Map Reduce Aplicado ao Sequenciamento de DNA Humano. Comparativo entre Implementações das Linguagens Java e C++

Raffael B. Schemmer, Bruno R. Filho, Junior F. Barros, Júlio C. Anjos, Cláudio F. R. Geyer
Instituto de Informática
Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre, RS, Brasil
{raffael.schemmer, brfilho, jfbarros, jcsanjos, geyer}@inf.ufrgs.br

Resumo

This work presents a comparative study of two approaches of programming of Map Reduce model, supported by Hadoop environment. This proposed uses C++ and Java languages to implement a problem that processes an alignment and sequencing samples of human genome DNA. In this study, we compare the approaches and implementation in qualitative and quantitative view. Showing the overhead in each of two approaches, though gathering developing costs as execution gains. The results showing Java is more simple and strong, in both programming details and features as in execution scheduling of process and execution timings.

1. Introdução

A evolução dos computadores, têm dependência direta no crescimento do número de transistores dos circuitos integrados, componentes hoje, fundamentais para construção dos computadores. Dispositivos de caráter eletromecânicos são utilizados para o armazenamento da informação de forma persistente tendo por natureza tecnológica, permitir a persistência da informação na ordem de centenas a milhares de vezes mais, comparado a um circuito integrado, formado por transistores, equivalente a uma memória do tipo persistente. Ainda, dado a limitação tecnológica, dispositivos eletromecânicos tendem a ser dezenas de vezes mais lentos, com relação a capacidade de leitura e escrita de dados, e centenas a milhares de vezes mais lentos, com relação ao tempo de acesso de um dado.

Frente a disponibilidade em número de transistores e as restrições tecnológicas impostas pelos dispositivos eletromecânicos, o cenário atual implica no uso de diferentes tipos de tecnologias de memória, assumindo por hora que cada qual, sobreponha apenas suas qualidades, permitindo que por fim, se consiga ler e escrever dados na hierarquia de memória, de maneira que o ou os processadores do sistema computacional, se mantenham ativos e operacionais, com dados a serem processados a maior parte do tempo possível. No escopo da computação orientada a dados, onde a maior parte do tempo consumido pela aplicação se resume a leitura e a escrita dos dados na memória persistente, é comum que se faça uso de réplicas de recursos do tipo eletromecânicos, de maneira a melhorar as taxas globais de leitura e escrita dos dados e do tempo de acesso. Taís réplicas podem se dar de forma centralizada, onde em apenas um nodo ou nó do sistema computacional existam um ou mais dispositivos eletromecânicos. Outra abordagem, feita de forma distribuída, diferentes réplicas dos dados são persistidas em nodos ou nós em diferentes sistemas computacionais, conectados sobre diferentes redes de interconexão. Este cenário, referenciado pela literatura como máquina do tipo Cluster [2], será a arquitetura de máquina alvo utilizada por este trabalho.

No escopo das disciplinas de programação de alto desempenho e sistemas distribuídos, o uso de linguagens e modelos de programação paralelas simplificados são extremamente comuns e aplicados, dado principalmente seu poder de abstração e aplicação e de representação. É comum se fazer uso de linguagens como C++ juntamente com bibliotecas para programação concorrente e distribuída como PThreads e MPI. Em contrapartida, Java não favorece a exploração das características comentadas, sendo estas uma das características que inibem o uso da linguagem no escopo de programação paralela de alto desempenho. Diante dos aspectos comentados, e assumindo que o ambiente Hadoop é implementado de maneira nativa sobre Java, e também, dá suporte à linguagem C++ e o modelo de programação Map Reduce através do componente Hadoop Pipes, este trabalho têm como motivação e objetivo, rescrever a proposta de Bruno Filho em linguagem C++, comparando questões quantitativas com relação ao desenvolvimento da aplicação e qualitativas quanto ao tempo de execução.

Este trabalho assume as vantagens do ambiente Hadoop na simplificação do modelo de programação e das funcionalidades orientadas ao mapeamento e particionamento dos dados podem favorecer e minimizar o esforço quanto a codificação de uma aplicação de alto desempenho com foco intensivo em dados. Ainda, a capacidade de escrita de código em linguagem C++, oferece uma oportunidade que assume minimização do esforço quanto ao aprendizado da sintaxe e semântica da linguagem Java, e também do entendimento básico de técnicas avançadas de orientação a objetos utilizadas pela linguagem. Este trabalho desenvolve um estudo comparativo de duas implementações de um problema de análise e sequenciamento de amostras de DNA humano, tendo como plataforma alvo o ambiente Hadoop utilizando linguagens Java e C++. Se pretende através deste, realizar um comparativo quanto ao esforço de desenvolvimento e ao tempo de execução e consumo de recursos das diferentes implementações propostas.

2. Conceitos Básicos e Implementação

O ambiente Hadoop [4], dá suporte a uma infraestrutura de persistência distribuída, de maneira que para um dado escrito sobre o sistema de arquivos distribuído utilizado, seja possível aumentar a velocidade de acesso da informação e também suportar o conceito de tolerância a falhas, onde um mesmo dado é replicado várias vezes, de maneira que caso exista falha em uma das réplicas, as demais assegurem o acesso ao dado. Também, o ambiente Hadoop dá suporte ao processamento distribuído, através da técnica de programação Map Reduce. Através de um conjunto de aplicações, ou serviços implementados sobre o ambiente Hadoop, responsáveis pela execução e escalonamento automatizado da aplicação, e também pelo particionamento e pela persistência automatizada dos dados, o ambiente Hadoop e o modelo de programação Map Reduce, oferecem um ambiente de programação de aplicações distribuídas, que simplifica em grande parte as atividades envolvidas no projeto de aplicações distribuídas.

