Centro Federal de Educação Tecnológica de Minas Gerais
Departamento de Computação
Engenharia de Computação

Mariane Raquel Silva Gonçalves

Comparação de algoritmos paralelos de ordenação em MapReduce

Orientadora: Prof[a]. Dr[a]. Cristina Duarte Murta

Belo Horizonte
2012

Sumário

1 Introdução

Fazer aqui uma introdução geral da área do conhecimento à qual o tema escolhido está ligado.

1.1 Definição do Problema

A ordenação é um dos problemas fundamentais da ciência da computação e um dos problemas algorítmicos mais estudados. Muitas aplicações dependem de ordenações eficientes como base para seu próprio desempenho. A ordenação por si só é um problema que abrange desde sistemas de banco de dados à computação gráfica, e muitos outros algoritmos podem ser descritos em termos de ordenação [Satish et al. 2009, Amato et al. 1996].

Na última década, a quantidade de dados (de trabalho, utilizada pelos sistemas, disponíveis) aumentou várias ordens de grandeza, fazendo o processamento dos dados um desafio para a computação sequencial. Como resultado, torna-se crucial substituir a computação tradicional por computação distribuída eficiente [Lin e Dyer 2010]. A mudança no modelo de programação sequencial para paralelo é um fato inevitável e ocorre gradualmente, desde que a indústria declarou que seu futuro está em computação paralela, com o aumento do número de núcleos dos processadores [Asanovic et al. 2009].

Uso crescente de computação paralela em sistemas computacionais gera a necessidade de algoritmos de ordenação inovadores, desenvolvidos para dar suporte a essas aplicações. Isso significa desenvolver rotinas eficientes de ordenação em arquiteturas paralelas e distribuídas.

O MapReduce é um modelo de programação paralela desenvolvido pela Google para processamento de grandes volumes de dados distribuídos em *clusters* [Dean e Ghemawat 2008]. Esse modelo propõe simplificar a computação paralela, escondendo detalhes da paralelização do desenvolvedor e utilizando duas funções principais - map e reduce. Uma das implementações mais conhecidas e utilizadas

do modelo é o Hadoop [White 2009], ferramenta de código aberto, desenvolvida por Doug Cutting em 2005 e apoiada pela Yahoo!.

O trabalho proposto por Pinhão (2011) apresentou uma avaliação da escalabilidade de algoritmos de ordenação paralela no modelo MapReduce. Para tal, foi desenvolvido no ambiente Hadoop o algoritmo de Ordenação por Amostragem, e seu desempenho foi avaliado em relação à quantidade de dados de entrada e ao número de máquinas utilizadas.

Considerando esse contexto, o presente trabalho segue este tema e busca continuar a análise, com a implementação do algoritmo Quick Sort no mesmo ambiente, análise de escalabilidade e comparação do desempenho dos dois algoritmos.

1.2 Motivação

O volume de dados que é produzido e manipulado (outra palavra?) em indústrias, empresas e até mesmo em âmbito pessoal aumenta a cada ano. O desenvolvimento de soluções capazes de lidar com tais volumes de dados é uma das preocupações atuais, tendo em vista a quantidade de dados processados diariamente, e o rápido crescimento desse volume de dados. Não é fácil medir o volume total de dados armazenados digitamente, mas uma estimativa da IDC [Gantz 2008] colocou o tamanho do "universo digital"em 0,18 zettabytes em 2006, e previa um crescimento dez vezes até 2011 (chegando a 1,8 zettabytes). *The New York Stock Exchange* gera cerca de um terabyte de novos dados comerciais por dia. O Facebook armazena aproximadamente 10 bilhões de fotos, que ocupam mais de um petabyte. *The Internet Archive* armazena aproximadamente 2 petabytes de dados, com aumento de 20 terabytes por mês [White 2009]. Estima-se que dados não estruturados são a maior porção e de a mais rápido crescimento dentro das empresas, o que torna o processamento de tal volume de dados muitas vezes inviável.

