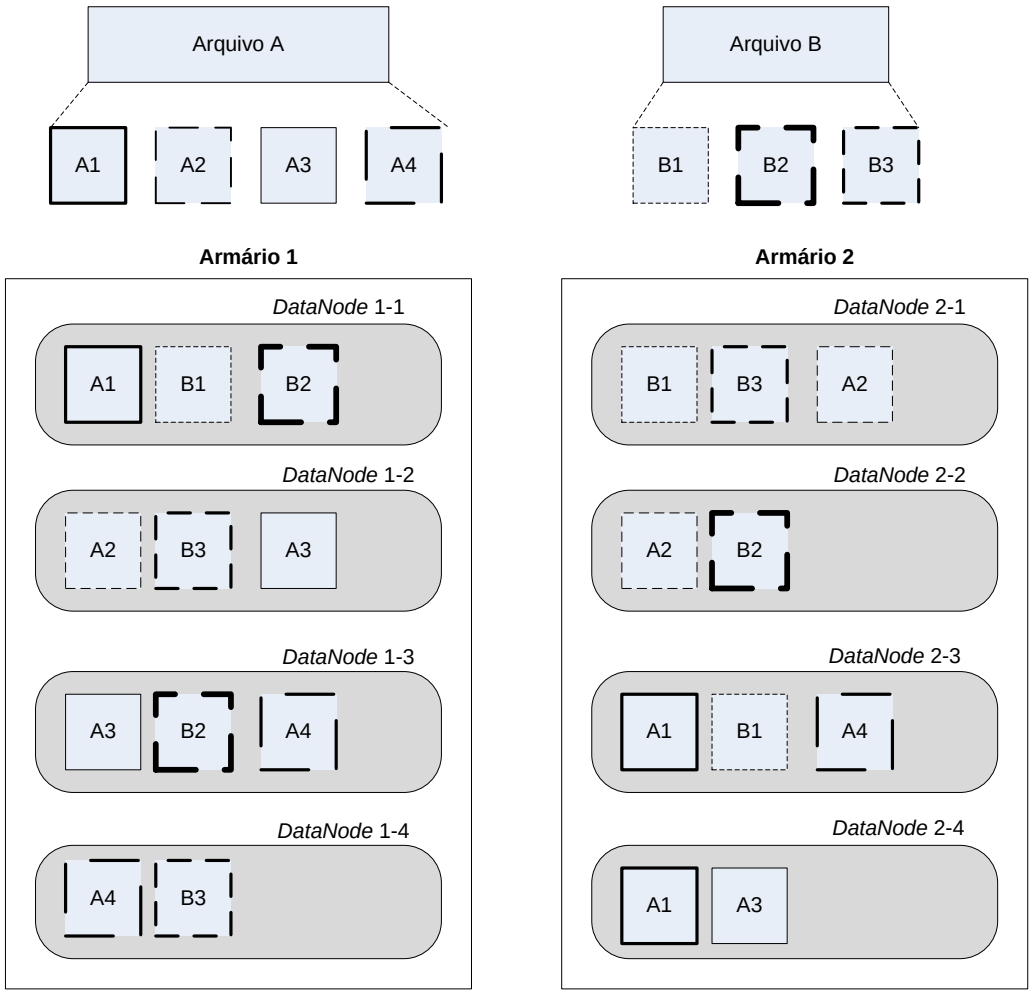

Figura 8 - Replicação de blocos no HDFS. Adaptado de (GOLDMAN *et al.*, 2012).

Além de distribuir os blocos (e suas respectivas réplicas) de cada arquivo pelo *cluster*, o HDFS contém ainda um balanceador, responsável por garantir que essa distribuição seja a mais uniforme possível. A métrica de balanceamento considera a razão entre o espaço ocupado e a capacidade total do disco local de cada *DataNode*. A mesma razão é calculada considerando o uso e a capacidade do *cluster* inteiro. Se a diferença entre os valores de qualquer *DataNode* e do *cluster* é maior que um valor pré-definido (número fracionário entre 0 e 1), não há o balanceamento. Nessa situação, blocos são movidos entre nós para atingir o balanceamento.

O processo de leitura e escrita de arquivos no HDFS também está intimamente ligado ao fator replicação aplicado aos blocos. Numa operação de leitura de arquivo, o *NameNode* recupera uma lista contendo todos os blocos que compõem esse arquivo, e a localização de todas as réplicas desses blocos. As réplicas de cada bloco são ordenadas de acordo com a menor distância até o leitor que enviou a requisição ao *NameNode*. A

leitura do arquivo é feita de forma sequencial e transparente, de tal maneira que o leitor não precisa se preocupar em iterar sobre os blocos que formam o arquivo.

Numa operação de escrita, o *NameNode* armazena os metadados do novo arquivo e um conjunto de *DataNodes* é alocado para armazenar as réplicas do primeiro bloco do arquivo. Cada *byte* escrito é enviado para todos os *DataNodes*, sequencialmente. Se outros blocos são necessários, novos conjuntos de *DataNodes* são alocados, e esse processo continua até que o arquivo seja completamente escrito no HDFS. Ao final da operação de escrita, os blocos já estão replicados e distribuídos pelo *cluster*.

2.6.2 Interação com o HDFS

A arquitetura do HDFS apresentada na seção anterior evidencia como um sistema de arquivos distribuído pode diferir de um sistema de arquivos convencional. No entanto, a interface de comunicação com usuários e aplicações clientes foi planejada para ser o mais semelhante possível com aquilo que já existe e é amplamente utilizado.

O principal meio de gerenciar os arquivos no HDFS é através de um programa de linha de comando que acompanha os executáveis do HDFS. Com esse programa é possível iniciar e parar o *NameNode* ou qualquer um dos *DataNodes*, formatar e apagar todos os dados do *cluster*, obter o estado atual do *cluster*, executar checagem de blocos, dentre outras operações administrativas.

E é claro, também é possível usar a linha de comando para criar, renomear, apagar, copiar e mover diretórios e arquivos. Há outras funções que servem para copiar/mover arquivos do sistema de arquivos local para dentro do HDFS, e vice-versa. Muitos comandos de sistemas UNIX foram replicados para o HDFS, e são utilizados da mesma maneira, com a mesma estrutura. Dentre eles, destacam-se: *cat, chmod, chown, du, ls*. Uma lista completa pode ser encontrada na documentação[3] do HDFS.

Se a intenção é criar uma aplicação que interaja com o HDFS remotamente, ou sem utilizar a linha de comando, a opção ideal é a interface de programação (API) que é disponibilizada para o HDFS. Essa API, também escrita em Java, provê meios de executar, via programas, todas as operações disponíveis via linha de comando.

[3] Documentação online do HDFS - http://hadoop.apache.org/docs/r1.2.1/hdfs_user_guide.html

Porém, essa API não se limita a estabelecer a comunicação com o HDFS. A classe principal (*FileSystem*) foi concebida como uma abstração de um sistema de arquivos, e várias subclasses implementam da maneira apropriada a comunicação com sistemas de arquivos específicos. A classe *DistributedFileSystem* é a que efetivamente gerencia a comunicação com o HDFS.

Outras implementações estão disponíveis nessa API, permitindo que um programa manipule dados de várias fontes de forma fácil, sem a necessidade de utilizar bibliotecas separadas e aprender como cada API funciona. Há inclusive uma classe que encapsula as funções de manipulação de arquivo nativas do Java, o que permite que a classe *FileSystem* seja usada também para o sistema de arquivo local (subclasse *LocalFileSystem*).

Outra possibilidade oferecida pela API decorre da existência de uma implementação (subclasse *S3FileSystem*) para interagir com o serviço de armazenamento de dados na nuvem da Amazon[4], o S3. Uma aplicação pode usar o arcabouço Hadoop para processar dados em um *cluster* e, ao final, extrair os dados gerados e movê-los para o S3, para posterior análise.

Para concluir a exposição das opções de interação com o HDFS, há ainda um servidor *web* que é executado pelo *NameNode* e expõe informações sobre o *cluster*. É possível acessar estatísticas do *cluster* e dos dados armazenados, navegar entre os diretórios, e visualizar os arquivos. Não é possível efetuar tarefas administrativas através dessa visualização.

No próximo capítulo é apresentado o processo de migração do Chiron para ambientes de processamento paralelo com memória distribuída, que inclui a descrição da arquitetura modificada e o detalhamento das funcionalidades adicionadas.

[4] Amazon *Simple Storage Service* (S3) - http://aws.amazon.com/s3/

Capítulo 3 - Migração do Chiron para Ambiente de Processamento Paralelo com Memória Distribuída

Esta dissertação apresenta uma versão modificada do Chiron, doravante identificada por ChironSN (Chiron + *"Shared Nothing"*), que é capaz de operar em ambientes de processamento paralelo com memória distribuída. A abordagem utilizada adapta a arquitetura do Chiron e adiciona o suporte a ambientes computacionais onde os nós computacionais não precisam estar conectados a um mesmo meio físico de armazenamento de arquivos. Cabe destacar que a camada de abstração utilizada para manipular o sistema de arquivos manteve a compatibilidade com discos compartilhados.

O ChironSN inclui todas as funcionalidades pré-existentes no Chiron (retrocompatibilidade), e os novos recursos foram planejados de modo a não afetar negativamente a experiência de uso dos cientistas já acostumados a usá-lo. Parte dos novos recursos, aliás, são invisíveis ao usuário, pois estão relacionados a otimizações no código e aperfeiçoamentos na arquitetura, na busca por um aproveitamento ainda melhor do ambiente computacional disponível. Para o uso do cientista, novos parâmetros de definição e execução dos *workflows* estão disponíveis. A avaliação experimental apresenta cenários em que tais parâmetros são relevantes.

3.1 A integração entre Chiron e HDFS

Conforme exposto anteriormente, o Chiron tem se mostrado uma ótima ferramenta para a condução de experimentos científicos que processam grandes conjuntos de dados. Entretanto, há uma parcela de ambientes com memória distribuída que não incluem um disco compartilhado dedicado entre os nós, caracterizando um cenário onde o Chiron não é totalmente funcional.

Tomando como exemplo o *cluster* onde os experimentos dessa dissertação foram executados: cada nó contém um disco para armazenamento local e os nós são conectados entre si através de uma rede de alto desempenho (20Gb/s). Outra interface de rede (1Gb/s) conecta os nós a um roteador central, onde está conectado também o servidor de arquivos dos usuários do *cluster*. Esse servidor de arquivos pode ser acessado a partir dos nós, mas além das taxas de transferências serem menores, é uma área compartilhada e acessada frequentemente por todos os usuários.

Um experimento que dependesse da leitura e escrita de arquivos nesse servidor central teria desempenho prejudicado, e ainda interferiria no acesso dos outros usuários, além de ser uma prática condenada pelos administradores do *cluster*. A prática recomendada é copiar os dados necessários para os nós no começo da utilização e salvar no servidor os arquivos necessários ao final dos experimentos.

Para fazer uso apenas do armazenamento local em cada nó, o Chiron seria obrigado a distribuir, junto às ativações, os arquivos que essas ativações requerem para executar corretamente. Como a distribuição das ativações é centralizada em uma instância do Chiron, a utilização da rede ficaria desbalanceada entre os nós, além de adicionar pontos de contenção à execução.

A solução escolhida foi a utilização de um sistema de arquivos distribuídos, capaz de armazenar os dados consumidos e produzidos pelas ativações e prover acesso a eles através de uma interface única. No início do experimento, o cientista copia os dados de entrada para esse sistema de arquivos distribuído, e todas as ativações podem manipular os arquivos conforme necessário, sem se preocupar com a localização real desses arquivos. Para o desenvolvimento do ChironSN, foi usado o sistema de arquivos distribuído que faz parte do Apache Hadoop, o HDFS.

A opção pelo HDFS é justificada por três razões:

- O HDFS é um sistema de arquivos distribuído projetado para o armazenamento de conjuntos de dados em larga escala. Sua arquitetura garante balanceamento de carga entre os nós, tolerância a falhas e alta disponibilidade;
- A interface de programação para interagir com o HDFS contém uma camada de abstração que prevê a utilização de outros sistemas de arquivos, garantindo a compatibilidade com a versão atual, que manipula arquivos em disco compartilhado;
- A grande base de usuários do Hadoop e a quantidade de projetos desenvolvidos sobre a plataforma do Hadoop constituem um cenário onde o suporte oferecido ao HDFS tende a ser constante, o que reflete na continuidade das atualizações e dos aperfeiçoamentos da tecnologia.

A primeira etapa no processo de adaptação do Chiron para utilizar o HDFS foi a identificação, durante do ciclo de execução de um *workflow*, de todos os momentos em que o Chiron interage com o sistema de arquivos. Dentre essas interações, destacam-se:

- Leitura dos dados de entrada para uma relação;
- Gerenciamento das pastas onde os arquivos consumidos e produzidos por uma ativação serão armazenados;
- Extração dos dados produzidos pelas ativações.

Antes de fazer uso da API do HDFS, todas as funções do Chiron que manipulavam arquivos foram agrupadas em uma classe, chamada de *ChironFS* (abreviação para "Chiron *File System*"). Dessa maneira, quaisquer alterações na interação com o sistema de arquivos ficam centralizadas nessa classe.

A seguir, as funções da classe *ChironFS* foram modificadas para aproveitar os recursos da API do HDFS. A primeira ação a ser tomada foi criar uma nova função na classe *ChironFS*, que seria responsável por notificar à API do HDFS qual sistema de arquivos ela estaria manipulando: HDFS ou sistema de arquivos convencional (disco compartilhado). Essa notificação é feita no momento em que o *workflow* é submetido para execução, a partir de um parâmetro extraído do XML de descrição do experimento, conforme ilustrado na Figura 9.

Sistema de arquivos convencional

```
<Workflow   tag="sort" execmodel="DYN_FAF" exectag="sort-1"
            wfdir="/home/user/Desktop/sort"
            expdir="/home/user/Desktop/sort/exp" >
    <Relation name="sortIAct1" filename="input-10000.txt" />
</Workflow>
```

HDFS

```
<Workflow   tag="sort" execmodel="DYN_FAF" exectag="sort-1"
            wfdir="/home/user/Desktop/sort"
            expdir="/sort"
            hdfs="localhost:9000">
    <Relation name="sortIAct1" filename="input-10000.txt" />
</Workflow>
```

Figura 9 - Parâmetro de execução necessário para usar o HDFS

O parâmetro *expdir* contém o diretório onde os dados de entrada e saída do *workflow* são armazenados. Quando o parâmetro *hdfs* não está presente no XML, o ChironSN assume que o sistema de arquivos convencional será utilizado. Quando o parâmetro *hdfs* está presente, seu valor deve conter o IP e a porta que estão configurados

para o *NameNode*; e o parâmetro *expdir* será tratado como um diretório relativo à raiz do HDFS.

