Análise e Otimização do Apache Hadoop em uma Arquitetura Paralela com Memória Compartilhada

Éder J. Scheid[1], **Paulo V. M. Cardoso**[1], **Andrea S. Charão**[1]

[1]Laboratório de Sistemas de Computação – Universidade Federal de Santa Maria (UFSM)
Santa Maria – RS – Brasil

eder@inf.ufsm.br, pcardoso@inf.ufsm.br, andrea@inf.ufsm.br

Resumo. *O foco deste trabalho é analisar a otimização do* framework Apache Hadoop *em uma arquitetura paralela onde os dados não estão distribuídos. Os resultados encontrados mostram que a configuração deste* framework *se demonstra muito sensível à diversos fatores relacionados à peculiaridades da arquitetura utilizada.*

1. Introdução

O *framework Apache Hadoop* é uma ferramenta voltada para ambientes onde os dados estão distribuídos entre vários computadores, o qual oferece uma ótima solução para o processamento de *big data* [SAS Institute Inc. 2014], mas a literatura sobre como este *framework* se comporta em um ambiente com memória compartilhada, no caso uma SGI UV2000 [Silicon Graphics International Corp 2014], ainda não possui um grande acervo de resultados relevantes.

O objetivo geral deste trabalho é executar e analisar experimentos com o *framework Apache Hadoop* em um ambiente para o qual o mesmo não foi projetado: uma máquina NUMA (*Non-uniform memory access*) [Bligh, Martin J. and Dobson, Matt and Hart, Darren 2004], que apresenta uma arquitetura paralela. Máquinas NUMA aproveitam o fato que processadores *multicore* vêm se popularizando cada vez mais e se tornam uma alternativa para o processamento de alto desempenho em uma única máquina. Além disso, no sistema de armazenamento da máquina utilizada, todos os dados são armazenados em um só local, sem a necessidade de transferência pela rede.

A proposta do presente trabalho é relevante pois experimenta o uso de uma tecnologia (*Apache Hadoop*) projetada para um ambiente específico (cluster de computadores) em um ambiente totalmente diferente do seu propósito (única máquina NUMA), podendo assim gerar dados satisfatórios e contribuir para a comunidade com dados e análises e ainda otimizar a tecnologia para o uso no ambiente em questão.

2. Desenvolvimento

2.1. Algoritmo Usado

Para os primeiros experimentos, foi executado o algoritmo de exemplo PiEstimator [The Apache Software Foundation 2009], que aplica o método de quasi-Monte Carlo. O algoritmo PiEstimator recebe dois argumentos, o primeiro sendo o número de tarefas *map* que serão iniciadas e o segundo o número de pontos utilizados para estimar o valor do Pi. A escolha deste algoritmo se deve ao fato de que não existe um uso extensivo de disco

para calcular o valor do Pi, mas sim um uso maior de memória e processamento destes dados.

2.2. Resultados Iniciais

Para termos uma base de comparação inicial, foi escolhido usar o *framework Hadoop* em sua configuração *default*, ou seja, sem nenhuma alteração e no modo *standalone*, na qual nem o sistema de arquivos do *Hadoop* (*HDFS*) [Apache Software Foundation 2014b] ou o gerenciador de recursos (*YARN*) [Apache Software Foundation 2014a] foram configurados. Além disto foram feitos experimentos com o *Hadoop* configurado para usar o *HDFS* e o *YARN*, como também configurado para usar 80% da memória disponível no sistema no caso 397 *gigabytes*. Com o algoritmo escolhido, foram feitos, para cada configuração, um total de 25 experimentos com número de pontos aumentando gradualmente, como também o número de *maps*. O número de pontos variou de 1 milhão de pontos até 10 bilhões de pontos, já o número de *maps* variou de apenas 1 *mapper* até 48 (número total de *cores* da máquina utilizada) *mappers*.

2.2.1. Configurações do Hadoop

O *Hadoop* em modo *standalone* é executado como um simples processo Java, em modo não distribuído. O sistema de disco local é usado, não há replicação de dados e não existe nenhum outro processo gerenciando todos os componentes do *framework*. Nenhum dos arquivos de configuração do *Hadoop* foram alterados, garantindo uma base para comparações.

Já com a configuração pseudo-distribuída, o *framework Hadoop* simula um processamento distribuído em uma única máquina fazendo com que cada parte do *Hadoop* execute em um processo Java diferente. Foram configurados os seguintes arquivos: *core-site.xml, hdfs-site.xml, mapred-site.xml e yarn- site.xml*. Após a configuração do *Hadoop* e seus arquivos de propriedades, foram realizados os mesmos experimentos na qual o ambiente *standalone* foi submetido.

2.2.2. Comparações entre os resultados

A Figura 1, mostra o gráfico resultante dos experimentos com o *Hadoop* operando em modo *standalone*. Podemos notar que, quanto mais pontos e mais tarefas *map*, mais tempo a execução do algoritmo leva. Uma outra característica só se apresenta com números de pontos maiores, por exemplo, todos os algoritmos que executaram com 10 bilhões de pontos, demoraram aproximadamente 10 vezes mais do que quando executaram com 1 bilhão de pontos. O padrão deve seguir se duplicarmos o número de *maps* e aumentarmos a quantidade de pontos em 10 vezes, pois uma única máquina Java cuidará de todos estas tarefas *maps*, mostrando assim um comportamento não paralelo.