Mesmo para os computadores atuais, é um desafio conseguir lidar com quantidades de dados tão grandes. É preciso buscar soluções escaláveis, que apresentem bom desempenho em tais condições. As tendências atuais em *design* de microprocessadores estão mudando fundamentalmente a maneira que se obtém desempenho de sistemas computacionais.

Nos últimos 40 anos, o aumento no poder computacional deu-se, largamente, ao aumento na capacidade do hardware. O fator principal dessa melhoria foi a capacidade de dobrar, a cada dois anos, o número de dispositivos microeletrônicos em uma mesma área de silício, a um custo quase constante. Esse crescimento exponencial no número de transistores presentes nos processadores é conhecida como Lei de Moore [Manferdelli et al. 2008]. No entanto, o aumento no número de transistores, e consequente aumento na velocidade do processador foi limitado por questões físicas, como a dissipação do calor.

Há alguns anos a indústria percebeu tal fato, e declarou que seu futuro está em computação paralela, com o aumento crescente do número de núcleos dos processadores [Asanovic et al. 2009]. Atualmente, arquitetos sabem que o aumento no desempenho só pode ser alcançado com o uso de computação paralela, e têm recorrido cada vez mais a arquiteturas paralelas para continuar a fazer progressos [Manferdelli et al. 2008].

Com os novos modelos de processadores multicore, o modelo de programação single core está sendo substituído rapidamente pelo novo modelo, e com isso surge a necessidade de escrever software para sistemas com multiprocessadores e memória compartilhada [Ernst et al. 2009]. Arquiteturas multi-core podem oferecer um aumento significativo de desempenho sobre as arquiteturas de núcleo único, de acordo com as tarefas paralelizadas. No entanto, muitas vezes isto exige novos paradigmas de programação para utilizar eficientemente a arquitetura envolvida [Prinslow 2011].

A técnicas tradicionais de programação paralelas - como passagem de mensagens e memória compartilhada, em geral são complexas e de difícil entendimento para grande parte dos desenvolvedores. Em tais modelos, é preciso gerenciar localidades temporais e espaciais e lidar explicitamente com concorrência, criando e sincronizando *threads* através de mensagens e *semáforos*. Dessa forma, não é uma uma tarefa simples escrever códigos paralelos corretos e escaláveis para algoritmos não triviais [Ranger et al. 2007].

O MapReduce surgiu como uma alternativa aos modelos tradicionais, com o objetivo de simplificar a computação paralela. O maior benefício desse modelo é a simplicidade. O foco do programador é a descrição funcional do algoritmo, e não as formas de paralelização. Nos últimos anos o modelo têm se estabelecido

como uma das plataformas de computação paralela mais amplamente utilizadas no processamento de terabyte e petabyte de dados [Ranger et al. 2007]. MapReduce e sua implementação *open source* Hadoop oferecem uma alternativa economicamente atraente através de uma plataforma eficiente de computação distribuída, capaz de lidar com grandes volumes de dados e mineração de petabytes de informações não estruturadas [Cherkasova 2011].

// texto conector

A ordenação é um dos problemas fundamentais da ciência da computação e algoritmos paralelos para ordenação têm sido estudados desde o início da computação paralela. Os algoritmos ótimos existentes em arquitetura sequencial, como Quick Sort e Heap Sort necessitam de um tempo mínimo *(n log n)* para ordenar uma sequência de n elementos [Aho et al. 1974].

Na ordenação paralela, fatores como movimentação de dados, balanço de carga, latência de comunicação e distribuição inicial das chaves são considerados ingredientes chave para o bom desempenho, e variam de acordo com o algoritmo escolhido como solução [Kale e Solomonik 2010].

Dado o grande número de algoritmos de ordenação paralela e grande variedade de arquiteturas paralelas, é uma tarefa difícil escolher o melhor algoritmo para uma determinada máquina e instância do problema. Além disso, não existe um modelo teórico conhecido que pode ser aplicado para prever com precisão o desempenho de um algoritmo em arquiteturas diferentes [Amato et al. 1996].

Assim, estudos experimentais assumem uma crescente importância para a avaliação e seleção de algoritmos apropriados para multiprocessadores. É preciso que mais estudos sejam realizados para que determinado algoritmo pode ser recomendado em certa arquitetura com alto grau de confiança.