Todas as outras funções da classe *ChironFS* utilizam o sistema de arquivos escolhido quando o ChironSN solicita alguma operação de manipulação de arquivo. O encapsulamento de funções que são executadas em um mesmo contexto é uma ótima prática de programação e facilita eventuais manutenções no futuro (SCOTT, 2009). Quaisquer modificações serão transparentes para o resto do ChironSN, desde que a funcionalidade permaneça inalterada. A classe *ChironFS* funciona como um componente anexo ao Chiron, servindo de interface para a API do HDFS. A Figura 10 ilustra essa comunicação.

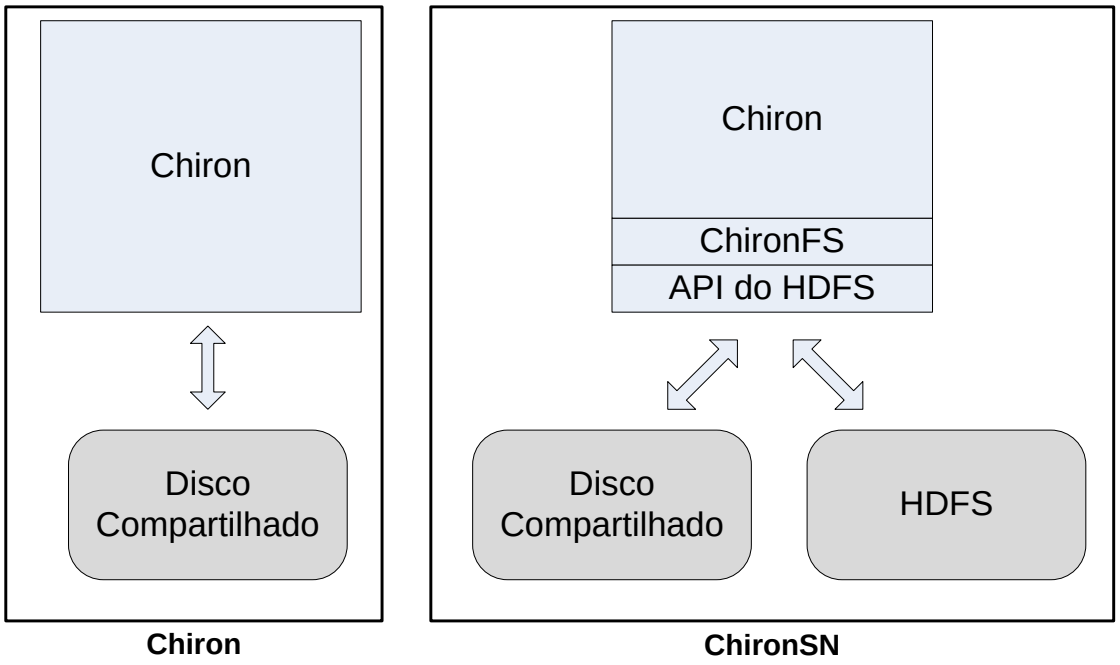

Figura 10 - Comunicação do ChironSN com os sistemas de arquivos

A preocupação em minimizar a quantidade de novos parâmetros necessários para executar um *workflow* no ChironSN está relacionada à proposta de expandir a comunidade de usuários do Chiron, mas sem aumentar a complexidade para aqueles que já estão acostumados com o seu uso.

A Figura 11 contém um esquema ilustrando o ambiente (com *k* nós computacionais) configurado para a utilização conjunta do Chiron e do HDFS. Qualquer instância do ChironSN enxerga o HDFS como um sistema de arquivos convencional (ou compartilhado), embora os arquivos estejam distribuídos pelos discos de

armazenamento local de cada um dos nós. A configuração padrão do HDFS não prevê um *DataNode* em execução no mesmo nó que o *NameNode*. No entanto, como o ChironSN executa ativações em todos os nós, foi feita a opção por manter *DataNodes* ativos em todos os nós.

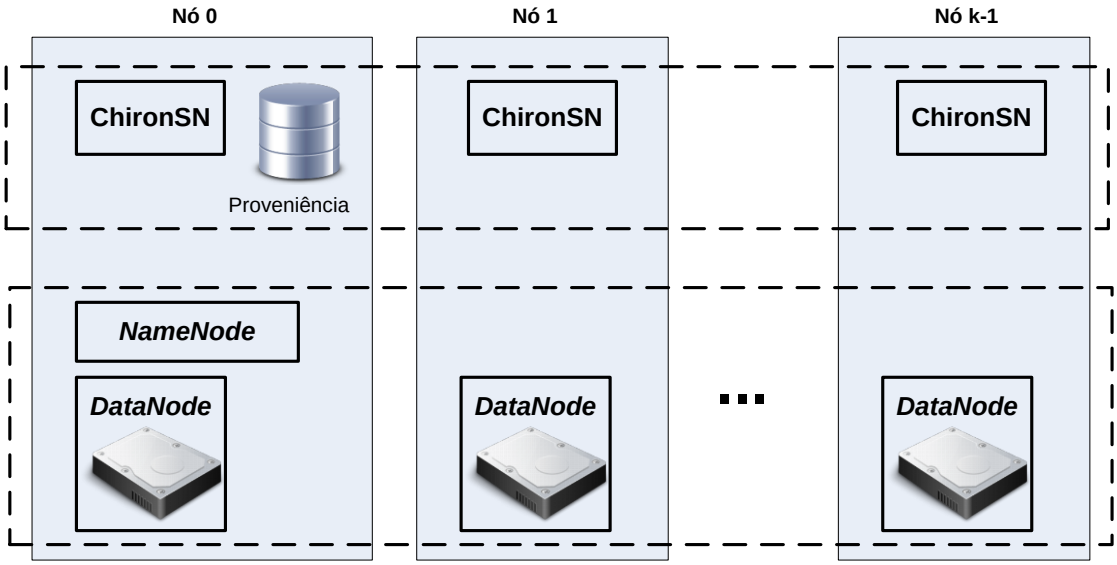

Figura 11 - ChironSN e HDFS integrados no *cluster*

Apesar da migração para um ambiente de memória distribuída, a base de dados de proveniência foi mantida como um ponto de centralização. Como a maior parte dos experimentos executados com o Chiron armazenam apenas referências a arquivos, o volume de dados manipulado pela base de dados tende a não prejudicar o desempenho desses experimentos. De certa maneira, a base de proveniência tem perfil similar ao *NameNode*, que também é um ponto central na arquitetura do HDFS.

Sobre os operadores algébricos disponíveis no Chiron, a integração com o HDFS tem efeito apenas sobre aqueles que envolvem a execução de um programa externo. Os operadores *SRQuery* e *MRQuery* continuam sendo executados diretamente na base de proveniência, dispensando qualquer manipulação no sistema de arquivos.

3.2 Aplicação da localidade dos dados

Em um ambiente de memória distribuída, a utilização de um sistema de arquivos distribuído garante que, mesmo estando fisicamente distribuídos entre os nós computacionais, os arquivos sejam acessados a partir de cada nó como se estivessem em um sistema de arquivos local, do ponto de vista do usuário. Apesar de existir essa

transparência na distribuição dos arquivos, o plano de execução paralela de uma aplicação nesse ambiente deve considerar a localidade desses dados.

O cenário mais comum para uma execução paralela é que cada unidade dessa execução (uma ativação, no ChironSN) processe pelo menos um dos arquivos que compõem o conjunto de dados de entrada. Aplicando-se a localidade dos dados, cada uma das unidades de execução é enviada para o nó onde o arquivo a ser processado está fisicamente localizado (leitura local), de modo que sejam evitadas leituras remotas. Quando a localidade dos dados não é considerada, há grande chance de que cada um dos múltiplos nós envolvidos na execução paralela precise recuperar seus arquivos de outros nós, gerando muito tráfego na rede que conecta os nós e atrasando a execução das tarefas.

O HDFS implementa o conceito de localidade e sua API oferece funções que permitem recuperar a informação sobre quais nós armazenam cada um dos blocos de cada arquivo. Como um bloco é a unidade mínima de armazenamento no HDFS, a API retorna, para um arquivo qualquer, as localizações de todos os blocos desse arquivo. A quantidade de localizações disponíveis para cada um dos blocos depende do fator de replicação configurado para o HDFS.

No contexto de um experimento executado pelo ChironSN, os dados de entrada disponibilizados pelo cientista, bem como os dados produzidos pelas atividades, ficam armazenados no HDFS e replicados conforme a configuração do sistema de arquivos. Quando uma atividade do *workflow* contém referências a arquivos, é importante que o ChironSN envie as ativações para os nós onde os arquivos referenciados estejam fisicamente armazenados.

O aproveitamento da localidade pelo ChironSN foi implementado no nó coordenador e exigiu modificações em duas etapas do ciclo de execução de uma atividade: na criação das ativações e na construção da mensagem de resposta às requisições de ativações enviadas pelas outras instâncias do ChironSN. É importante realçar que tais modificações apenas têm efeito quando as ativações manipulam pelo menos um arquivo. Se todos os dados a serem processados pela ativação estão contidos na tupla enviada na mensagem MPI, não há necessidade de se preocupar com a questão da localidade.

No momento de criação das ativações, incluiu-se um passo onde é criada uma lista com os nós (*DataNodes*) onde cada uma das ativações deve ser preferencialmente executada, de modo que a localidade dos dados seja mais bem aproveitada. Essa lista é armazenada na base de proveniência para cada ativação, e o processo de criação dessa lista é descrito a seguir:

1) Para cada arquivo referenciado na tupla de entrada de uma ativação, recuperam-se os endereços dos *DataNodes* onde os blocos desse arquivo estão armazenados;
2) Como há replicação, alguns *DataNodes* podem armazenar blocos de diferentes arquivos para uma mesma ativação. Quando isso ocorrer, esses *DataNodes* têm prioridade;
3) Para minimizar as transferências entre os nós, blocos com tamanho maior também têm prioridade. Logo, a maior prioridade é para *DataNodes* que armazenam a maior quantidade e os maiores blocos dos arquivos que serão processados por uma ativação.

A segunda parte da modificação para considerar a localidade consistiu em adaptar o processo de seleção das ativações que são enviadas para execução. Para cada requisição que chega ao nó coordenador, a mensagem de resposta é montada de acordo com o algoritmo:

1) Identifica-se o nó que enviou a requisição, a fim de se determinar qual *DataNode* está em execução naquele nó;
2) Busca-se por uma ativação ainda não executada que tenha aquele *DataNode* na sua lista de *DataNodes* preferenciais. A ativação que contiver aquele *DataNode* com maior prioridade em sua lista é selecionada para execução. Se houver empate, a primeira ativação é selecionada;
3) Caso nenhuma ativação pronta para execução contenha aquele *DataNode* na sua lista de prioridades, a primeira dessas ativações é enviada.

3.3 Modificações complementares

Durante o processo de criação do ChironSN, bem como durante a avaliação experimental, algumas possibilidades de modificações foram detectadas. Algumas delas

estavam ligadas a melhorias no código, com relação ao aproveitamento dos recursos computacionais. Outras modificações foram motivadas pela análise de resultados obtidos na realização de experimentos preliminares.

Essa seção apresenta tais modificações, e como elas se integram ao ChironSN. Um fator comum entre elas é que todas são aplicáveis apenas às atividades cujo operador algébrico prevê a execução de um programa pelas ativações.

3.3.1 Adição de tipos de atributo de relação

A descrição conceitual de um *workflow* científico processado pelo Chiron prevê três opções que o cientista pode utilizar para definir o tipo dos atributos que compõem uma relação: *float* (número), *string* (texto) e *file* (referências a arquivos). Cada um desses tipos é mapeado para uma coluna de uma tabela na base de dados de proveniência, onde o valor do atributo é armazenado.

Para a maioria dos *workflows*, esses tipos são suficientes para representar os dados manipulados. Entretanto, o tipo *string* é mapeado para uma coluna com 250 caracteres de tamanho, o que pode dificultar experimentos que precisem manipular longas cadeias de texto. Para os experimentos desta dissertação, o tipo *text* foi incluído à implementação do ChironSN, que é mapeado para o tipo de mesmo nome na base de dados, permitindo que textos de qualquer extensão sejam utilizados como valores.

Uma situação diferente diz respeito ao tipo *float*, que parece exagerado quando o *workflow* precisa lidar apenas com números naturais ou inteiros. O tipo *float* aceita um parâmetro opcional onde se define a quantidade de casas decimais, mas do ponto de vista computacional, ele não deixa de ser estrutura de dados que ocupa pelo menos o dobro de espaço em memória que um número natural/inteiro. Em vista disso, o tipo *integer* também foi adicionado à implementação do ChironSN.

3.3.2 Quantidade variável de tuplas por ativação

A álgebra que suporta o Chiron não especifica uma quantidade de tuplas que deve ser consumidas em uma ativação. Todavia, a implementação do Chiron considera apenas uma tupla sendo consumida por ativação, balizada pelo comportamento usual onde as ativações manipulam arquivos cujas referências estão contidas naquela tupla transmitida.

Quando as tuplas contêm os dados reais, sem referências a arquivos, o tempo de processar uma tupla em um nó remoto tende a ser menor que os tempos gastos com transmissão da mensagem MPI, inicialização da ativação e extração dos dados produzidos. Utilizar despacho estático e enviar um conjunto de ativações de uma só vez é uma opção, mas se considerarmos ativações de uma mesma atividade, todas elas terão instruções semelhantes, variando apenas a tupla com os dados a serem consumidos.

Sendo assim, a melhor opção é agrupar essas tuplas e enviar dentro de uma única ativação, ao invés de criar uma ativação para cada tupla. Com isso em mente, o parâmetro *numTuplesPerActivation* foi incluído como parâmetro de execução das atividades, sendo aplicado no momento em que as ativações são criadas. A Figura 12 mostra como utilizar o parâmetro no XML de descrição do experimento.

```
<Workflow   tag="sort" execmodel="DYN_FAF" exectag="sort-2"
            wfdir="/home/user/Desktop/sort"
            expdir="/sort" hdfs="localhost:9000">
    <Activity tag="act1_map"    numTuplesPerActivation="1000" />
    <Activity tag="act2_reduce" numTuplesPerActivation="-1" />
     <Relation name="sortIAct1" filename="input-10000.txt" />
</Workflow>
```

Figura 12 - Utilização do parâmetro *numTuplesPerActivation*

Qualquer valor positivo é válido, e será a quantidade de tuplas que o ChironSN alocará para cada ativação. O valor -1 é usado quando se deseja que o ChironSN envie uma ativação para cada uma das linhas de execução aptas a processar ativações, considerando todas as instâncias do ChironSN ativas. Nesse caso, a quantidade de tuplas por ativação será a divisão entre o total de tuplas de entrada e o total dessas linhas de execução. Quando o parâmetro é omitido, utiliza-se o comportamento padrão do Chiron, de uma tupla por ativação.

Em atividades de *Reduce*, vários fragmentos horizontais agrupados pelos respectivos atributos podem ser incluídos em uma mesma ativação. Se o valor do parâmetro *numTuplesPerActivation* é alcançado no meio de um fragmento, as tuplas continuam sendo adicionadas à ativação até que esse fragmento esteja completo. Esse parâmetro é ignorado quando as tuplas de entrada contêm pelo menos uma referência a arquivo.

