Análise comparativa de ambientes e linguagens para computação intensiva de dados na nuvem

Robespierre Dantas[1], **Marcos Barreto**[1]

[1]LaSiD, IM, DCC, UFBA
CEP 40.170-110 – Salvador – BA – Brazil

{pierre.pita,marcoseb}@gmail.com

Abstract. *This paper presents a preliminary effort to study and compare the environments, frameworks and programming models more referenced in the study of art dedicated to the processing of data-intensive applications.*

Resumo. *Este trabalho apresenta um esforço preliminar de estudo e comparação dos ambientes, frameworks e modelos de programação mais referenciados no estudo da arte dedicados ao processamento de aplicações data-intensive.*

1. Introdução

Este artigo apresenta uma análise comparativa entre ambientes e linguagens para o processamento de aplicações intensivas de dados, desde a proposta precursora do Google - MapReduce [1], passando pelo esforço da *Yahoo!* em disponibilizá-lo em código aberto [5], sua versão otimizada [2] e a proposta da Microsoft para aplicações imperativas e de propósito geral [6].

Itens como tipo de dados de entrada, manipulação de dados intermediários, comunicação entre processos, tolerância à falhas e desempenho (vazão de dados processados) serão levados em consideração na comparação. O entendimento sobre os pontos fortes e fracos de cada ambiente apresentado neste trabalho será de grande valia para permitir a criação de um módulo de suporte à linguagens e modelos de programação numa infraestrutura baseada em *cloud computing* destinada a atender aplicações de *data-intensive*.

2. MapReduce

Inspirado no *Modelo de Programação Funcional*, o MapReduce [1] foi proposto pela Google para processar, de forma distribuída, uma grande massa de dados. O uso deste *framework* consiste na implementação, pelo desenvolvedor, de duas funções:

- ***map***: Função responsável por percorrer e processar toda a origem de dados brutos para classificá-los e ordená-los em pares de chaves de interesse do desenvolvedor;
- ***reduce***: Função que agrupa todos os dados intermediários provenientes da etapa '*map*' em grupos com valores de chaves iguais.

Uma característica do *MapReduce* é prover um alto nível de abstração ao desenvolvedor, extigüindo a necessidade deste de dominar ou sequer ter experiência com processamento paralelo e sistemas distribuídos. A Figura 1 mostra a arquitetura proposta pela Google para executar o MapReduce.

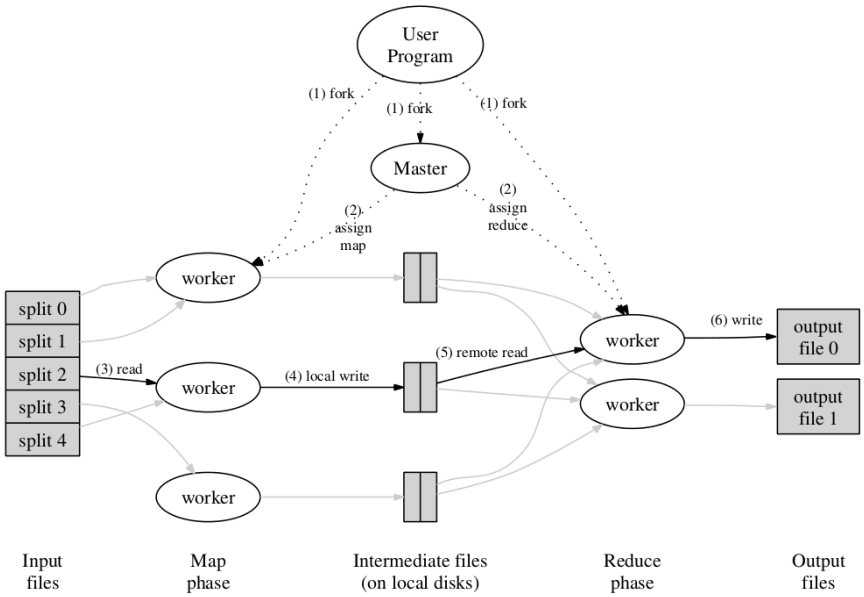

Figura 1. Arquitetura de execução do *MapReduce* [1]

A implementação segue os seguintes passos: O dado bruto de entrada é dividido em pedaços de 16 a 64 MB. Uma máquina ou *thread* (chamada *worker*) é eleita Master da execução, ela terá o papel de atribuir tarefas **map** ou **reduce** a outros *workers* e lidar com a comunicação entre eles. Os *workers map* utilizarão a divisão do dado bruto (*split*) a ele atribuída pelo Master para efetuar seu processamento e guardarão os dados intermediários em memória, que serão transferidos periódicamente para o disco. Os *workers reduce* são avisados pelo *Master* da existência de dados intermediários nas partições intermediárias e os acessam para processá-los remotamente, via RPC. Terminado o processamento *reduce dos workers*, o resultado é escrito em arquivos de saída e o programa de usuário é alertado da finalização e localidade dos resultados.