1.3 Objetivos

Os objetivos deste trabalho são:

- Estudar a programação paralela aplicada à algoritmos de ordenação;
- Implementar um ou mais algoritmos de ordenação paralela no modelo MapRe-

duce, com o software Hadoop;

- Comparar duas ou mais implementações de algoritmos paralelos de ordenação.

O trabalho desenvolvido por Pinhão (2011) apresentou um estudo sobre a computação paralela e algoritmos de ordenação no modelo MapReduce, através da implementação do algoritmo de Ordenação por Amostragem feita em ambiente Hadoop.

Este projeto busca continuar o estudo sobre ordenação paralela feito no trabalho citado, com a análise de desempenho dos algoritmos de ordenação ordenação por amostragem e quick sort. A análise busca compará-los com relação à quantidade de dados a serem ordenados, variabilidade dos dados de entrada e número máquinas utilizadas.

1.4 Organização do Texto

Esse projeto está organizado em 5 capítulos. O próximo capítulo apresenta o referencial teórico para o desenvolvimento do trabalho. O Capítulo 3 descreve a metologia de pesquisa, indicando os passos a serem seguidos durante o desenvolvimento. Os resultados preliminares obtidos até a entrega do projeto são apresentados no Capítulo 4. As conclusões obtidas até o momento e os próximos passos para a conclusão do projeto estão no Capítulo 5.

2 Referencial Teórico

Nesse capítulo serão apresentados os conceitos gerais do tema e o estado da arte.

2.1 Computação Pararela

modelos de computação paralela (memória compartilhada, distribuida, threads, paralelismo de dados e map reduce)

2.2 MapReduce

O MapReduce é um modelo de programação paralela criado pela Google para processamento de grandes volumes de dados em *clusters*. Esse modelo propõe simplificar a computação paralela e ser de fácil uso, abstraindo conceitos complexos da paralelização - como tolerância a falhas, distribuição de dados e balanço de carga - e utilizando duas funções principais: map e reduce. A complexidade do algoritmo paralelo não é vista pelo desenvolvedor, que pode se ocupar em desenvolver a solução proposta [Dean e Ghemawat 2008].

Esse modelo de programação é inspirado em linguagens funcionais, tendo como base as primitivas map e reduce. Os dados de entrada são específicos para cada aplicação, e descritos pelo usuário. A saída é um conjunto de pares no formato <chave, valor>. A função map é aplicada aos dados de entrada e produz uma lista intermediária de pares <chave, valor>. Todos os valores intermediários associados a uma mesma chave são agrupados e enviados à função reduce. A função reduce é então aplicada para todos os pares intermediários com a mesma chave. A função combina esses valores para formar um conjunto menor de resultados. Tipicamente, há apenas zero ou um valores de saída em cada função reduce.

Visão geral do fluxo de execução As chamadas da função map são distribuídas automaticamente entre as diversas máquinas através do particionamento dos dados de entrada em M conjuntos. Cada conjunto pode ser processado em paralelo por diferentes máquinas. As chamadas reduce são distribuídas através do particionamento do conjunto intermediário de pares em R partes. O número de partições R pode ser definido pelo usuário.

A Figura 1 apresenta o fluxo de uma execução do MapReduce. A sequência de ações descrita a seguir explica o que ocorre em cada um dos passos. A numeração dos itens a seguir corresponde à numeração da figura.