3.3.3 Criação das Ativações

Nessa etapa do ciclo de execução do Chiron, as tuplas da relação de entrada de uma atividade são alocadas para as ativações que irão processá-las. Essas tuplas são identificadas por um código único e sequencial, gerado no momento de inserção na base de proveniência, e cada ativação armazena os códigos das tuplas alocadas para ela. Duas modificações foram feitas na consulta que itera sobre as tuplas de uma relação e extrai os códigos que serão vinculados às ativações.

A primeira alteração foi no conjunto de atributos que eram incluídos na consulta. Na base de proveniência, uma tupla contém o valor dos atributos e o seu código identificador. A consulta incluía todos esses campos, sendo que o único necessário era o código, pois nessa etapa o importante é a quantidade, e não o conteúdo das tuplas. Na nova versão da consulta, apenas o código é considerado na projeção. Um caso particular é a consulta das tuplas para uma atividade de *Reduce*, onde o valor do atributo de agrupamento também é incluído na projeção.

A segunda alteração consistiu em alterar o modo de execução da consulta, a fim de evitar que todas as tuplas fossem recuperadas do banco de dados e armazenadas em memória, de uma só vez. As tuplas agora são iteradas através de um cursor que recupera os dados do banco em blocos. Conforme as tuplas de um bloco são iteradas, um novo bloco é recuperado do banco. Essa medida reduz as chances de falhas por insuficiência de memória RAM.

3.3.4 Transmissão das tuplas para as ativações

A utilização de uma quantidade variável de tuplas por ativação (*numTuplesPerActivation*) originou um problema relacionado ao tempo de resposta da linha de execução que processa as solicitações de ativações. As mensagens MPI recebidas são tratadas sequencialmente, e é importante que essas mensagens sejam respondidas o mais rápido possível com ativações prontas para execução. Uma fila de mensagens não respondidas representa ociosidade nos nós que contêm instâncias executoras do ChironSN.

Quando a quantidade de tuplas por ativação aumentou, o tempo para responder a mensagem também aumentou, visto que mais dados passaram a ser recuperados do banco e inseridos na ativação que seria enviada na mensagem MPI. Há de se destacar

que uma mensagem de solicitação recebida normalmente contém uma ativação já executada, com dados produzidos que devem ser salvos na base de dados.

Para contornar esse problema e manter baixo o tempo de resposta às solicitações, optou-se por criar uma etapa anterior à execução das ativações, onde são escritos em disco arquivos que contêm as tuplas que cada ativação irá processar. Cada um desses arquivos é salvo já nos respectivos diretórios de execução de cada ativação. Essa etapa é executada independentemente do sistema de arquivos escolhido para o experimento. Quando o HDFS está sendo utilizado, a localização desses arquivos criados também é considerada no momento em que o ChironSN seleciona as ativações para execução.

Outra modificação feita para minimizar os tempos de transmissão de mensagens foi mover para a instância coordenadora a extração dos dados produzidos pelas ativações. Ao invés da instância executora extrair os dados produzidos e enviar através da mensagem MPI, ela envia apenas a notificação de que terminou a execução, e uma linha de execução na instância coordenadora se preocupa em extrair esses dados produzidos e armazená-los na base de proveniência. Quando todas as ativações disponíveis terminam de executar, o ChironSN espera essa linha de execução terminar de extrair todos os dados, para então prosseguir com a execução de outras atividades.

3.3.5 Adição de parâmetro à instrumentação de ativação

Cada ativação tem um diretório próprio, onde arquivos produzidos e consumidos ficam armazenados. Quando o ChironSN invoca a execução do programa associado à ativação, ele o faz de tal forma que o programa seja executado no contexto desse diretório. No entanto, quando esse diretório está no HDFS, isso é impossível.

A solução encontrada para notificar o programa sobre o diretório que deve ser manipulado foi adicionar o parâmetro *WORKDIR* às opções de instrumentação do ChironSN. Se a linha de comando utilizada por uma ativação para invocar o programa contiver esse parâmetro, este será substituído pelo diretório de trabalho daquela ativação.

3.4 Os fluxos de execução do ChironSN

Este capítulo apresentou o processo de integração entre Chiron e HDFS, e discorreu sobre modificações adicionais que visam a aumentar as opções de

configurações para a execução do ChironSN, que certamente serão úteis em experimentos futuros.

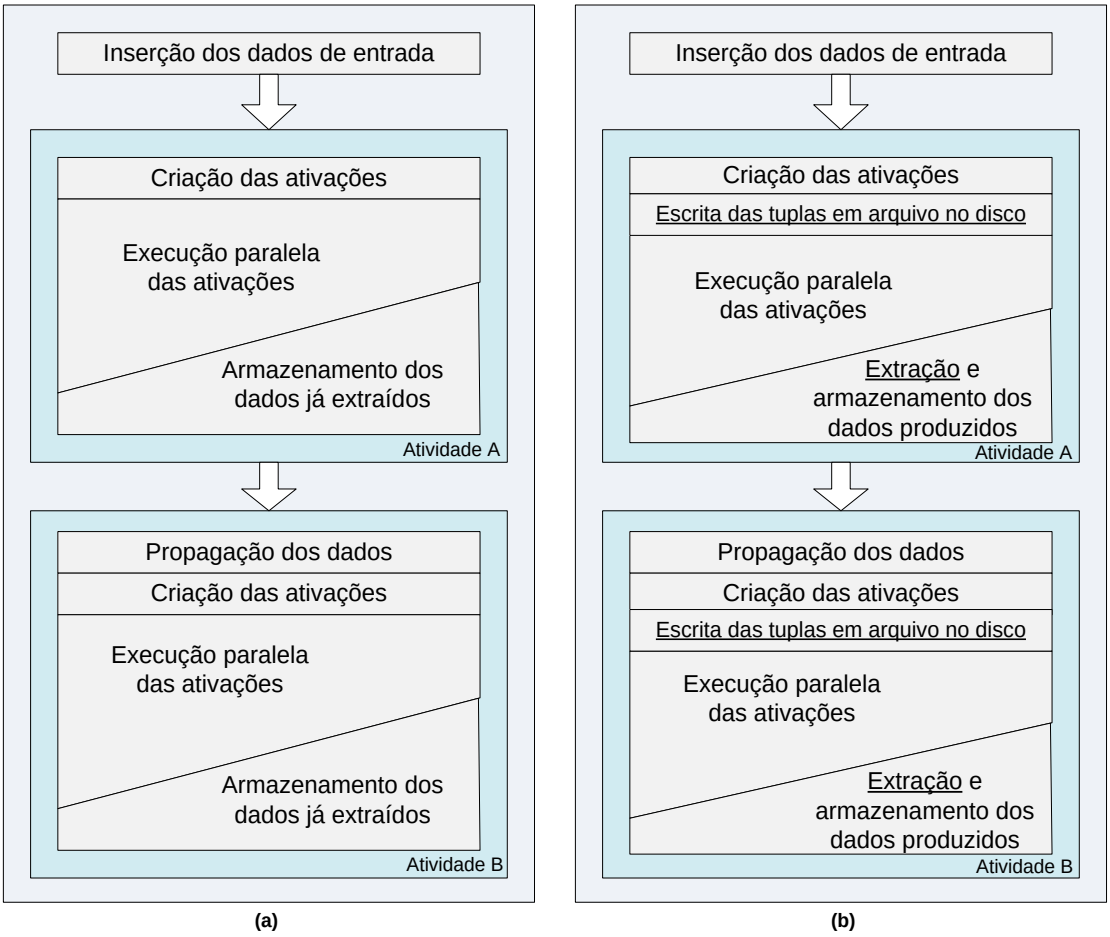

Figura 13 - Fluxo de execução do ChironSN com (a) uma tupla por ativação e (b) múltiplas tuplas por ativação

Para ilustrar as modificações ocorridas no fluxo de execução do ChironSN, a Figura 13 simula dois cenários para a execução de um mesmo *workflow* composto por duas atividades, *A* e *B*, sendo que a atividade *B* é dependente da atividade *A*. Os dados de entrada da atividade *A* estão armazenado no sistema de arquivos, e são importados para a base de proveniência no começo da execução.

A Figura 13a representa a o fluxo de execução padrão do ChironSN, similar ao Chiron, quando apenas uma tupla por ativação é utilizada. Já o cenário ilustrado na Figura 13b exibe o fluxo de execução seguido quando mais de uma tupla é processada por ativação. Em ambos os casos está presente a linha de execução que processa as ativações concluídas e atua em segundo plano, armazenando na base de proveniência as tuplas produzidas pelas ativações.

Chiron e ChironSN apresentam a mesma rotina de execução quando é utilizado o comportamento padrão de uma tupla consumida por ativação. Quando se faz a escolha pelo consumo de múltiplas tuplas por ativação, há a inclusão da etapa onde são escritas em disco as tuplas que serão processadas por cada ativação. Além disso, há também a mudança no momento em que os dados produzidos pelas ativações são extraídos. Para economizar banda na transmissão das mensagens MPI, a extração é feita no nó coordenador através de uma linha de execução secundária, que acessa o diretório das ativações concluídas e recupera as tuplas produzidas para armazená-las na base de proveniência. No comportamento padrão, essa linha de execução apenas persiste os dados que foram extraídos no nó executor e foram incluídos na mensagem MPI.

Capítulo 4 - Avaliação Experimental

Com o intuito de validar o ChironSN e o processo de migração para ambiente de processamento paralelo com memória distribuída, programas com aplicação científica de uma suíte de *benchmark* para Hadoop foram modelados como *workflows* usando a álgebra do Chiron. Além de comparar a execução com o programa equivalente do Hadoop, a análise também mensurou como a exploração da localidade dos dados impactou no tempo de execução dos *workflows* pelo ChironSN. Para um dos *workflows*, houve também a comparação com a versão original do Chiron, que opera com limitações em ambientes de memória distribuída. Este capítulo apresenta a suíte de *benchmark* (seção 4.1); os critérios de seleção dos programas de *benchmark* e a definição dos *workflows* (seção 4.2); a descrição do ambiente de execução (seção 4.3); e a avaliação experimental, desde a preparação (seção 4.4) até os resultados (seções 4.5 e 4.6).

4.1 HiBench

A suíte de *benchmark* HiBench (HUANG *et al*., 2010) é um conjunto de programas cujo objetivo é avaliar diversos aspectos do arcabouço Hadoop, dentre eles: tempo de execução, aceleração e frequência de execução de tarefas, aproveitamento dos recursos computacionais disponíveis, bem como a adequada utilização da largura de banda. Há ainda uma parte do *benchmark* que é responsável por testar a eficiência do HDFS em variados cenários de leitura e escrita de arquivos.

O HiBench é um projeto de código aberto e estão disponíveis todos os detalhes de implementação, configuração e execução dos *benchmarks*. Quando a implementação é aproveitada de outro projeto, há a indicação de como acessá-la.

A lista de programas incluídos na suíte HiBench é dividida em quatro categorias, conforme mostrado a seguir:

- Micro *benchmarks*: ordenação e contagem de palavras;
- Busca e indexação na web: Indexação Nutch e *PageRank*;
- Aprendizado de máquina: Classificação Bayesiana e Clusterização K-Means;
- *Benchmark* do HDFS.

Um diferencial do HiBench é que, além da inclusão dos programas habituais de *benchmark* para Hadoop (e Map/Reduce em geral), há também a preocupação em abordar problemas do mundo real que operam sobre conjuntos de dados não-sintéticos. O programa de contagem de palavras pode ser utilizado como ponto de partida para calibrar os parâmetros de uma avaliação experimental. A utilização de aplicações como Classificação Bayesiana e *PageRank* permite que se faça um julgamento mais preciso acerca da eficiência do Hadoop na análise de dados em larga escala, e como ele contribui para o avanço da experimentação científica.

Para os micro *benchmarks*, o HiBench aproveita um conjunto de componentes utilitários que são disponibilizadas pelo próprio Hadoop e estão incluídas em todas as distribuições. A geração do conjunto de dados de entrada é um programa Map/Reduce cujo produto são linhas de texto formadas por palavras aleatórias, que são extraídas do dicionário do sistema operacional da máquina.

No programa de contagem de palavras, cada mapeamento itera sobre palavras de cada linha de texto e produz (*palavra*, *1*). A etapa de combinação reduz as partições ao processar somas intermediárias para ocorrências da mesma palavra, e os redutores consolidam a contagem das palavras para o texto inteiro, produzindo pares (*palavra, N*), onde N é total de ocorrências de cada palavra.

O programa de ordenação consiste em colocar as linhas de um texto em ordem alfabética. Para tal, as funções de mapeamento e redução não executam nenhuma computação sobre as linhas de texto, simplesmente propagando os dados de entrada para a saída. Toda a ordenação do texto é feita pela etapa de ordenação presente no fluxo de execução de um programa Map/Reduce. Quando as linhas de texto chegam à etapa de redução, elas já estão na ordem alfabética pretendida.

Indexação Nutch e *PageRank* foram considerados para o HiBench por representarem duas abordagens para sistemas de indexação e busca em larga escala. Ambos são capazes de processar milhões de páginas *web* com eficiência utilizando programas Map/Reduce.

A indexação Nutch é parte integrante do motor de busca Apache Nutch (2014), um projeto em código aberto que usa o Hadoop como plataforma de apoio ao processamento dos dados. A implementação em Map/Reduce mapeia todos os links

contidos em cada página web e a redução agrupa os links para criar um índice invertido identificando através de quais páginas se chega a uma outra página.

No caso do *PageRank* (PAGE *et al*., 1998), a implementação usada pelo HiBench consiste em uma sequência de programas Map/Reduce encadeados. Devido à natureza do algoritmo de *PageRank*, algum desses programas são executados diversas vezes até que uma condição de parada pré-definida seja satisfeita.

Os *benchmarks* de Classificação Bayesiana e Clusterização K-Means foram considerados principalmente em razão da relevância da área de aprendizado de máquina. Para estes *benchmarks*, o Hibench utiliza o Apache Mahout (2014), um projeto em código aberto para aprendizado de máquina e mineração de dados, construído sobre a arquitetura escalável do Hadoop.

O *benchmark* de Classificação Bayesiana, antes de executar o programa Map/Reduce do classificador, efetua um pré-processamento dos dados de entrada (um conjunto de documentos de texto) para coletar as estatísticas necessárias à classificação. A parte mais custosa desse pré-processamento é o cálculo do *TF-IDF* (*Term Frequency–Inverse Document Frequency*), uma medida que expressa a relevância de termos em uma coleção de documentos, e é utilizada como fator de ponderação pelo classificador.

A primeira fase do *benchmark* de Clusterização K-Means é um programa Map/Reduce que opera iterativamente sobre um conjunto de vetores numéricos de dimensão pré-definida a fim de determinar os vetores centroides. Quando a execução converge ou um número pré-definido de iterações é atingido, outro programa Map/Reduce é executado para concluir a clusterização e vincular os vetores de entrada aos centroides obtidos.