As falhas ligadas ao *Master* são contornadas a partir de *checkpoints* que este envia a outros *workers*. Caso ele falhe, outro *worker* passa a gerenciar a execução, assumindo ela a partir do *checkpoint* mais recente. Um *downtime* de um *worker* qualquer é percebido pelo *Master* através de *pings* constantes, em falta de resposta, a tarefa atribuída à este nó falho é delegada a outro ocioso ou com menos carga.

Levando em consideração a escassez de recursos de largura de banda e como uma forma de abstrair a necessidade de tolerância a falhas da origem dos dados brutos, o *MapReduce* considera que os dados de entrada a serem processados são provenientes de discos locais à rede, gerenciados pelo *GFS* [3].

3. Apache Hadoop

O objetivo do *Hadoop* é prover processamento paralelo em grandes conjuntos de máquinas em *cluster* de forma confiável, escalável, distribuída e com todos os benefícios de um software *open-source*.

Assim como no MapReduce [1], o *Apache Hadoop* é executado e divide o seu trabalho em tarefas, que podem ser de dois tipos, ***map*** e ***reduce***. Os nomes do nós, porém são diferentes, existe um *jobtracker* e vários *tasktrackers*. O primeiro é o responsável por

coordenar a execução através de escalonamento de tarefas, já os demais executam o que lhe foi delegado e enviam relatórios de progresso ao *jobtracker*.

O coordenador da execução registra e utiliza os progressos passados pelos *task-trackers* para re-escalonar a tarefa a outro nó em caso de falha nesses nós. Esse registro é utilizado por outro nó, caso o *jobtracker* falhe. Processo bastante análogo ao *MapReduce* da proposto pela Google com a diferença de que não há checkpoints em intervalos de tempos, e sim o registro do progressão da execução.

O Hadoop utiliza o conceito de *otimização de alocação de dados*, que ajude o *jobtracker* a atribuir uma tarefa map a um *tasktraker* que possa utilizar os dados de entrada do HDFS [4].

4. Microsoft DryadLINQ

O DryadLINQ [6] apresenta um modelo de programação que consegue compilar uma aplicação imperativa desenvolvida em linguagem de propósito geral num cluster robusto de forma distribuída e transparente, provendo para o programador a ilusão de estar escrevendo para um simples computador (sem se preocupar com escalonamento, distribuição ou tolerância a falhas).

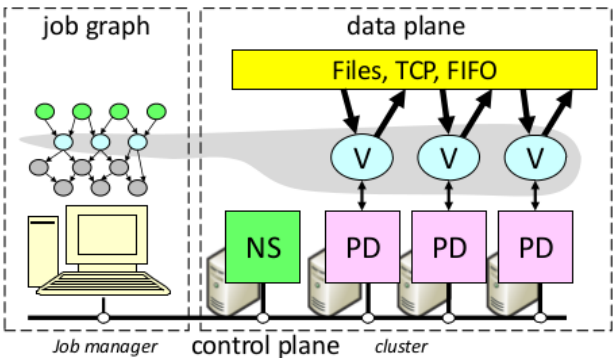

Figura 2. Arquitetura do DryadLINQ

Usando príncipios do *Dryad*, que propôs a possibilidade de se executar aplicações de forma distribuída através da sua modelagem em dígrafos acíclicos (DAG) e no *LINQ* (*Language Integrated Query*), suíte de funcionalidades de consulta à diversas fontes de dados em linguagens .NET, o DryadLINQ utiliza a arquitetura apresentada na Figura 2 que mostra a estrutura deste *framework*. O *Job Manager* é responsável por distribuir os vértices (V) em computadores ociosos com ajuda dos monitores remotos (PD). Os NS são os servidores de nome que cuidam da adesão de computadores ao cluster. A Tolerância à falhas do *DryadLINQ* é provida pelo *Job Manager* que re-executa processos lentos ou falhos.

5. Twister: Iterative MapReduce

O Twister, segundo [2], se apresenta como um modelo de programação e propõe uma melhora ao tempo de execução do proposto pela Google [1], o Iterative MapReduce.