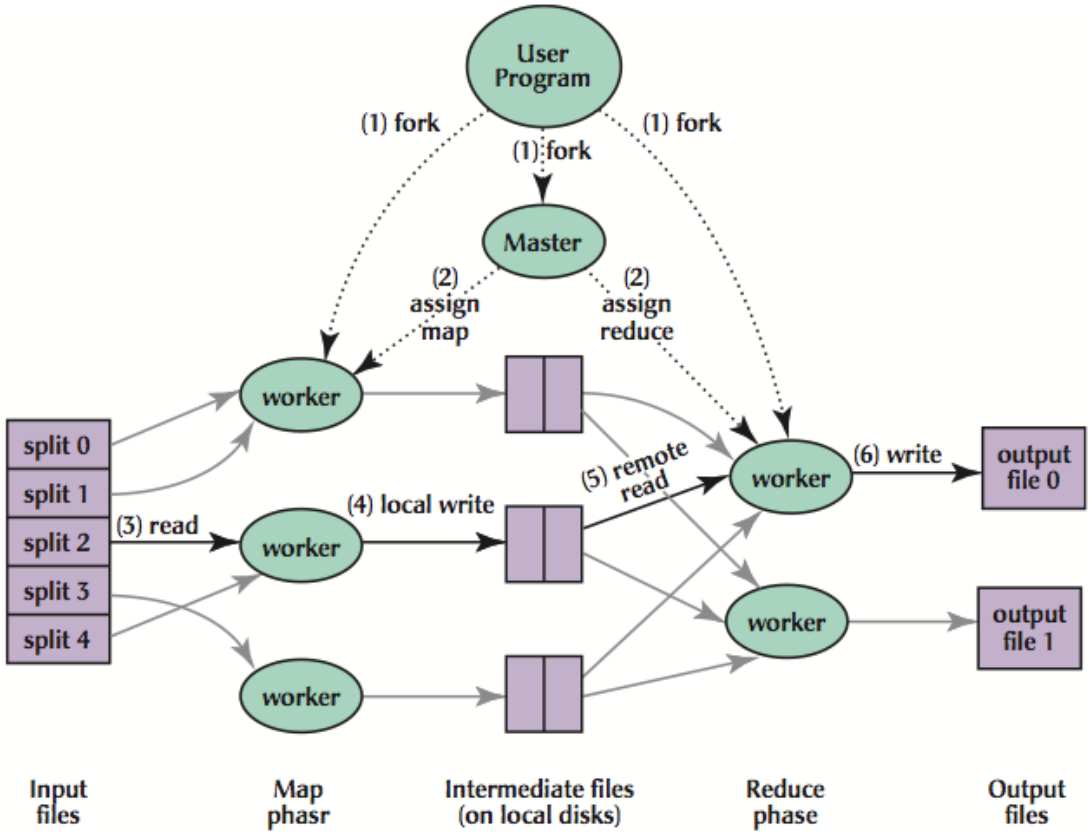

Figura 1: Visão geral do funcionamento do modelo MapReduce.

1. A biblioteca MapReduce no programa do usuário primeiro divide os arquivos de entrada em M pedaços. Em seguida, iniciam-se muitas cópias do programa em um cluster de máquinas.

2. Uma das cópias do programa é especial: o mestre (master). Os demais são

trabalhadores (escravos, slaves) cujo trabalho é atribuído pelo mestre. Existem M tarefas Map e R tarefas Reduce a serem atribuídas. O mestre atribui aos trabalhadores ociosos uma tarefa Map ou uma tarefa Reduce.

3. Um trabalhador que recebe uma tarefa Map lê o conteúdo do fragmento de entrada correspondente. Ele analisa pares (chave, valor), a partir dos dados de entrada e encaminha cada par para a função Map definida pelo usuário. Os pares (chave, valor) intermediários, produzidos pela função Map, são colocados no buffer de memória;

4. Um trabalhador que recebe uma tarefa Map lê o conteúdo do fragmento de entrada correspondente. Ele analisa pares (chave, valor), a partir dos dados de entrada e encaminha cada par para a função Map definida pelo usuário. Os pares (chave, valor) intermediários, produzidos pela função Map, são colocados no buffer de memória;

5. Periodicamente, os pares colocados no buffer são gravados no disco local, divididos em regiões R pela função de particionamento. As localizações desses pares bufferizados no disco local são passadas de volta para o mestre, que é responsável pelo encaminhamento desses locais aos trabalhadores Reduce;