Por fim, há o *benchmark* para HDFS, que é uma extensão daquele que é disponibilizado pelo próprio Hadoop. Na versão do Hadoop, tarefas simultâneas de mapeamento executam leituras e escritas no disco, e emitem como resultado o volume de dados lidos/escritos e o tempo total de execução da tarefa. Posteriormente, uma tarefa de redução calcula um valor médio, contabilizando o total de dados lidos/escritos pelos mapeamentos e dividindo pelo tempo total gasto.

A versão estendida desse *benchmark*, apresentada pelo HiBench, executa as mesmas tarefas simultâneas de mapeamento, com leituras e escritas no disco. A diferença é que agora essas tarefas emitem resultados parciais, indicando quanto foi lido/escrito até um instante *t*, que é o sincronizado entre os nós. Quando a tarefa de redução consolida os dados produzidos pelos mapeamentos, é possível analisar o comportamento das taxas de leitura e escrita em cada um dos intervalos definidos pelos instantes *t*.

4.2 Seleção de *benchmarks* para avaliar o ChironSN

Embora cada um dos *benchmarks* disponíveis no HiBench seja relevante à sua maneira, alguns deles possuem detalhes de implementação que dificultam ou impossibilitam sua adaptação para o ChironSN, por fazerem uso de alguns recursos específicos da arquitetura do Hadoop.

Como exemplo, podemos citar os *benchmarks* de *PageRank* e Clusterização K-Means, que possuem uma etapa com execuções iterativas de um programa Map/Reduce. Na definição do *workflow* sobre a qual o ChironSN opera não há como indicar a execução iterativa de um conjunto de atividades.

Um exemplo de fator complicador para adaptação ao ChironSN são os programas Map/Reduce que geram conjuntos de dados independentes, sendo que apenas um destes foi produzido pela tarefa de redução. Enquanto esse conjunto de dados oficial é enviado como entrada da próxima etapa, os outros dados ficam armazenados em disco para uso em etapas posteriores da execução do *benchmark*. Esse comportamento é encontrado nos programas de *PageRank* e Classificação Bayesiana.

Com o ChironSN, cada ativação de uma determinada atividade tem um diretório próprio para armazenar seus arquivos, o qual não é acessado por outras ativações. Considerando um *workflow* com cinco atividades, se uma ativação da atividade nº 1 criasse um arquivo temporário para ser consumido pela atividade nº 4, as informações desse arquivo teriam que ser propagadas por todas as atividades até chegar o momento da atividade nº 4 ser executada. No Hadoop, todas as tarefas escrevem em um diretório comum, o que facilita essa operação de manipular arquivos temporários.

O *benchmark* para o HDFS não é relevante para essa dissertação, visto que ele consiste apenas em medir as taxas de escrita e leitura no disco, o que independe da

abordagem algébrica e do ChironSN. O *benchmark* de ordenação é outro que foi descartado por não representar um problema condizente com a avaliação proposta, uma vez que não apresenta computação nas funções de mapeamento e redução, sendo utilizado apenas para mensurar o desempenho da ordenação dos resultados intermediários de um programa Map/Reduce.

Dentre os *benchmarks* restantes, o programa de contagem de palavras foi o primeiro selecionado para a avaliação do ChironSN em ambientes de memória distribuída. Embora seja um programa de baixa complexidade, ele é frequentemente utilizado como etapa intermediária em experimentos que envolvem mineração de texto e busca e recuperação de informação.

O *benchmark* de indexação Nutch era outro candidato potencial para ser avaliado com o ChironSN. No entanto, ele é um programa Map/Reduce de complexidade baixa, assim como a contagem de palavras, e a intenção era usar algum *benchmark* que fosse composto por programas Map/Reduce encadeados.

Observando mais atentamente os *scripts* de execução dos *benchmarks* do HiBench, percebeu-se que na Classificação Bayesiana o pré-processamento do conjunto de dados de entrada e o treinamento do classificador são dois *scripts* independentes, executados sequencialmente. O classificador utiliza arquivos temporários e sua adaptação estava além dos objetivos desta dissertação. No entanto, o pré-processamento de texto era viável, e atendia à condição desejável de conter um encadeamento de programas Map/Reduce.

Essa etapa de pré-processamento consiste principalmente no cálculo do *TF-IDF* para a coleção de documentos que compõem o conjunto de dados de entrada do classificador Bayesiano. Operações adicionais desse pré-processamento foram removidas/omitidas para que o *benchmark* executasse apenas o cálculo do *TF-IDF*.

As subseções seguintes (4.2.1, 4.2.2) mostram os detalhes da adaptação de cada um dos *benchmarks* selecionados para a abordagem algébrica utilizada pelo ChironSN. O Apêndice A apresenta os arquivos XML utilizados para descrever os *workflows* e os experimentos.

4.2.1 *Workflow* para o programa de contagem de palavras

No programa de contagem de palavras, cada mapeamento itera sobre um conjunto de linhas de texto, produzindo um par *(W, 1)* para cada palavra *W* encontrada. A etapa de combinação calcula um soma parcial de cada palavra, produzindo *lista(W, n)*. As tarefas de redução recebem pares *(W, lista(n_1, n_2, ..., n_k))* e produzem *lista(W, N)*, onde $N = n_1 + n_2 + ... + n_k$, com a contagem de cada palavra na amostra.

O *workflow* de contagem de palavras é composto de duas atividades, uma de fragmentação (*SPLIT_MAP*) e uma de redução (*REDUCE*). A Figura 14 exibe uma representação do fluxo de dados entre as atividades desse *workflow*.

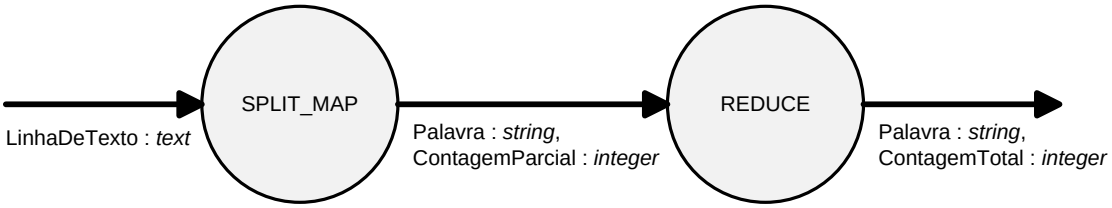

Figura 14 - *Workflow* para contagem de palavras

Como não há o conceito de combinação de resultados intermediários na álgebra do ChironSN, optou-se por implementar um armazenamento em memória das contagens parciais. O programa invocado por uma ativação de fragmentação só produz os dados de saída quando o seu bloco de dados de entrada foi completamente consumido. Dessa maneira, palavras não se repetirão nesse conjunto de resultados intermediários.

4.2.2 *Workflow* para o programa de cálculo do *TF-IDF*

O cálculo do *TF-IDF* ("*term frequency–inverse document frequency*") tem por objetivo mensurar a relevância de um termo dentro de um documento em uma coleção, mas considerando todos os documentos dessa coleção. O *TF-IDF* é largamente utilizado nas áreas de Recuperação de Informação e Mineração de Texto, sendo aplicado como fator de ponderação na indexação de termos-chave para coleções de documentos.

A frequência de termo (*TF*) diz respeito apenas à importância desse termo em cada um dos documentos em que ele aparece. A frequência inversa de documento (*IDF*) mede o quanto um termo é representativo de um subconjunto de documentos da coleção, ou o quão específico um termo é para determinado documento.

A conjunção das duas estatísticas (*TF-IDF*) permite identificar, dentre os termos mais frequentes dos documentos, quais são os mais relevantes para representar o conteúdo de um único documento da coleção (BAEZA-YATES & RIBEIRO-NETO, 2013).

Dependendo de como os dados de entrada – uma coleção de documentos – estão organizados, são necessários de 2 a 4 programas de Map/Reduce encadeados para calcular o *TF-IDF* de todos os termos. Os três primeiros programas costumam ser usados para extrair dados requeridos para o cálculo. São eles:

- Número de ocorrências de cada um dos termos em cada um dos documentos (*NumOcorrenciasDoTermoNoDoc*);
- Número total de termos em cada documento (*NumTermosNoDoc*);
- Número de documentos em que cada um dos termos aparece (*NumDocsDiferentes*);
- Número total de documentos (*NumDocs*). Esse valor normalmente é conhecido de antemão, sendo passado como parâmetro.

O Apêndice B contém um pseudocódigo para quatro programas Map/Reduce que são executados sequencialmente e produzem os valores do *TF-IDF*. A versão adaptada para a álgebra do ChironSN é mais compacta e conta com menos etapas (atividades) que a versão equivalente do Hadoop.

A Figura 15 contém o *workflow* utilizado na execução do cálculo do *TF-IDF* no ChironSN. Durante a primeira atividade é feita a fragmentação dos documentos presentes na coleção. Para cada um dos documentos é extraído o total de termos e a quantidade de ocorrências de cada termo naquele documento.

A atividade de redução opera sobre fragmentos agrupados pelos termos. Como a atividade anterior garante que o par *(Termo,NomeDoc)* seja único, basta fazer uma contagem de *NomeDoc* para saber em quantos documentos o *Termo* aparece.

A última atividade computa o valor do *TF-IDF* para cada uma das tuplas indexadas por *(Termo,NomeDoc)*, utilizando os valores adicionais da tupla e a quantidade total de documentos, que é passada como parâmetro.

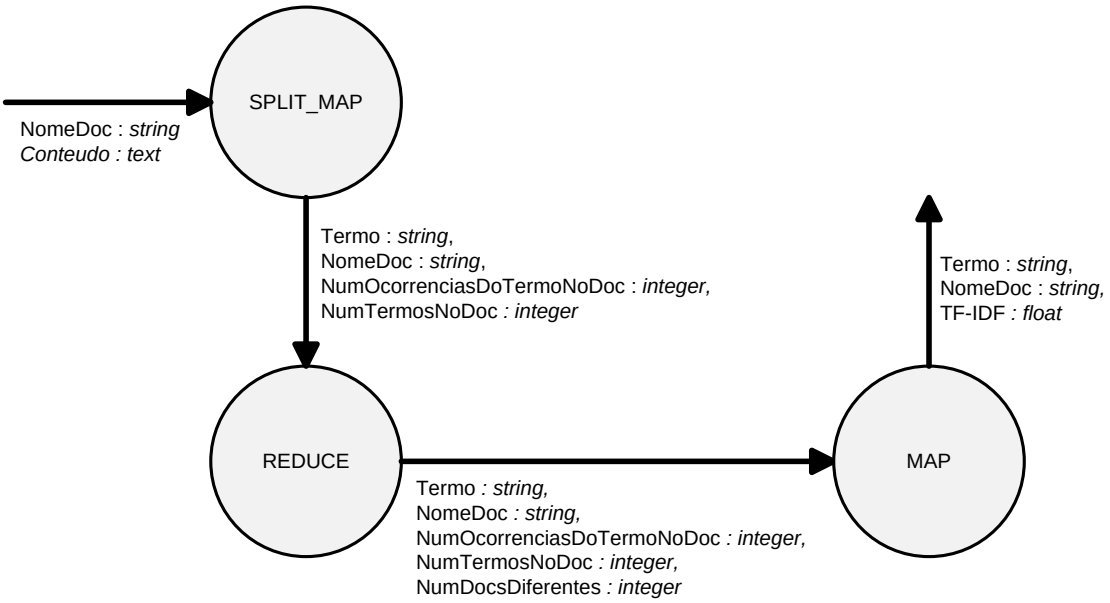

Figura 15 - *Workflow* para o cálculo do *TF-IDF*

4.3 O ambiente de execução

O Grid'5000[5] (CAPPELLO *et al.*, 2005) foi a plataforma utilizada para execução dos experimentos necessários à validação do ChironSN. O Grid'5000 é um instrumento de suporte à pesquisa científica, oferendo uma infraestrutura computacional onde é possível a execução de experimentos em todas as áreas da Ciência da Computação, com destaque para computação de alto desempenho, computação distribuída e, mais recentemente, *big data*. O Grid'5000 é mantido e desenvolvido pelo INRIA[6], e conta com o apoio de muitas universidades e organizações de fomento à pesquisa.

Um diferencial do Grid'5000 no apoio à experimentação científica é a possibilidade de configurar totalmente o ambiente de execução. Os nós reservados em um *cluster* funcionam como máquinas virtuais, onde o cientista pode instalar o *software* que precisar, mudar as configurações que forem necessárias, e ao final, salvar essa máquina virtual para reaproveitar em futuros experimentos.

[5] Grid'5000 - https://www.grid5000.fr/

[6] INRIA - Instituto Francês de Pesquisas em Ciência da Computação e Automação

No caso dos experimentos executados para essa dissertação, foi utilizado um ambiente padrão disponibilizado pelo Grid'5000, baseado no sistema operacional Debian[7]. Nesse ambiente foram instalados e configurados os *softwares* listados abaixo:

- Java 7 (1.7.0, *update* 45);
- Hadoop 1.2.1;
- HiBench 2.2;
- PostgreSQL 9.1.

Com relação ao *hardware*, utilizou-se até 32 nós do *cluster griffon*, localizado no sítio Nancy. Cada um desses nós tem as seguintes especificações:

- Modelo Carri System CS-5393B;
- Dois processadores (Intel Xeon L5420 @ 2.5GHz), com quatro núcleos cada;
- 16GB de memória RAM;
- 320GB de armazenamento local;

Uma rede de alto desempenho – Infiniband-20G (20Gb/s) – conecta os nós do *cluster*.

4.4 Detalhes sobre a preparação dos experimentos

O conjunto de dados de entrada processado por um *workflow* executado no ChironSN é o mesmo utilizado pelo *benchmark* equivalente no HiBench. A geração desse conjunto de dados fica a cargo do HiBench, que possui programas específicos para essa finalidade. O *benchmark* de contagem de palavras utilizou amostras de 1GB. Essas amostras consistem em linhas de texto geradas aleatoriamente, cujas palavras são extraídas do dicionário de palavras do sistema operacional. Para a execução do *TF-IDF*, o conjunto de dados de entrada é composto por 25.000 linhas de texto, onde cada linha de texto representa o conteúdo de um documento em uma coleção.

Como os *workflows* criados para os experimentos simulavam programas de Map/Reduce, foram utilizadas as estratégias *Primeira-Atividade-Primeiro* (mapeamento

[7] Debian 7.6 (*wheezy*) - "O Sistema Operacional Universal" - http://www.debian.org/releases/wheezy/

precisa concluir antes que a redução comece) e despacho dinâmico (cada nó executa apenas uma ativação por vez).

Quando o ChironSN está utilizando mais de uma tupla por ativação, o tempo total de execução de um *workflow* com uma atividade de mapeamento e uma de redução é medido conforme expresso na Figura 16.