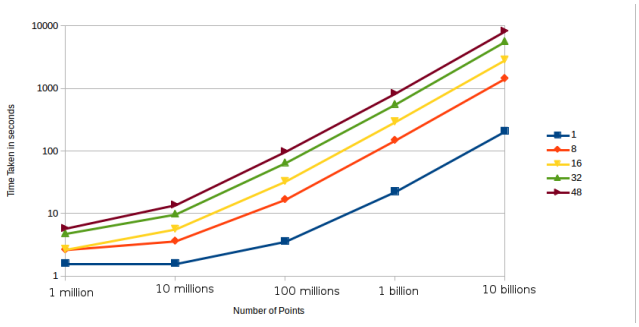

Figura 1. Gráfico do Número de Pontos por Tempo Levado para execução no Hadoop 2.5.0 Standalone

Quando comparamos a execução de apenas uma tarefa *map* entre o *Hadoop Standalone* e o *Hadoop* em modo Pseudo-Distribuído, notamos que o tempo de execução com 10 bilhões de pontos é praticamente o mesmo. Mas nos outros pontos temos a diferença de aproximadamente 18 segundos. Esta diferença pode se dar por diversos fatores, como o coletor de lixo do Java tentando limpar a memória ou o tempo que o gerenciador de recursos demora para alocar as diversas estruturas de dados necessárias.

Ao final destes dois experimentos, percebemos que para este caso, usando o algoritmo *PiEstimator*, o uso de uma configuração pseudo-distribuído não é favorável, visto que uma vez obteve um desempenho baixo comparado à configuração *standalone*, possivelmente gastando tempo de processamento em outras tarefas que afetaram a execução do algoritmo.

3. Otimização do Hadoop

Após os experimentos iniciais com as duas configurações (*standalone* e pseudo-distribuído) foi necessário reavaliar a configuração imposta para o *Hadoop*. Levantando questionamentos sobre qual algoritmo de escalonamento (*scheduler*) o *Hadoop* estaria usando, sobre o número de *containers* que o *Hadoop* estaria alocando e como estes usam a memória e bem como outras variáveis que poderiam estar afetando o desempenho do *Apache Hadoop*.

Após os novos parâmetros do *Hadoop* devidamente configurados foram realizados experimentos visando comparar os novos resultados com os obtidos a partir dos experimentos iniciais.

3.1. Discussão e comparação dos Resultados

Executando o algoritmo do *PiEstimator* com o *Hadoop* otimizado e nas mesmas condições, número de *mappers* e número de pontos, dos testes iniciais, chegamos à resultados particularmente interessantes em relação aos testes anteriores à otimização do *framework*. Comparando estes resultados com os resultados encontrados com o *Hadoop* utilizando apenas 3 *containers*, chegamos ao gráfico da Figura 2 que mostra a porcentagem de aumento no desempenho do *Hadoop* com a nova configuração. Em quase todos os casos existiu um aumento no desempenho do *Hadoop*. Foi constatado ainda, uma diminuição no tempo de execução de aproximadamente 4682.3 segundos, ou seja, um aumento de até 94.93% no desempenho do *Hadoop* otimizado quando o algoritmo do *Pi-*

Estimator executou com 32 *mappers*. Possivelmente, comprovando assim a teoria de que o *Apache Hadoop* não faz uso dos recursos (*cores*) totais da máquina.

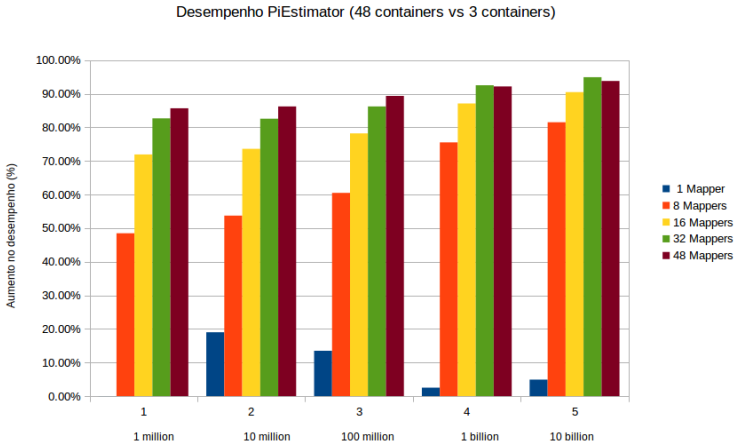

Figura 2. Aumento no desempenho do algoritmo PiEstimator entre o Hadoop configurado para 48 *containers* e 3 *containers*.

4. Conclusão

Este trabalho teve como objetivo principal analisar o comportamento do *framework Apache Hadoop* em uma máquina (SGI UV2000) com uma arquitetura bem específica (arquitetura de acesso à memória não uniforme).

Tendo em vista os resultados encontrados durante os experimentos, concluímos que a otimização do *framework Apache Hadoop* em máquinas de memória compartilhada, em especial na SGI UV2000, se demonstra trabalhosa e com muitas variáveis que devem serem levadas em consideração ao configurar um *framework* deste tipo.

Referências

Apache Software Foundation (2014a). Apache hadoop nextgen mapreduce (yarn). http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html, Acesso em setembro de 2014.

Apache Software Foundation (2014b). Hdfs architecture guide. http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html, Acesso em setembro de 2014.

Bligh, Martin J. and Dobson, Matt and Hart, Darren (2004). Linux on numa systems.

SAS Institute Inc. (2014). Big data - what it is and why it matters. http://www.sas.com/en_us/insights/big-data/what-is-big-data.html, Acesso em agosto de 2014.

Silicon Graphics International Corp (2014). Sgi uv: The big brain computer. https://www.sgi.com/products/servers/uv/, Acesso em agosto 2014.

The Apache Software Foundation (2009). Class piestimator. http://hadoop.apache.org/docs/r1.1.2/api/org/apache/hadoop/examples/PiEstimator.html, Acesso em setembro de 2014.