6. Quando um trabalhador Reduce é notificado pelo mestre sobre essas localizações, ele usa chamadas de procedimento remoto para ler os dados no buffer, a partir dos discos locais dos trabalhadores Map. Quando um trabalhador Reduce tiver lido todos os dados intermediários para sua partição, ela é ordenada pela chave intermediária para que todas as ocorrências da mesma chave sejam agrupadas. Se a quantidade de dados intermediários é muito grande para caber na memória, um tipo de ordenação externa é usado;

7. O trabalhador Reduce itera sobre os dados intermediários ordenados e, para cada chave intermediária única encontrada, passa a chave e o conjunto correspondente de valores intermediários para função Reduce do usuário. A saída da função Reduce é anexada a um arquivo de saída final para essa partição Reduce;

8. Quando todas as tarefas Map e Reduce são concluídas, o mestre acorda o programa do usuário. Neste ponto, a chamada MapReduce no programa do usuário retorna para o código do usuário.

2.2.1 Hadoop

O Hadoop [White 2009] é uma das implementações do MapReduce, um *framework open source* desenvolvido por Doug Cutting em 2005 que provê o gerenciamento de computação distribuída.

Um dos principais benefícios do Hadoop é a sua capacidade de lidar com falhas (disco, processos, falhas de nós), permitindo que o trabalho do usuário possa ser concluído.

Facebook, Yahoo! e eBay utilizam o ambiente Hadoop para processar diariamente terabytes de dados e logs de eventos em seus *clusters* para detecção de spam, *business intelligence* e diferentes tipos de otimização [Cherkasova 2011].

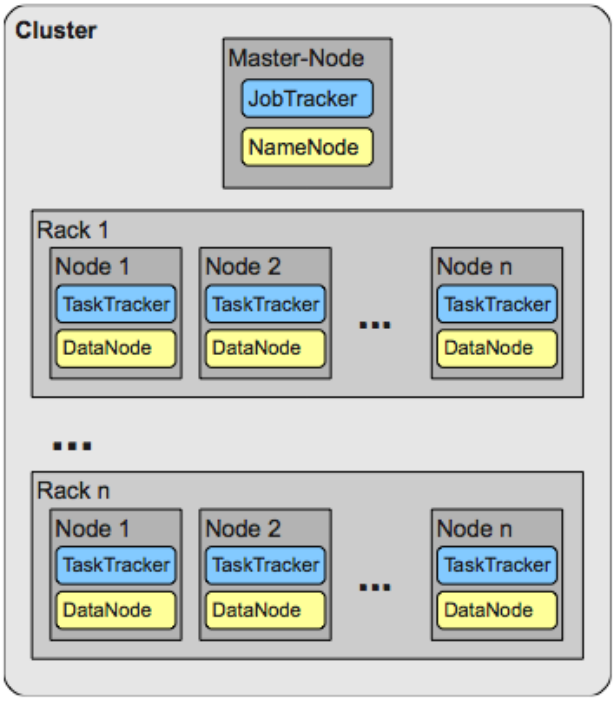

Figura 2: Visão abstrata do cluster.

2.3 Ordenação Paralela

Um grande número de aplicações paralelas possui uma fase de computação intensa, na qual uma lista de elementos deve ser ordenada com base em algum de seus atributos. Um exemplo é o algoritmo de Page Rank [Page et al. 1999] da Google: as páginas de resultado de uma consulta são classificadas de acordo com sua relevância,

e então precisam ser ordenadas de maneira eficiente [Kale e Solomonik 2010]. No exemplo do Page Rank, o número de páginas a serem ordenadas é enorme, e elas são recolhidas de diversos servidores da Google; é uma questão fundamental escolher algoritmo paralelo com o melhor desempenho dentre as soluções possíveis.

Na criação de algoritmos de ordenação paralela, é ponto fundamental ordenar coletivamente os dados de cada processo individual, de forma a utilizar todas as unidades de processamento e minimizar os custos de redistribuição de chaves entre os processadores.