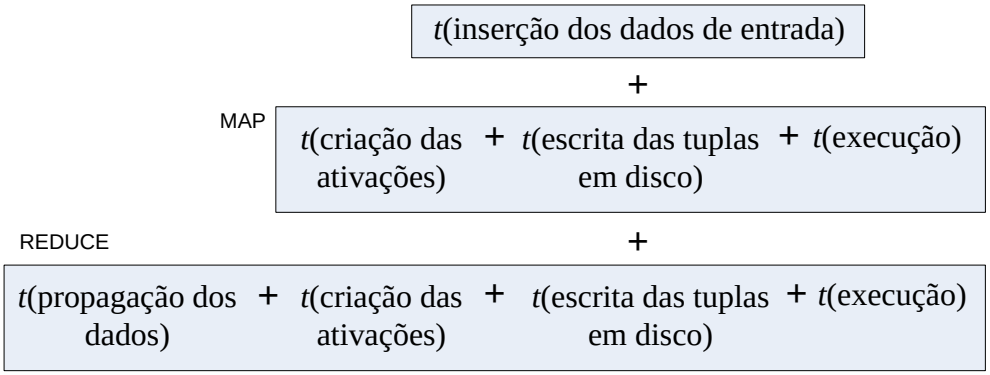

Figura 16 - Tempo de execução de um *workflow* com múltiplas tuplas por ativação

No entanto, os resultados que serão apresentados desconsideraram os três primeiros intervalos de tempo. Experimentos preliminares mostraram que quando os dados de entrada são apenas referências a arquivos, as etapas de inserção de dados e criação das ativações demoram um tempo muito inferior ao tempo de execução, tornando-os desprezíveis para o cálculo do tempo total. O tempo de escrever as tuplas em disco não existe quando o ChironSN processa referências a arquivos. Para as atividades de *Reduce*, todos os intervalos de tempo são considerados.

Para avaliar como a quantidade de nós interfere no tempo de execução dos *workflows*, seriam utilizadas configurações com 1, 2, 4, 8, 16 e 32 nós. No entanto, como também se pretendia avaliar o impacto da localidade dos dados, foram utilizados apenas 4, 8, 16 e 32 nós. Isso foi feito devido à utilização de um fator de replicação igual a 3 em todas as execuções.

Todos os valores utilizados na análise experimental são médias de valores obtidos a partir de 3 execuções consecutivas do mesmo experimento. Em todas as execuções do ChironSN, foi utilizado o parâmetro *numTuplesPerActivation* com valor igual a -1, de tal forma que cada nó executaria apenas uma ativação por atividade. Assim, a quantidade de dados consumidos por cada ativação diminuía à proporção que a quantidade de nós aumentava.

As duas seções seguintes (4.5, 4.6) apresentam os resultados experimentais obtidos a partir das execuções com Hadoop e ChironSN para os dois problemas selecionados. O Apêndice C contém tabelas com os dados coletados durante a avaliação experimental.

4.5 Contagem de palavras

O primeiro resultado a ser analisado é a variação de tempo de execução obtida com os três cenários considerados, conforme expresso na Figura 17. No caso da contagem de palavras, o ChironSN obteve melhor desempenho que o equivalente no Hadoop. Quando se considera apenas o ChironSN, é possível observar como o aproveitamento da localidade dos dados é capaz de reduzir o tempo de execução.

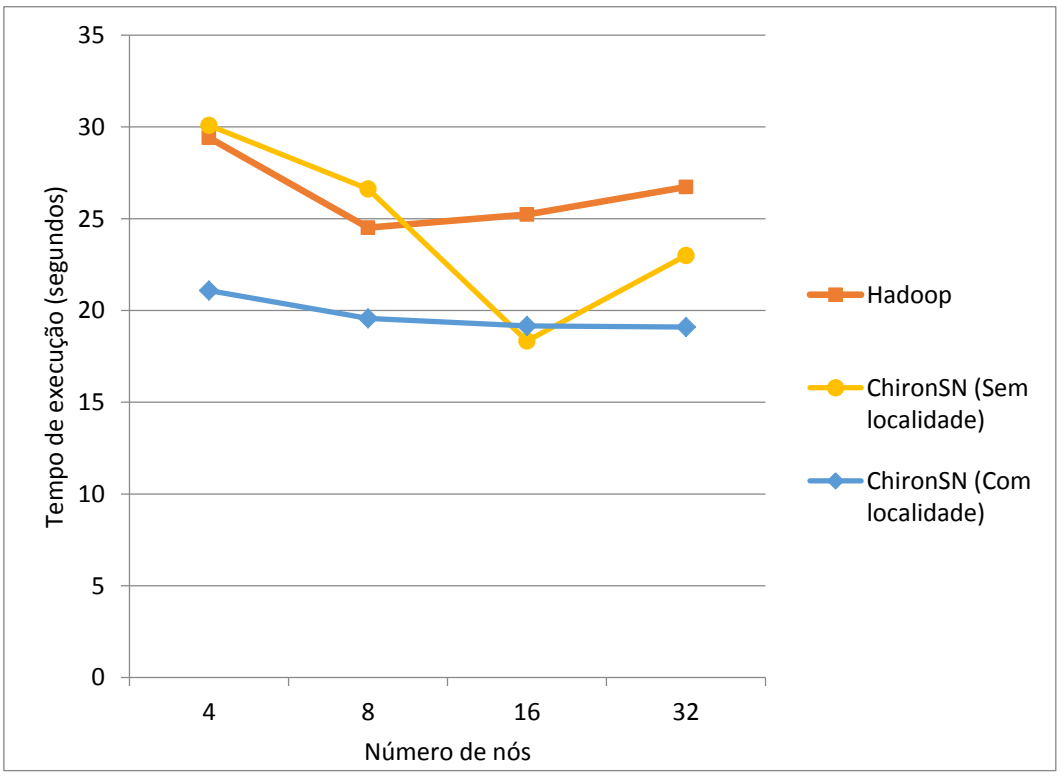

Figura 17 - Contagem de Palavras - ChironSN e Hadoop

É importante dizer que os resultados produzidos pela atividade de *SplitMap* foram extraídos e inseridos na base de proveniência para serem ordenados e agrupados, como preparação para a atividade de *Reduce*. No caso da contagem de palavras, essa abordagem foi melhor que salvar referências a arquivos, pois é menos custoso salvar (*Palavra*, *ContagemParcial*) do que (*Palavra*, *FileRef*).

Para avaliar de forma mais detalhada o ganho com o aproveitamento da localidade dos dados, isolou-se o tempo médio de execução das ativações de cada atividade do *workflow*. A Figura 18 compara esse tempo médio de execução para a atividade de *SplitMap*. Diferente do tempo total de execução do *workflow*, a localidade dos dados interferiu pouco no tempo de execução desse conjunto de ativações.

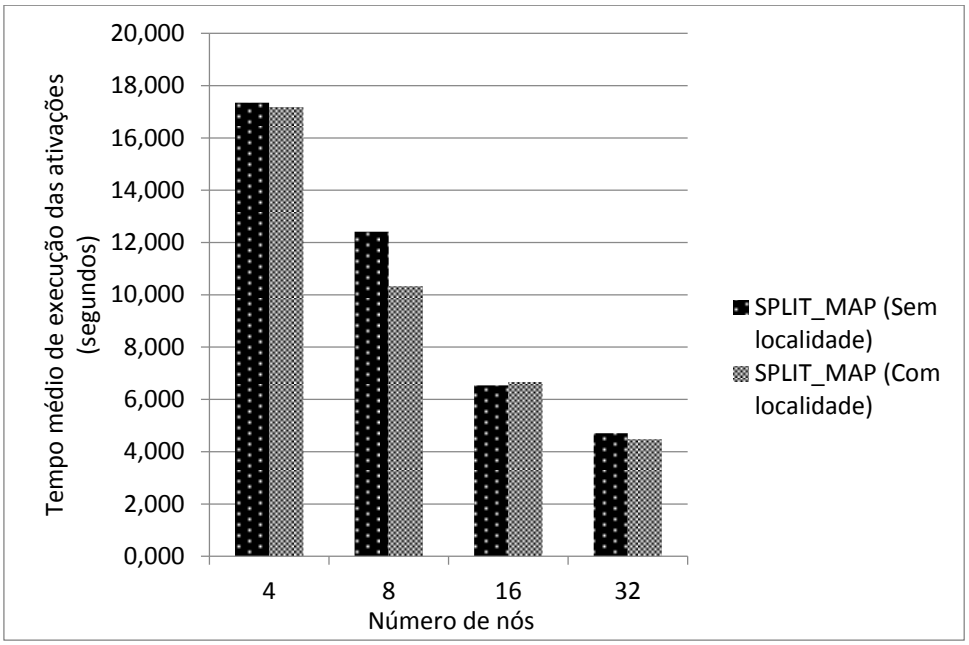

Figura 18 - Contagem de palavras - Tempo médio de execução das ativações de *SplitMap*

A Figura 19 contém um gráfico análogo ao da imagem anterior, mas centrado na atividade de *Reduce* do *workflow*. Para o conjunto de ativações dessa atividade, o aproveitamento da localidade foi mais efetivo, resultando em uma maior redução no tempo de execução das ativações, que se refletiu no tempo total de execução do *workflow*.

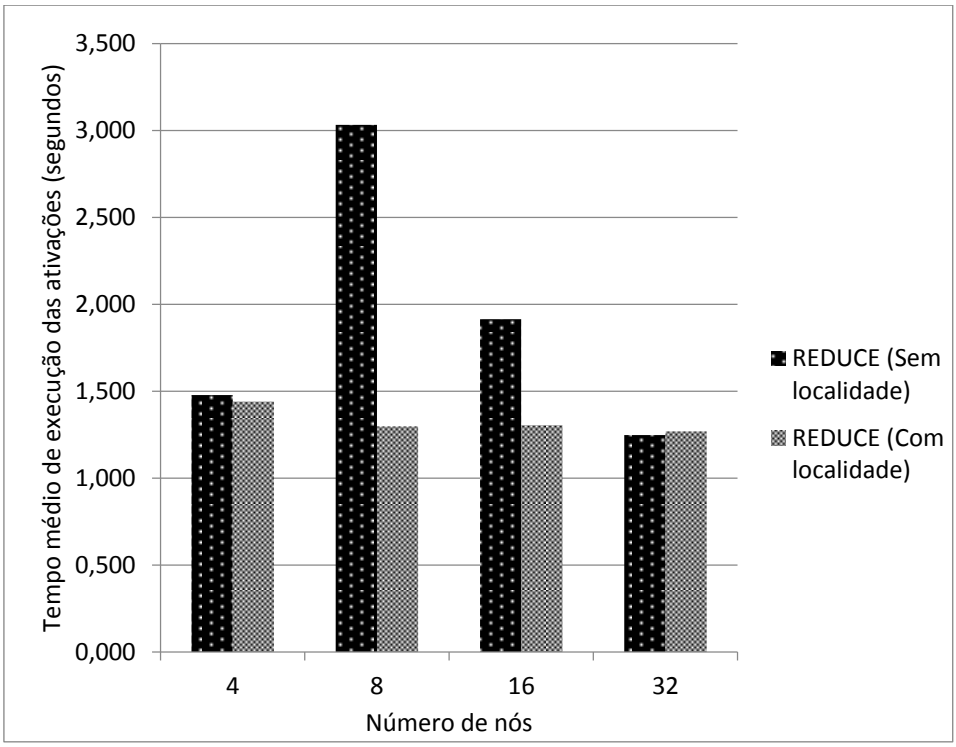

Figura 19 - Contagem de palavras - Tempo médio de execução das ativações de *Reduce*

Para complementar a análise, foi executado o *workflow* de contagem de palavras no ambiente de memória distribuída com a versão original do Chiron. Como essa versão requer um disco compartilhado que não estava disponível, toda a transmissão dos dados para serem processados pelas ativações teve que ser efetuada através das mensagens MPI. Para tal, todos os dados de entrada foram inseridos na base de proveniência. As tuplas que seriam consumidas foram enviadas via MPI e as tuplas produzidas foram extraídas e retornadas ao nó coordenador também via MPI. Como o Chiron opera com uma tupla por ativação, a quantidade de ativações foi bem alta, especialmente na primeira atividade, que manipula milhares de linhas de texto.

A primeira tentativa de execução utilizou a mesma amostra do ChironSN, mas 1GB mostrou-se inviável, mesmo com 32 nós. Passadas 2 horas de execução, nem 10% das ativações de *SplitMap* haviam sido executadas. Preferiu-se cancelar a execução e recomeçar com apenas um fragmento do conjunto de dados originais: 32MB. A nova execução foi concluída após cerca de 1 hora e 9 minutos, com os resultados esperados. Esse experimento mostrou que o Chiron é capaz de operar também com memória distribuída, embora sua eficiência fique comprometida devido às condições adversas. Com o ChironSN, a amostra de 1GB foi processada em menos de 20 segundos com 32 nós.

Entretanto, esse experimento também evidenciou que criar muitas ativações, seja no Chiron ou no ChironSN, pode ser prejudicial quando o processamento a ser feito por cada uma delas é muito rápido. Na execução com 32MB, cada ativação de *SplitMap* processou apenas uma linha de texto e o tempo médio de execução das ativações foi de 0,78s, ao passo que o tempo de transmissão (ida+volta) dessas ativações foi de aproximadamente 1,52s. Esses intervalos de tempo mostram que um *overhead* acompanha as ativações, e por isso é importante planejar o volume de dados que cada ativação consumirá. Para efeito de comparação, a Figura 18 mostra que as ativações foram executadas em pouco mais de 4s com uma amostra 32 vezes maior (1GB/32 nós, diferente de 32MB/32 nós).

4.6 *TF-IDF*

Com os resultados da execução do *workflow* para o cálculo do *TF-IDF*, a primeira análise envolve a questão da localidade dos dados. Conforme ilustrado no gráfico da Figura 20, apesar do resultado adverso com 8 nós, a localidade foi bem aproveitada nas outras configurações, principalmente na execução com 32 nós.

Outra análise sobre a Figura 20 mostra que é bem nítida a vantagem obtida pelo programa de *TF-IDF* do Hadoop. Tal discrepância, conforme ilustrado no gráfico da Figura 20, pode ser explicada pela característica centralizadora da base de proveniência do ChironSN. No caso das atividades de *Reduce*, a ordenação e o agrupamento dos valores por chaves é feito pelo SGBD, o que leva à extração e inserção de todos os dados produzidos na atividade de *SplitMap*. No cenário de execução considerado, a atividade de *Reduce* precisa ordenar uma quantidade de tuplas que equivale a todos os pares (*Termo*,*NomeDoc*) existentes na coleção de documentos (aproximadamente 7.500.000 tuplas). Em uma comparação simples, a quantidade tuplas produzidas pelo *workflow* de contagem de palavras na atividade de *SplitMap* aproxima-se “apenas” de 30.000.