Abolindo as práticas de particionamento de dados intermediários e discos locais usados nas demais soluções baseadas em MapReduce, o Twister aposta numa abordagem

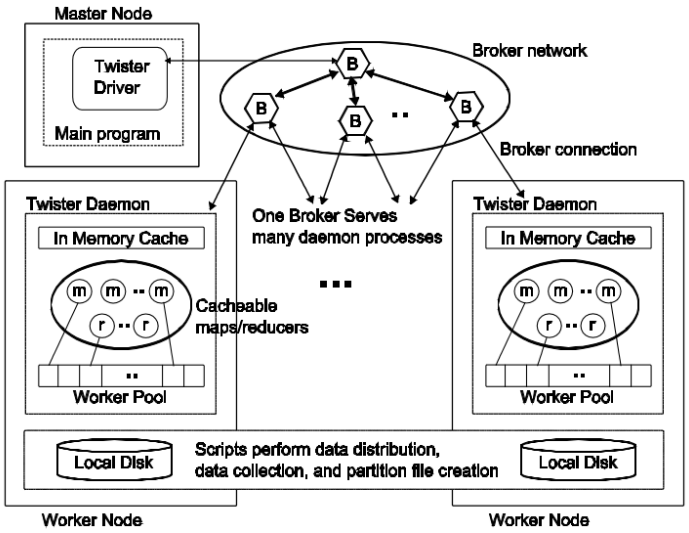

Figura 3. Arquitetura de execução do Twister [2]

in-memory. Outra estratégia de mudança no *runtime* original proposto por [1] é na comunicação de processos: o Iterative MapReduce implementado no Twister utiliza a arquitetura *publish/subscribe*, evitando assim o *overhead* causado por mensagens.

A Figura 3 apresenta a arquitetura geral do Twister, na qual é possível verificar a existência de três entidades principais: o Twister Driver, responsável por toda computação MapReduce; o Twister Daemon que é executado em todos os nós (capazes de atender tarefas *map* e *reduce* numa mesma execução) e a Broker Network, rede sobreposta que representa a comunicação entre os nós.

No quesito tolerância a falhas, a técnica utilizada pelo Twister é guardar o estado da aplicação entre as iterações para que seja possível retornar a alguma interação anterior em caso de falhas. Dessa forma, não são tratadas as falhas nos nós Twister Driver ou Daemon para que não se perca desempenho ao utilizar as mesmas técnicas que MapReduce.

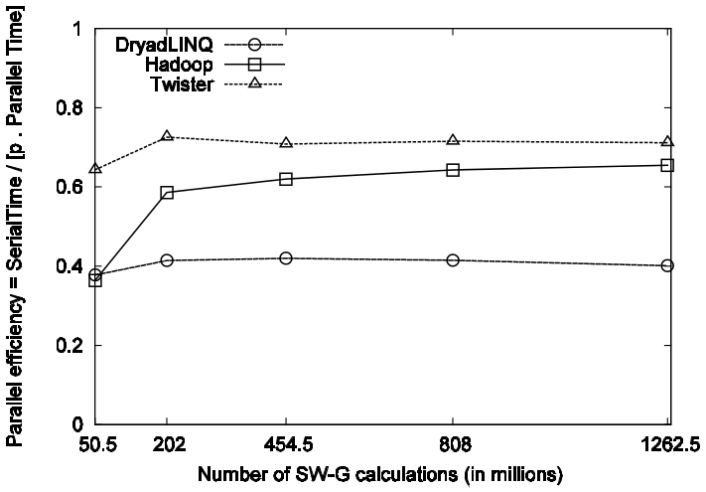

Figura 4. Resultado das medições feitas por [2] para avaliar o desempenho no processamento paralelo alcançado pelos ambientes.

6. Conclusões e Trabalhos Futuros

A Tabela 1 resume a análise comparativa feita neste artigo, levando em consideração as técnicas e conceitos mais importantes das soluções apresentadas. É possível perceber que os métodos implementados e propostos pelo Twister se destacam em muitos aspectos, principalmente no seu desempenho relativo ilustrado na Figura 4.

Devido às grandes contribuições do MapReduce [1] em termos de provimento de abstração, escalabilidade, distribuição e processamento paralelo de aplicações *data-intensive*, todas as maiores contribuições nessa área o tomaram como ponto de partida, seja para proposta de melhoria, extensão e/ou abertura de código fonte.

	Entrada de Dados	**Dados Intermediários**	**Comunicação entre Processos**	**Tolerância à Falhas**
MapReduce	GFS	Disco local	Master/Worker	Checkpoints e Heartbeat
Hadoop	HDFS	Disco local	JobTracker/ TaskTracker	Registro de execução
Twister	Discos Distribuídos	Em memória	Publish/ Subscribe	Estado da aplicação
DryaLINQ	DryadTable	Disco local	JobManager/ Cluster	Heartbeat

Tabela 1. Principais características dos ambientes estudados

O estudo comparativo apresentado neste artigo servirá de base para a concepção de um módulo de suporte ao desenvolvimento de aplicações intensivas de dados em diferentes plataformas. Este módulo será parte de uma infraestrutura de computação em nuvem cujo objetivo será executar aplicações *data-intensive*.

Referências

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[2] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 810–818, New York, NY, USA, 2010. ACM.

[3] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, October 2003.

[4] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[5] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

[6] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. Dryadlinq: a system for general-purpose distributed data-parallel computing using a high-level language. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 1–14, Berkeley, CA, USA, 2008. USENIX Association.