- importância da ordenação paralela
- formas de ordenação: memória e disco
- grandes dados: apenas disco
- grandes dados: problematização (tempo, limite de memória)
- grandes dados: sort benchmark

- algoritmos de ordenação paralelos
- funcionamento geral
- condições / ingredientes / limites
- diferentes algoritmos para diferentes aplicações
- descrição de algoritmos (e diagramas): sample sort, quick sort

2.4 Algoritmos de ordenação Paralela

Condições de implementação de algoritmos paralelos de ordenação

• **Habilidade de explorar distribuições iniciais parcialmente ordenadas** Alguns algoritmos podem se beneficiar de cenários nos quais a sequência de entrada dos dados é mesma, ou pouco alterada. Nesse caso, é possível obter melhor desempenho ao realizar menos trabalho e movimentação de dados. Se a alteração na posição

dos elementos da sequência é pequena o suficiente, grande parte dos processadores mantém seus dados iniciais e precisa se comunicar apenas com os processadores vizinhos.

Movimentação dos dados A movimentação de dados entre processadores deve ser mínima durante a execução do algoritmo. Em um sistema de memória distribuída, a quantidade de dados a ser movimentada é um ponto crítico, pois o custo de troca de dados pode dominar o custo de execução total e limitar a escalabilidade.

Balanceamento de carga O algoritmo de ordenação paralela deve assegurar o balanceamento de carga ao distribuir os dados entre os processadores. Cada processador deve receber uma parcela equilibrada dos dados para ordenar, uma vez que o tempo de execução da aplicação é tipicamente limitada pela execução do processador mais sobrecarregado.

Latência de comunicação A latência de comunicação é definida como o tempo médio necessário para enviar uma mensagem de um processador a outro. Em grandes sistemas distribuídos, reduzir o tempo de latência se torna muito importante.

Sobreposição de comunicação e computação Em qualquer aplicação paralela, existem tarefas com focos em computação e comunicação. A sobreposição de tais tarefas permite que sejam feitas tarefas de processamento e ao mesmo tempo operações de entrada e saída de dados, evitando que os recursos fiquem ociosos durante o intervalo de tempo necessário para a transmissão da carga de trabalho.

2.4.1 Sample Sort

2.4.2 Quick Sort

3 Metodologia

O início do projeto será destinado ao estudo mais detalhado da computação paralela, em especial os algoritmos de ordenação paralela, dos fatores que influenciam o desempenho de tais algoritmos, o modelo MapReduce e a plataforma Hadoop. O passo seguinte é conhecer detalhadamente o algoritmo paralelo a ser implementado e definir as estratégias para sua implentação ambiente Hadoop. O algoritmo implementado deve ser cuidadosamente avaliado para verificar um funcionamento adequado com diferentes entradas e número de máquinas.

Em seguida, serão realizados experimentos para testes de desempenho dos algoritmos com relação à quantidade de máquinas, quantidade de dados e conjunto de dados. Os resultados obtidos serão analisados e permitirão comparar a desempenho dos algoritmos em cada situação.

3.1 Infraestrutura necessária

A infra estrutura necessária ao desenvolvimento do projeto será fornecida pelo Laboratório de Redes e Sistemas (LABORES) do Departamento de Computação (DECOM). Esse laboratório possui um *cluster* formado por cinco máquinas Dell Optiplex 380, que serão utilizadas na realização dos testes dos algoritmos. Os algoritmos serão desenvolvidos em linguagem Java, de acordo com o modelo MapReduce, no ambiente Hadoop.

Cada máquina do *cluster* apresenta as seguintes características:

- Processador Intel Core 2 Duo de 3.0 GHz
- Disco rígido SATA de 500 GB 7200 RPM
- Memória RAM de 4 GB
- Placa de rede Gigabit Ethernet
- Sistema operacional Linux Ubunbu 10.04 32 bits
- Sun Java JDK 1.6.0 19.0-b09

- Hadoop 0.20.2

3.2 Cronograma de trabalho

O cronograma de trabalho inclui as atividades que devem ser realizadas e como elas devem ser alocadas durante as disciplinas TCC I e TCC II para que o projeto possa ser concluído com sucesso. As tarefas a serem desenvolvidas estão descritas a seguir:

1. Pesquisa bibliográfica sobre o tema do projeto e escrita da proposta
2. Estudo mais detalhado dos algoritmos de ordenação paralela, modelo MapReduce e Hadoop.
3. Configuração do ambiente Hadoop no laboratório.
4. Implementação e testes.
5. Escrita, revisão e entrega do relatório.
6. Análise comparativa entre os resultados.
7. Escrita e revisão do projeto final.
8. Entrega e apresentação.