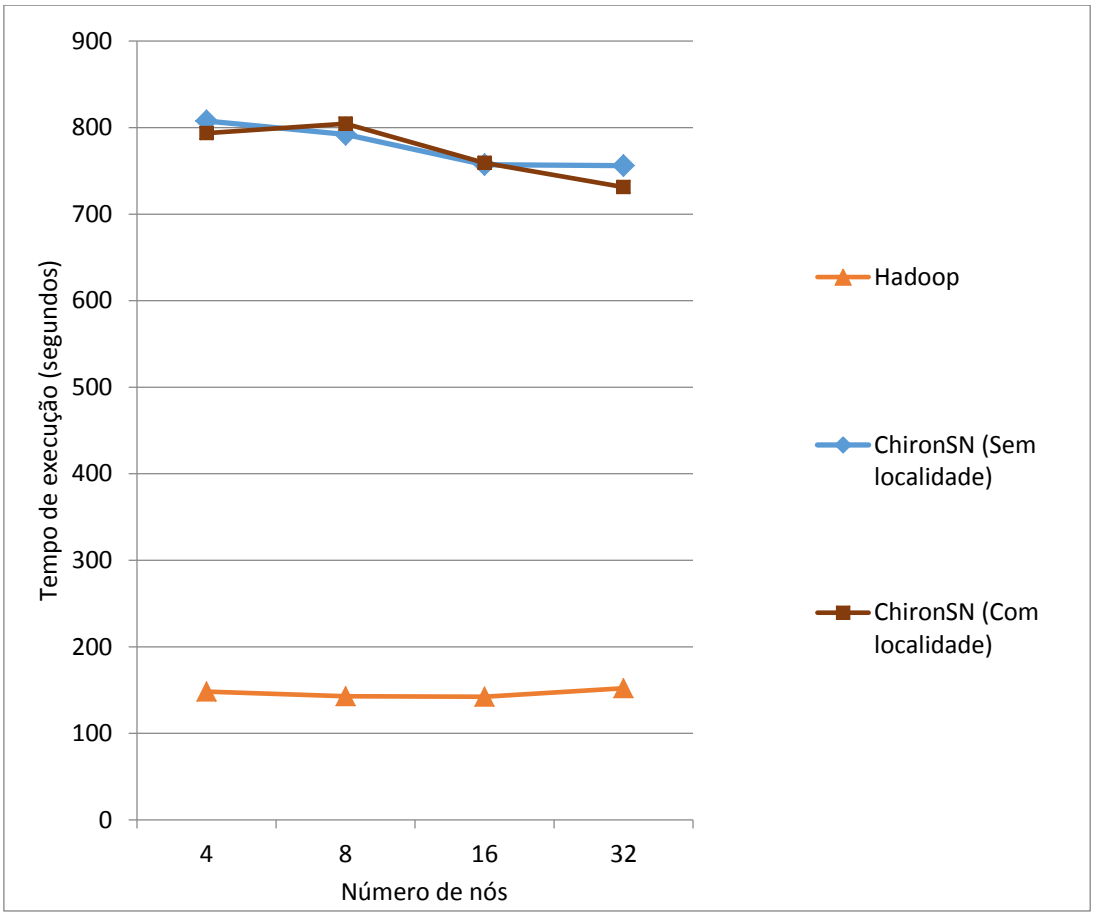

Figura 20 - *TF-IDF* - ChironSN e Hadoop

O tempo gasto com a ordenação e o agrupamento das tuplas deve ser somado a outros dois intervalos de tempo:

- O tempo para extrair do disco e transferir para o nó coordenador (base de proveniência) todas as tuplas produzidas na atividade anterior (*SplitMap*);
- O tempo para escrever as tuplas (já ordenadas e separadas por chave) no disco, no diretório de cada uma das ativações de *Reduce*.

O tempo total gasto em cada uma das atividades do *workflow* de *TF-IDF* está exibido na Figura 21 de tal forma que é fácil perceber como a extração dos dados e a ordenação representam a maior parte do tempo de execução das atividades de *SplitMap* e *Reduce*, respectivamente. A terceira atividade é bem mais rápida porque cada uma das ativações de *Map* processará um dos arquivos que foi gerado pelas ativações de *Reduce*. Devido à característica do operador *Map*, cada tupla consumida equivalerá a uma tupla produzida e ambas as tuplas conterão apenas referências a arquivos.

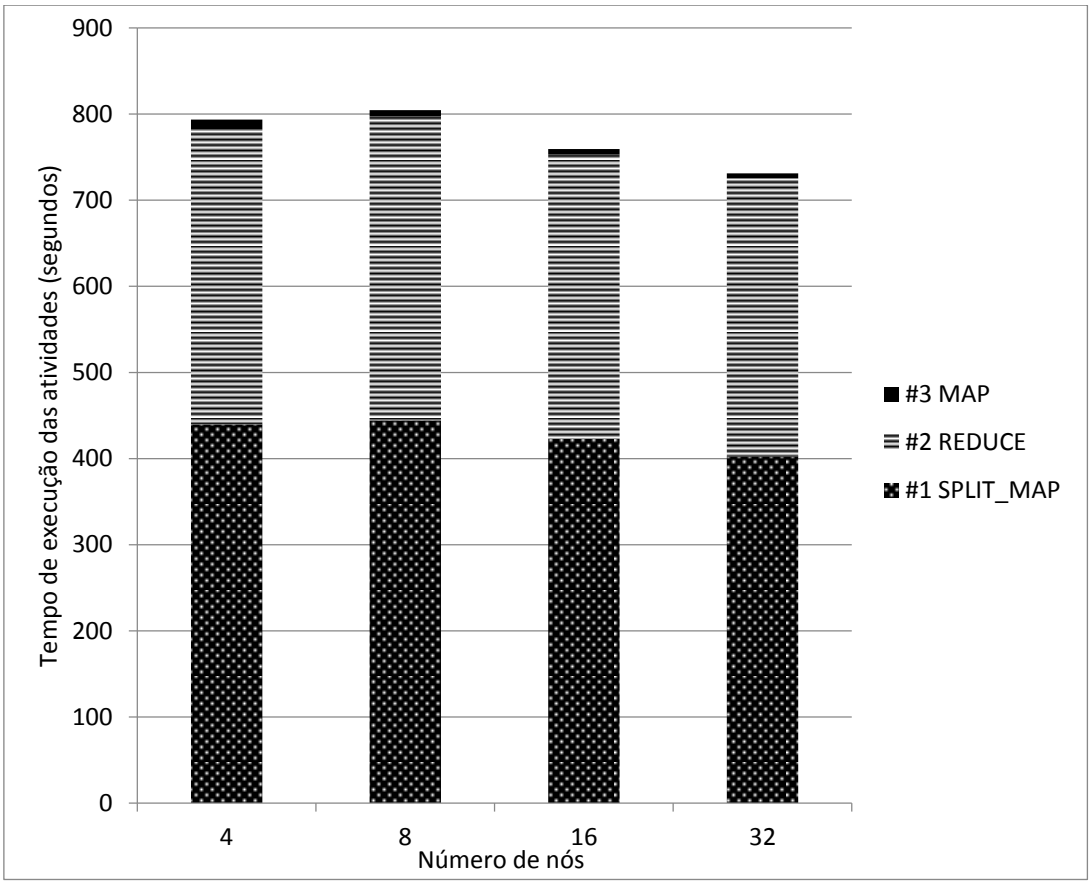

Figura 21 - *TF-IDF* - Tempo de execução das três atividades encadeadas

Os três próximos gráficos (Figura 22, Figura 23 e Figura 24) exibem o tempo médio de execução das ativações para cada uma das três atividades do *workflow* de *TF-IDF*, comparando as abordagens com e sem aproveitamento da localidade dos dados. É possível observar que nas 3 atividades, em pelo menos 3 das 4 configurações, a localidade fez diferença e ajudou a reduzir o tempo de execução.

Nos gráficos relativos às atividades de *SplitMap* e *Reduce*, é possível também comparar com a Figura 21 e observar como o tempo de execução da atividade é muito superior ao tempo médio de execução das ativações. É importante lembrar que foi executada apenas uma ativação por nó e a extração dos dados foi feita no nó coordenador, não sendo contabilizada no ciclo de vida de uma ativação.

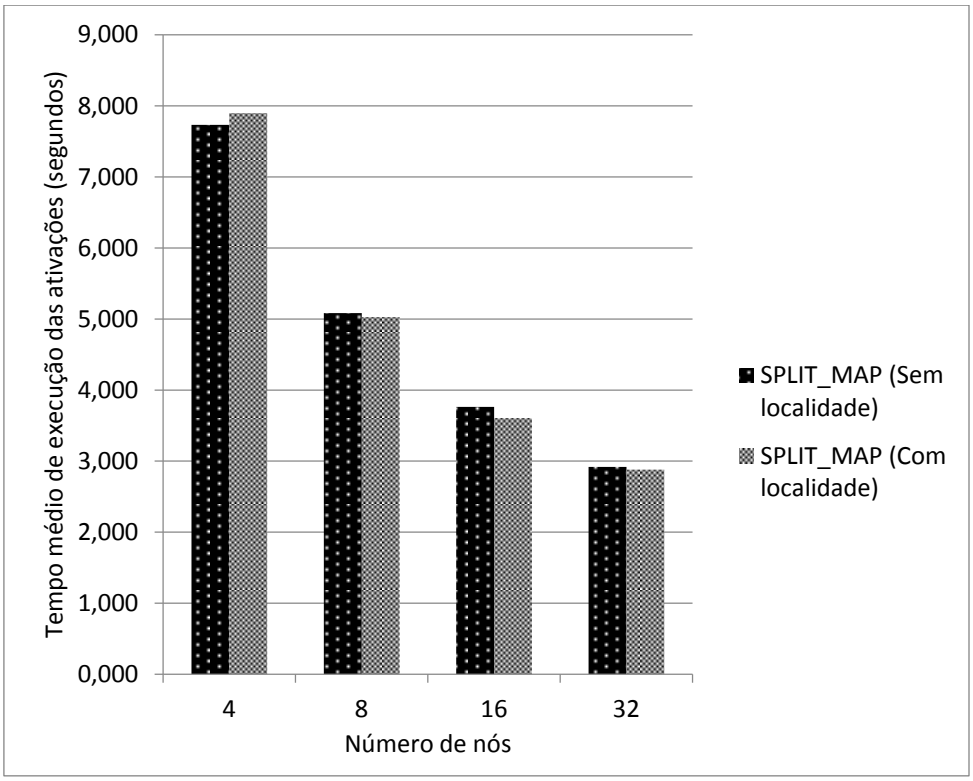

Figura 22 - *TF-IDF* - Tempo médio de execução das ativações de *SplitMap*

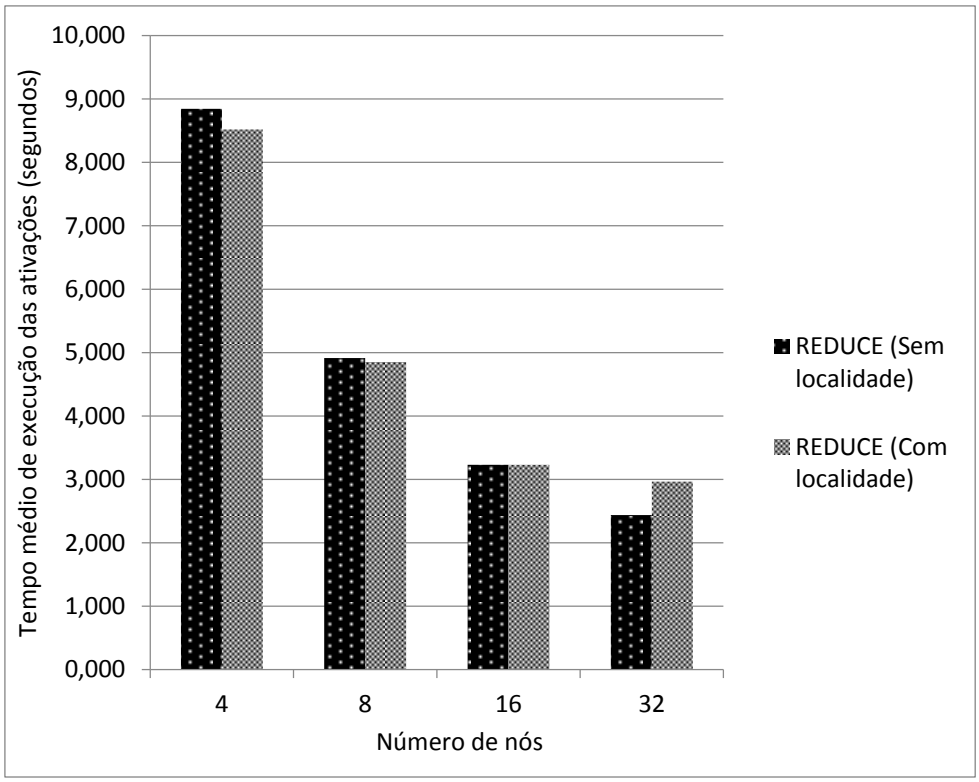

Figura 23 - *TF-IDF* - Tempo médio de execução das ativações de *Reduce*

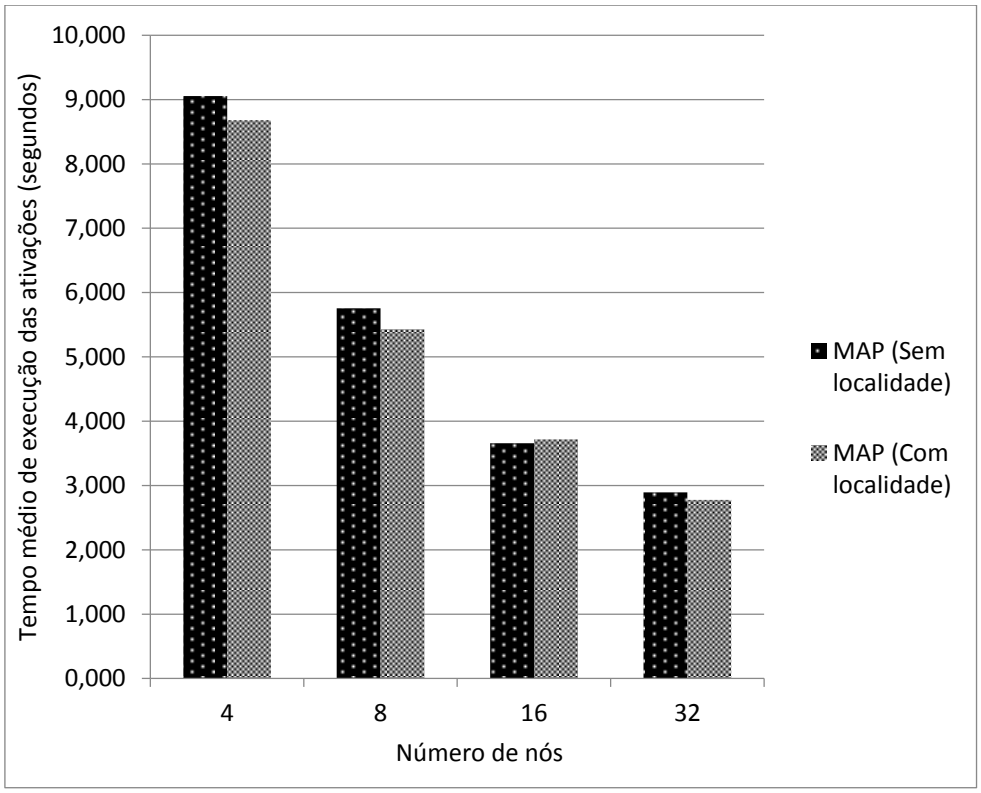

Figura 24 - *TF-IDF* - Tempo médio de execução das ativações de *Map*

4.7 Considerações sobre os resultados experimentais

A avaliação experimental conduzida para validar o ChironSN mostrou que o protótipo apresentado nesta dissertação está apto a operar em ambientes de memória distribuída, conforme a proposta original. A análise do conjunto de resultados obtidos evidencia que o ChironSN obtém melhor desempenho em alguns cenários de uso, como o problema de contagem de palavras.

Adicionalmente, os resultados do cálculo do *TF-IDF* apontam modificações que permitirão ao ChironSN aproveitar de forma mais eficiente os recursos do ambiente de processamento paralelo com memória distribuída. O perfil centralizador da base de proveniência do ChironSN representa um ponto de contenção na execução de um *workflow* quando há a necessidade de ordenar e agrupar uma quantidade muito grande de tuplas. Uma versão futura do ChironSN pode implementar um modelo de agregação similar àquele utilizado pelo Hadoop, onde os dados produzidos em um nó são fragmentados e enviados diretamente aos nós que executarão a agregação. Essa estratégia minimiza a centralização, balanceando o processamento e a transmissão de dados entre os nós computacionais.

Cabe ainda destacar a implementação da exploração da localidade dos dados, que resultou em ganhos no tempo de execução dos *workflows* na maior parte dos cenários considerados. Quando um experimento manipula um grande volume de dados em ambientes de memória distribuída, é importante que os nós acessem arquivos armazenados localmente, evitando transferências através da rede que conecta o nós.

Capítulo 5 - Conclusões

Esta dissertação apresentou as motivações, a pesquisa e o esforço envolvidos na adaptação do Chiron (OGASAWARA *et al*., 2013) para suportar a execução de *workflows* científicos em ambientes de processamento paralelo com memória distribuída. O processo de criação do ChironSN foi pautado por algumas metas, dentre elas: manutenção da simplicidade na configuração dos experimentos; utilização de metodologias e tecnologias que viabilizem modificações e expansões; e a adição de recursos que aumentem a quantidade de cenários onde o ChironSN pode ser utilizado. O fato de o ChironSN manter a compatibilidade com disco compartilhado é um reflexo de um cuidadoso processo de desenvolvimento de software.

A escolha pelo HDFS como sistema de arquivos distribuído provou-se um acerto durante a pesquisa e os experimentos. Além da alta disponibilidade e tolerância a falhas, a sua API permitiu que o conceito de localidade dos dados fosse aproveitado pelo ChironSN no momento de distribuição das ativações para execução.

Devido à falta de um ambiente de PAD onde um mesmo experimento pudesse ser executado com Chiron (disco compartilhado) e ChironSN (memória distribuída), não foi possível avaliar como o ChironSN se comportaria ao executar *workflows* já processados pelo Chiron.

Por ora, fica a constatação de que o ChironSN é uma contribuição aos sistemas que apoiam a execução paralela de *workflows* científicos. A migração para ambientes de memória distribuída aumenta a área de alcance do Chiron, atraindo novos cientistas e seus variados experimentos científicos.

Como futuras contribuições e expansões à pesquisa apresentada nesta dissertação, é interessante desenvolver um mecanismo para descentralizar a ordenação das tuplas na preparação das atividades de agregação. Ao passo que o volume de tuplas a serem agregadas aumenta, o tempo para transferir os dados ao nó principal, inserir no banco e efetuar a ordenação torna-se um ponto de contenção na execução dos experimentos.

Também é possível citar a necessidade de se avaliar o comportamento e desempenho do ChironSN na execução de experimentos previamente executados pelo Chiron, mas agora no ambiente de memória distribuída, com a utilização do HDFS.

Com relação aos novos recursos do ChironSN, há algumas modificações que ficaram pendentes. O parâmetro *numTuplesPerActivation* é útil quando se está executando o ChironSN sem utilizar referências a arquivos, quando todos os dados ficam salvos diretamente na base de proveniência. Mas esse parâmetro pode ter outro uso. O ChironSN ainda não tem a capacidade de fragmentar os arquivos de entrada; o cientista precisa efetuar a divisão manualmente e fornecer ao ChironSN um arquivo com as referências a esses fragmentos. O parâmetro *numTuplesPerActivation* pode ser adaptado e aplicado aos dados de entrada, de tal forma que o ChironSN leia um arquivo grande e vá criando fragmentos de acordo com o valor especificado pelo parâmetro. Esses fragmentos são salvos no disco, mas como é o ChironSN que está fazendo a divisão, ele já aproveitaria para salvar na base de proveniência as referências a esses arquivos criados, poupando o cientista do esforço de dividir os dados de entrada e indicar por escrito o nome de cada um desses arquivos.

Por fim, há a etapa de escrita das tuplas em disco, que foi feita de tal forma que a execução das ativações começa apenas quando todos os arquivos foram criados. Uma modificação desejável é que, conforme os arquivos com as tuplas sejam criados, a execução das respectivas ativações seja iniciada.

Referências Bibliográficas

ALTINTAS, I., BERKLEY, C., JAEGER, E., *et al.*, 2004, "Kepler: an extensible system for design and execution of scientific workflows". In: *Scientific and Statistical Database Management*, p. 423-424, Greece.

APACHE HADOOP, "The Apache™ Hadoop® Project". Website, 2014. Disponível em: <http://hadoop.apache.org/>.

APACHE MAHOUT, "Apache Mahout™: Scalable machine learning and data mining". Website, 2014. Disponível em: <https://mahout.apache.org/>.

APACHE NUTCH, "Apache Nutch™". Website, 2014. Disponível em: <http://nutch.apache.org/>.

BAEZA-YATES, R., RIBEIRO-NETO, B., 2013, *Recuperação de Informação: Conceitos e Tecnologia das Máquinas de Busca*, 2 ed. Bookman.

CALLAHAN, S. P., FREIRE, J., SANTOS, E., *et al.*, 2006, "VisTrails: visualization meets data management". In: *SIGMOD*, p. 745-747, Chicago, Illinois, USA.

CAPPELLO, F., CARON, E., DAYDE, M., *et al.*, 2005, "Grid'5000: a large scale and highly reconfigurable grid experimental testbed". In: *Proc. of the 6th IEEE/ACM International Workshop on Grid Computing (SC'05)*. IEEE/ACM, pp. 99-106.

CARPENTER, B., GETOV, V., JUDD, G., *et al.*, 2000, "MPJ: MPI-like message passing for Java", *Concurrency: Practice and Experience*, v. 12, n. 11, pp. 1019-1038.

DEAN, J., GHEMAWAT, S., 2008, "MapReduce: Simplified Data Processing On Large Clusters". In: *Communications of the ACM*, v. 51, pp. 107-113.

DEELMAN, E., GANNON, D., SHIELDS, M., *et al.*, 2009, "Workflows and e-Science: An overview of workflow system features and capabilities", *Future Generation Computer Systems*, v. 25, n. 5, pp. 528-540.

DEELMAN, E., MEHTA, G., SINGH, G., *et al.*, 2007, "Pegasus: Mapping Large-Scale Workflows to Distributed Resources", *Workflows for e-Science*, pp. 376-394.

DIAS, J., OGASAWARA, E., OLIVEIRA, D., *et al.*, 2013, "Algebraic dataflows for big data analysis". *BigData'2013: International Conference on Big Data, Santa Clara: États-Unis*, pp. 150-155.

GEIST, A., GROPP, W., HUSS-LEDERMAN, S., *et al.*, 1996, "MPI-2: Extending the message-passing interface". In: *Euro-Par'96 Parallel Processing*. Springer-Verlag, pp. 128-135.

GHEMAWAT, S., GOBIOFF, H., LEUNG, S.-T., 2003, "The Google file system". In: *ACM SIGOPS Operating Systems Review*. v. 37, pp. 29-43.

GIL, Y., DEELMAN, E., ELLISMAN, M., *et al.*, 2007, "Examining the Challenges of Scientific Workflows", *Computer*, v. 40, n. 12, p. 24-32.

GOLDMAN, A., KON, F., JUNIOR, F. P., *et al*., 2012, "Apache Hadoop: Conceitos teóricos e práticos, evolução e novas possibilidades". *XXXI Jornadas de Atualizações em Informática*.

GUERRA, G., ROCHINHA, F., ELIAS, R., *et al.*, 2012, "Uncertainty Quantification in Computational Predictive Models for Fluid Dynamics Using Workflow Management Engine". *International Journal for Uncertainty Quantification*, v. 2, n. 1, p. 53-71.

HOGAN, M, 2014. "Shared-Disk vs. Shared-Nothing". *White Paper*. Disponível em: <http://www.scaledb.com/pdfs/WP_SDvSN.pdf>. Acessado em 26/09/2014.

HUANG S., HUANG J., DAI J., *et al*., 2010, "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis". In: *Intl. Conference on Data Engineering Workshops (IEEE 2010)*, pp. 41-51.

LIN, J., 2012, "MapReduce is Good Enough? If All You Have is a Hammer, Throw Away Everything That's Not a Nail!". *arXiv preprint arXiv:1209.2191*.

MATTOSO, M. L. Q., DIAS, J., COSTA, F., *et al.*, 2014, "Experiences in using provenance to optimize the parallel execution of scientific workflows steered by users". In: *Workshop of Provenance Analytics*.

OCAÑA K., OLIVEIRA, D., DIAS, J., *et al.*, 2011, "Optimizing Phylogenetic Analysis Using SciHmm Cloud-based Scientific Workflow". In: *2011 IEEE Seventh International Conference on e-Science (e-Science)*, pp. 190-197.

OGASAWARA, E., DIAS, J., OLIVEIRA, D., *et al*., 2011, "An Algebraic Approach for Data-Centric Scientific Workflows". In: *Proceedings of the VLDB Endowment*, v. 4, n. 12, pp. 1328-1339.

OGASAWARA, E., 2011, *Uma Abordagem Algébrica para Workflows Científicos com Dados em Larga Escala*. Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

OGASAWARA, E., DIAS, J., SILVA, V., *et al*., 2013, "Chiron: A Parallel Engine For Algebraic Scientific Workflows", *Concurrency and Computation: Practice and Experience*, v. 25, n. 16, pp. 2327-2341.

OLIVEIRA, D., 2012, *Uma Abordagem de Apoio à Execução Paralela de Workflows Científicos em Nuvens de Computadores*, Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

OLIVEIRA, D., OGASAWARA, E., OCAÑA K., *et al.*, 2012, "An Adaptive Parallel Execution Strategy for Cloud-based Scientific Workflows". *Concurrency and Computation: Practice and Experience*, v. 24, pp. 1531-1550.

OLIVEIRA, D., OGASAWARA, E., SEABRA, F., *et al.*, 2010, "GExpLine: A Tool for Supporting Experiment Composition", *Provenance and Annotation of Data and Processes*, Springer Berlin / Heidelberg, pp. 251-259.

PAGE, L.; BRIN, S.; MOTWANI, R., *et al.*, 1998, "The PageRank Citation Ranking: Bringing Order to the Web". In: *Proceedings of the 7th International World Wide Web Conference*, pp. 161-172.

POSTGRESQL, "PostgreSQL: The world's most advanced open source database". Website, 2014. Disponível em: <http://www.postgresql.org/>

POSTGRESQL JDBC DRIVER, "PostgreSQL JDBC Driver". Website, 2014. Disponível em: <http://jdbc.postgresql.org/>

SCOTT, M., 2009, *Programming Language Pragmatics*. 3 ed. Morgan Kaufmann.

SHVACHKO, K., KUANG, H., RADIA, S., *et al.*, 2010, "The Hadoop Distributed File System". In: *Proceedings of Mass Storage Systems and Technologies (IEEE 26th Symposium)*, pp. 1-10.

TANENBAUM, A. S., 2007, *Modern Operating Systems*. 3 ed. Prentice Hall.

WHITE, T., 2012, *Hadoop: The Definitive Guide*. 3 ed. O'Reilly Media.

ZHAO, Y., HATEGAN, M., ClIFFORD, B., *et al.*, 2007, "Swift: Fast, Reliable, Loosely Coupled Parallel Computation". In: *3rd IEEE World Congress on Services*, pp. 199-206, Salt Lake City, USA.

Apêndice A

Os arquivos XML utilizados para descrever os *workflows* e os experimentos com o Chiron estão disponíveis a seguir.

Descrição conceitutal para o *workflow* de contagem de palavras:

```
<?xml version="1.0" standalone="no"?>
<Chiron>
    <database name="chiron" server="localhost" port="5432" username="chiron" password="chiron"/>
    <Workflow tag="wordcount">
        <Activity tag="act1" type="SPLIT_MAP"
                activation='hadoop jar ChironMapReduce.jar wordcount map "%=WORKDIR%"'>
            <Relation reltype="Input" name="wcIAct1"/>
            <Relation reltype="Output" name="wcOAct1"/>
            <Field name="LINE" type="text" input="wcIAct1" />
            <Field name="WORD" type="string" output="wcOAct1" />
            <Field name="COUNT" type="integer" output="wcOAct1" />
        </Activity>
        <Activity tag="act2" type="REDUCE" operand="WORD"
                activation='hadoop jar ChironMapReduce.jar wordcount reduce "%=WORKDIR%"'>
            <Relation reltype="Input" name="wcIAct2" dependency="act1"/>
            <Relation reltype="Output" name="wcOAct2" />
            <Field name="WORD" type="string" input="wcIAct2" output="wcOAct2" />
            <Field name="COUNT" type="integer" input="wcIAct2" output="wcOAct2" />
            </Activity>
    </Workflow>
</Chiron>
```

Configuração do experimento de contagem de palavras:

```
<?xml version="1.0" standalone="no"?>
<Chiron>
    <database name="chiron" server="localhost" port="5432" username="chiron" password="chiron"/>
    <Workflow tag="wordcount" execmodel="DYN_FAF" exectag="wordcount-1"
            wfdir="/home/jferreira/chiron/wordcount"
            expdir="/wordcount" hdfs="griffon-1.nancy.grid5000.fr:9000">
        <Activity tag="act1" numTuplesPerActivation="-1" />
        <Activity tag="act2" numTuplesPerActivation="-1" />
        <Relation name="wcIAct1" filename="/hibench/Wordcount/Input" />
    </Workflow>
</Chiron>
```