Na Tabela 1 está descrito o cronograma esperado para o desenvolvimento do projeto. Cada atividade foi alocada para se adequar da melhor maneira ao tempo disponível, mas é possível que o cronograma seja refinado posteriormente, com a inclusão de novas atividades ou redistribuição das tarefas existentes.

Atividade	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov
1	•	•								
2		•	•							
3			•							
4			•	•		•	•			
5				•	•					
6								•		
7									•	
8										•

Tabela 1: Cronograma proposto para o projeto

3.3 Desenvolvimento //

3.4 Descrição dos experimentos

4 Resultados Preliminares

5 Conclusões e Propostas de Continuidade

Referências Bibliográficas

[Aho et al. 1974]Aho, A.; Hopcroft, J.; Ullman, J. *The Design and Analysis of Computer Algorithms*. [S.l.]: Addison-Wesley, 1974. (Addison-Wesley Series in Computer Science and Information Processing).

[Amato et al. 1996]Amato, N. M.; Iyer, R.; Sundaresan, S.; Wu, Y. *Y.: A comparison of parallel sorting algorithms on different architectures*. [S.l.], 1996.

[Asanovic et al. 2009]Asanovic, K.; Bodik, R.; Demmel, J.; Keaveny, T.; Keutzer, K.; Kubiatowicz, J.; Morgan, N.; Patterson, D.; Sen, K.; Wawrzynek, J.; Wessel, D.; Yelick, K. A view of the parallel computing landscape. *Commun. ACM*, ACM, New York, NY, USA, v. 52, n. 10, p. 56–67, out. 2009.

[Cherkasova 2011]Cherkasova, L. Performance modeling in mapreduce environments: challenges and opportunities. In: *ICPE'11*. [S.l.: s.n.], 2011. p. 5–6.

[Dean e Ghemawat 2008]Dean, J.; Ghemawat, S. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, ACM, New York, NY, USA, v. 51, n. 1, p. 107–113, jan. 2008.

[Ernst et al. 2009]Ernst, D.; Wittman, B.; Harvey, B.; Murphy, T.; Wrinn, M. Preparing students for ubiquitous parallelism. In: *SIGCSE*. [S.l.: s.n.], 2009. p. 136–137.

[Gantz 2008]Gantz, J. *The Diverse and Exploding Digital Universe*. 2008.

[Kale e Solomonik 2010]Kale, V.; Solomonik, E. Parallel sorting pattern. In: *Proceedings of the 2010 Workshop on Parallel Programming Patterns*. New York, NY, USA: ACM, 2010. (ParaPLoP '10), p. 10:1–10:12.

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

[Manferdelli et al. 2008]Manferdelli, J. L.; Govindaraju, N. K.; Crall, C. Challenges and opportunities in Many-Core computing. *Proceedings of the IEEE*, v. 96, n. 5, p. 808–815, may 2008.

[Page et al. 1999]Page, L.; Brin, S.; Motwani, R.; Winograd, T. *The PageRank Citation Ranking: Bringing Order to the Web*. 1999.

[Prinslow 2011]Prinslow, G. *Overview of Performance Measurement and Analytical Modeling Techniques for Multi-core Processors*. 2011.

[Ranger et al. 2007]Ranger, C.; Raghuraman, R.; Penmetsa, A.; Bradski, G.; Kozyrakis, C. Evaluating mapreduce for multi-core and multiprocessor systems. In: *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2007. (HPCA '07), p. 13–24.

[Satish et al. 2009]Satish, N.; Harris, M.; Garland, M. Designing efficient sorting algorithms for manycore gpus. In: *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*. Washington, DC, USA: IEEE Computer Society, 2009. p. 1–10.

[White 2009]White, T. *Hadoop: The Definitive Guide*. first edition. [S.l.]: O'Reilly, 2009.