Descrição conceitual para o *workflow* de cálculo do *TF-IDF*:

```
<?xml version="1.0" standalone="no"?>
<Chiron>
    <database name="chiron" server="localhost" port="5432" username="chiron" password="chiron"/>
    <Workflow tag="tfidf">
        <Activity tag="act1" type="SPLIT_MAP"
                activation='hadoop jar ChironMapReduce.jar tfidf_1 map "%=WORKDIR%"'>
            <Relation reltype="Input" name="tfidfIAct1" />
            <Relation reltype="Output" name="tfidfOAct1" />
            <Field name="LINE"                  type="text"     input="tfidfIAct1" />
            <Field name="TERM"                  type="string"                      output="tfidfOAct1" />
            <Field name="DOC"                   type="string"                      output="tfidfOAct1" />
            <Field name="TERM_COUNT_IN_DOC"     type="integer"                     output="tfidfOAct1" />
            <Field name="NUM_TERMS_IN_DOC"      type="integer"                     output="tfidfOAct1" />
        </Activity>
        <Activity tag="act2" type="REDUCE" operand="TERM"
                activation='hadoop jar ChironMapReduce.jar tfidf_1 reduce "%=WORKDIR%"'>
            <Relation reltype="Input" name="tfidfIAct2" dependency="act1" />
            <Relation reltype="Output" name="tfidfOAct2" />
            <Field name="TERM"                  type="string"   input="tfidfIAct2"  output="tfidfOAct2" />
            <Field name="DOC"                   type="string"   input="tfidfIAct2"  output="tfidfOAct2" />
            <Field name="TERM_COUNT_IN_DOC"     type="integer"  input="tfidfIAct2"  output="tfidfOAct2" />
            <Field name="NUM_TERMS_IN_DOC"      type="integer"  input="tfidfIAct2"  output="tfidfOAct2" />
            <Field name="NUM_DIFFERENT_DOCS"    type="integer"                      output="tfidfOAct2" />
        </Activity>
        <!-- -->
        <Activity tag="act3" type="MAP"
                activation='hadoop jar ChironMapReduce.jar tfidf_2 map "%=WORKDIR%" 25000'>
            <Relation reltype="Input" name="tfidfIAct3" dependency="act2" />
            <Relation reltype="Output" name="tfidfOAct3" />
            <Field name="TERM"                  type="string"   input="tfidfIAct3"  output="tfidfOAct3" />
            <Field name="DOC"                   type="string"   input="tfidfIAct3"  output="tfidfOAct3" />
            <Field name="TERM_COUNT_IN_DOC"     type="integer"  input="tfidfIAct3" />
            <Field name="NUM_TERMS_IN_DOC"      type="integer"  input="tfidfIAct3" />
            <Field name="NUM_DIFFERENT_DOCS"    type="integer"  input="tfidfIAct3" />
            <Field name="TFIDF"                 type="float"                       output="tfidfOAct3" />
        </Activity>
    </Workflow>
</Chiron>
```

Configuração do experimento de cálculo do *TF-IDF*:

```
<?xml version="1.0" standalone="no"?>
<Chiron>
    <database name="chiron" server="localhost" port="5432" username="chiron" password="chiron"/>
    <Workflow tag="tfidf" execmodel="DYN_FAF" exectag="tfidf-1"
            wfdir="/home/jferreira/chiron/tfidf"
            expdir="/tfidf" hdfs="griffon-1.nancy.grid5000.fr:9000">
        <Activity tag="act1" numTuplesPerActivation="-1" />
        <Activity tag="act2" numTuplesPerActivation="-1" />
        <Activity tag="act3" numTuplesPerActivation="-1" />
        <Relation name="tfidfIAct1" filename="/tfidf/input" />
    </Workflow>
</Chiron>
```

Apêndice B

Descrição de quatro programas Map/Reduce que são executados de forma encadeada para calcular o *TF-IDF* em uma coleção de documentos.

1. Contagem do número de ocorrências de cada termo nos documentos (*n*)
   - Mapeamento:
     - Entrada: *(NomeDoc, Conteudo)*
     - Saída: *lista((Termo,NomeDoc), 1)*
   - Redução (e Combinação)
     - Entrada: *lista((Termo,NomeDoc), lista(n_1, n_2, ..., n_k))*
     - Saída: *lista((Termo,NomeDoc), n)*
2. Contagem do número total de termos em cada um dos documentos (*N*)
   - Mapeamento:
     - Entrada: *((Termo,NomeDoc), n)*
     - Saída: *(NomeDoc, (Termo, n))*
   - Redução
     - Entrada: *lista(NomeDoc, lista((Termo$_1$,n_1), ... , (Termo$_k$, n_k)))*
     - Saída: *lista((Termo,NomeDoc), (n, N))*
3. Contagem do número de documentos em que cada termo aparece (*m*)
   - Mapeamento:
     - Entrada: *((Termo,NomeDoc), (n, N))*
     - Saída: *(Termo, (NomeDoc, n, N, 1))*
   - Redução
     - Entrada: *lista(Termo, lista((NomeDoc$_1$, n_1, N_1, 1), ... , (NomeDoc$_k$, n_k, N_k, 1)))*
     - Saída: *lista((Termo,NomeDoc), (n, N, m))*

4. Cálculo do *TF-IDF*
    - Assume-se que total de documentos (*D*) é conhecido
    - Funções de cálculo do *TF* e do *IDF* foram omitidas.
    - Para cada *(Termo,NomeDoc)*, o cálculo utiliza *D* e os respectivos *(n, N, m)*
    - Mapeamento:
        - Entrada: *((Termo,NomeDoc), (n, N, m))*
        - Saída: *((Termo,NomeDoc), TF*IDF)*
    - Redução
        - Não é necessário;
        - Pode ser usado para agrupar o *TF-IDF* por documentos, ao invés do agrupamento por termos.

Apêndice C

Tabelas com os resultados experimentais.

Tempos de execução do HiBench (em segundos):

	Nº da execução	Número de nós			
		32	16	8	4
Contagem de Palavras	1	24,75	30,739	31,838	46,785
	2	24,768	30,727	31,779	46,78
	3	24,757	30,733	30,766	47,767
	4	24,848	30,733	31,761	47,821

	Nº da execução	Número de nós			
		32	16	8	4
TF-IDF	1	152,561	140,473	145,045	146,779
	2	149,985	142,323	141,835	147,086
	3	151,307	143,053	141,809	148,032
	4	152,457	141,433	141,459	149,397

As cinco próximas páginas apresentam os resultados experimentais obtidos com a execução do ChironSN. Todos os tempos estão em milissegundos.

Workflow de Contagem de Palavras – Sem Localidade

# de nós	Nº da execução	# de tuplas por ativação de SPLIT_MAP	# de tuplas por ativação de REDUCE	Inserção dos dados	SPLIT_MAP				REDUCE					Total de execução das atividades
					Criação das ativações	# de ativações	Escrita das tuplas no disco	Execução	Propagação dos dados	Criação das ativações	# de ativações	Escrita das tuplas no disco	Execução	
32	1	51186	1000	117331	7270	32	38591	13507	731	134	32	3150	7183	70690
32	2	51186	1000	114635	7795	32	33560	12929	761	158	32	5834	4766	65937
32	3	51186	1000	116995	7363	32	41435	10652	783	160	32	862	7381	68694
16	1	102372	1000	119419	7263	16	33494	8919	313	109	16	490	8244	58931
16	2	102372	1000	113449	7296	16	35838	16349	566	167	16	468	3615	64506
16	3	102372	1000	115767	7321	16	35363	10134	344	110	16	487	4705	58521
8	1	204744	1000	129419	7287	8	38757	20233	374	267	8	2566	9143	78795
8	2	204744	1000	129416	7579	8	39349	16979	432	139	8	260	1991	66907
8	3	204744	1000	124492	12112	8	33047	20359	518	476	8	1380	4762	73171
4	1	409487	1000	118449	7457	4	36689	26424	77	43	4	4904	4113	79758
4	2	409487	1000	120728	7278	4	36621	21730	258	179	4	1037	1706	68939
4	3	409487	1000	117915	7463	4	33953	27661	78	108	4	143	1808	72830

Workflow de Contagem de Palavras – Com Localidade

# de nós	Nº da execução	# de tuplas por ativação de SPLIT_MAP	# de tuplas por ativação de REDUCE	Inserção dos dados	SPLIT_MAP				REDUCE					Total de execução das atividades
					Criação das ativações	# de ativações	Escrita das tuplas no disco	Execução	Propagação dos dados	Criação das ativações	# de ativações	Escrita das tuplas no disco	Execução	
32	1	51186	1000	109408	8281	32	34013	8607	988	354	32	5616	4410	62358
32	2	51186	1000	110188	7279	32	38789	8846	1064	414	32	2533	5193	64157
32	3	51186	1000	109108	7320	32	36647	9026	848	339	32	2025	6275	62521
16	1	102372	1000	116853	7321	16	37339	9255	325	99	16	3817	2451	60648
16	2	102372	1000	113409	7396	16	34614	14286	726	361	16	4150	3209	64972
16	3	102372	1000	113926	7304	16	35073	11703	333	109	16	4100	2542	61204
8	1	204744	1000	139060	7303	8	37919	14966	275	375	8	2217	1941	65089
8	2	204744	1000	122202	7278	8	36602	14467	139	76	8	375	1700	60671
8	3	204744	1000	121833	10195	8	34911	12827	531	334	8	5574	2908	67423
4	1	409487	1000	126931	7312	4	35898	18048	72	58	4	1491	1518	64441
4	2	409487	1000	114751	7287	4	34381	18918	77	51	4	233	2268	63266
4	3	409487	1000	115225	7321	4	35348	18376	80	58	4	208	1809	63235

Workflow de *TF-IDF* – Sem Localidade

# de nós	Nº da execução	# de tuplas por ativação de SPLIT_MAP	# de tuplas por ativação de REDUCE	# de tuplas por ativação de MAP	Inserção dos dados	SPLIT_MAP				REDUCE					MAP					Total de execução das atividades
						Criação das ativações	# de ativações	Escrita das tuplas no disco	Execução	Propagação dos dados	Criação das ativações	# de ativações	Escrita das tuplas no disco	Execução	Propagação dos dados	Criação das ativações	# de ativações	Escrita das tuplas no disco	Execução	
32	1	782	240050	1	11410	328	32	3794	414732	261773	20075	32	47937	6099	16	51	32	0	6284	761149
32	2	782	240050	1	8674	300	32	4939	408029	265068	19893	32	48350	4710	8	51	32	0	6917	758333
32	3	782	240050	1	9339	386	32	3858	414150	263114	19908	32	49286	5598	17	50	32	0	6276	762703
16	1	1563	480099	1	11195	302	16	3236	418433	257751	20208	16	53258	4681	8	59	16	0	7166	765171
16	2	1563	480099	1	9046	221	16	3203	417732	264632	20393	16	38231	4879	17	42	16	0	5317	754723
16	3	1563	480099	1	9316	213	16	3793	418068	257662	20256	16	51684	4548	17	58	16	0	6425	762794
8	1	3125	960197	1	11636	313	8	2968	437420	273548	20702	8	38239	5474	8	50	8	0	10559	789341
8	2	3125	960197	1	9355	213	8	4187	451984	268480	20116	8	38582	5792	8	42	8	0	7316	796788
8	3	3125	960197	1	9112	205	8	3811	441483	284433	20116	8	37617	5780	8	42	8	0	8783	802347
4	1	6250	1920393	1	11239	346	4	2847	446301	264890	21493	4	54391	9230	8	25	4	0	11100	810697
4	2	6250	1920393	1	9197	196	4	3173	453560	268954	21359	4	40423	10770	8	33	4	0	11616	810161
4	3	6250	1920393	1	9724	196	4	3702	442996	267292	21050	4	54053	9410	8	34	4	0	13716	812525

Workflow de *TF-IDF* – Com Localidade

# de nós	Nº da execução	# de tuplas por ativação de SPLIT_MAP	# de tuplas por ativação de REDUCE	# de tuplas por ativação de MAP	Inserção dos dados	SPLIT_MAP				REDUCE					MAP					Total de execução das atividades
						Criação das ativações	# de ativações	Escrita das tuplas no disco	Execução	Propagação dos dados	Criação das ativações	# de ativações	Escrita das tuplas no disco	Execução	Propagação dos dados	Criação das ativações	# de ativações	Escrita das tuplas no disco	Execução	
32	1	782	240050	1	9167	376	32	5214	402065	250470	19865	32	37298	4855	8	84	32	0	6409	726695
32	2	782	240050	1	8839	229	32	4241	400835	252270	20215	32	47203	12156	8	92	32	0	5451	742769
32	3	782	240050	1	8006	390	32	5833	403524	251721	20133	32	47897	5412	8	84	32	0	5550	740603
16	1	1563	480099	1	10405	346	16	3508	421601	261286	20536	16	39030	4712	8	58	16	0	8077	759215
16	2	1563	480099	1	9140	204	16	4508	425991	263323	20041	16	52055	4620	16	59	16	0	5234	776110
16	3	1563	480099	1	8965	221	16	3458	420505	261910	20390	16	38239	4502	16	50	16	0	5592	754928
8	1	3125	960197	1	11804	230	8	3149	437828	277875	20559	8	51575	5501	16	50	8	0	7468	804293
8	2	3125	960197	1	10051	205	8	3193	445440	273259	20657	8	51411	5683	17	50	8	0	7341	807299
8	3	3125	960197	1	9482	273	8	3318	449554	274708	19981	8	51143	5935	8	51	8	0	7266	812296
4	1	6250	1920393	1	11125	338	4	3019	439053	268172	21256	4	53584	8708	8	134	4	0	10999	805331
4	2	6250	1920393	1	9240	205	4	2941	443982	268000	21363	4	40673	9494	16	42	4	0	11202	797961
4	3	6250	1920393	1	9490	235	4	2957	435543	264740	21424	4	41929	9425	8	42	4	0	11108	787461

Contagem de Palavras - Tempo médio de execução das ativações					
# de nós	Nº da execução	Sem Localidade		Com Localidade	
		SPLIT_MAP	REDUCE	SPLIT_MAP	REDUCE
4	1	17,385	1,217	16,939	1,258
4	2	17,498	1,682	17,476	1,731
4	3	17,146	1,536	17,114	1,333
8	1	13,727	6,151	10,548	1,243
8	2	10,504	1,278	10,418	1,248
8	3	13,004	1,668	10,025	1,403
16	1	6,355	2,588	7,114	1,268
16	2	6,803	1,294	6,334	1,327
16	3	6,441	1,862	6,547	1,317
32	1	4,512	1,264	4,900	1,310
32	2	4,701	1,240	4,259	1,252
32	3	4,893	1,241	4,263	1,244

TF-IDF - Tempo médio de execução das ativações							
# de nós	Nº da execução	Sem Localidade			Com Localidade		
		SPLIT_MAP	REDUCE	MAP	SPLIT_MAP	REDUCE	MAP
4	1	7,828	8,395	8,511	8,248	7,965	8,814
4	2	7,697	9,815	9,290	7,696	8,655	8,633
4	3	7,669	8,322	9,364	7,742	8,932	8,584
8	1	5,087	4,905	6,194	5,059	4,816	5,435
8	2	5,061	4,892	5,484	4,994	4,846	5,271
8	3	5,099	4,943	5,582	5,024	4,891	5,576
16	1	3,893	3,214	3,737	3,619	3,202	4,020
16	2	3,682	3,247	3,627	3,593	3,282	3,564
16	3	3,712	3,225	3,605	3,599	3,207	3,568
32	1	2,906	2,393	2,940	2,893	2,372	2,669
32	2	2,962	2,528	3,079	2,877	4,103	2,725
32	3	2,886	2,389	2,661	2,871	2,417	2,937