A técnica de programação Map Reduce, associada as vantagens existentes na infraestrutura do Hadoop, nas questões de persistência, particionamento e escalonamento distribuído dos dados, favorece o uso desta plataforma para a implementação de um problema referente a análise e sequenciamento de DNA humano, processo este detalhado de forma resumida no parágrafo anterior. No ano de 2013/II, o então aluno de graduação na época Bruno Filho [3], realizou a proposta e implementação do problema de análise e sequenciamento de DNA humano fazendo uso da técnica de programação Map Reduce, de maneira conjunta com o ambiente Hadoop. No trabalho é utilizado o modelo de Map Reduce, para implementação das etapas de sequenciamento das amostras dos pacientes sequenciados com a base de referência do genoma humano para os genes de interesse. O processo de busca das mutações nas bases públicas é feito pela atividade de Reduce, que utiliza o resultado do sequenciamento realizado na etapa de Map, procurando para uma mutação encontrada uma possível patologia cadastrada nas bases públicas utilizadas.

A Figura 1 ilustra o modelo geral da aplicação desenvolvida. O campo (dados de entrada) refere-se aos dados das amostras dos pacientes, que contém o DNA a ser sequenciado. O trabalho proposto assume que os dados de entrada são disponibilizados de forma alinhada, sem erros ou redundâncias quanto a leitura. Também, os dados referentes ao genoma dos genes disponibilizados na entrada e o banco público de mutações são consultados em um procedimento externo a aplicação Map Reduce, onde apenas os dados dos genes são persistidos em arquivos específicos. Estes arquivos que contêm os genes do genoma de referência e as mutações dos genes são carregados em uma estrutura do tipo cache distribuída na implementação Java, e estaticamente em memória na versão C++, e utilizadas respectivamente nas etapas de Map e Reduce. A cache distribuída permite que todo nó escravo tenha acesso estático ao dado em nível de memória, não sendo necessário realizar consultas distribuídas a informação.

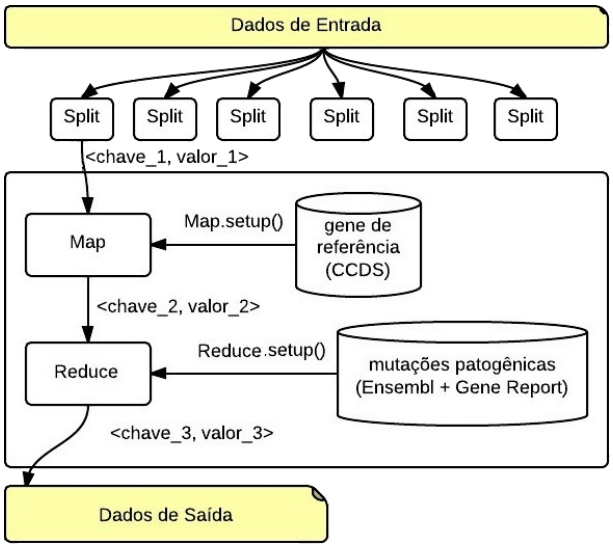

Figura 1. Modelo da aplicação de sequenciamento de DNA usando Map Reduce.

O funcionamento da aplicação acontece em duas etapas. Na etapa de Map a amostra do paciente é fragmentada em pedaços ou chunks, ilustrados pela Figura 1 pelos splits, sendo que cada Map disparado pela aplicação irá sequenciar, ou comparar os nucleotídeos presentes nos dados de

entrada das amostras com os respectivos nucleotídeos do genoma humano referente ao gene. Caso sejam encontradas mutações ou mudanças nos genes, uma segunda etapa é realizada referente ao Reduce, que recebe apenas um conjunto de <chave, valor> sendo eles respectivamente o nucleotídeo com a mutação seguido de sua respectiva posição no gene. O processo Reduce utiliza a posição do nucleotídeo no gene como valor para buscar a mutação na base pública de mutações. A partir dos valores encontrados na base é escrito então na saída padrão do Hadoop os dados encontrados.

O Reduce realiza a busca pelas diferenças encontradas no sequenciamento nas bases públicas de mutações para o gene então sequenciado. Esta mesma base, também previamente persistida pelo usuário no sistema de arquivos distribuído do Hadoop estaria persistida na cache do Hadoop na implementação Java, uma vez que cada processo Reduce quando executado, irá precisar ter em sua memória local estes valores a serem comparados. A busca consiste em encontrar na base pública de mutações se a variação corresponde a algum tipo de patologia conhecida ou não. A aplicação deverá escrever na saída padrão do Hadoop informando caso encontrou mutação, e se ela é ou não conhecida como uma patologia pelas bases públicas de mutações.

De forma resumida, o processo referente a codificação C++ consumiu 88% maior volume de linhas de código comparado a Java. O consumo adicional deve-se ao fato de C++ não possuir estruturas capazes de manipular Strings de maneira simplificada como o Java, o que exigiu mais linhas de código para implementação das funcionalidades. As maiores dificuldades quanto a codificação C++ encontraram-se na falta de documentação e APIs para codificação de funcionalidades (Apenas existindo a Util e a Pipes) [4]. O esforço e despendimento de tempo durante o desenvolvimento concentrou-se no processo referente a compilação e depuração do código, sendo necessário recompilar as bibliotecas Util e Pipes do Hadoop, que possuíam erros internos e limitações quanto a configuração. O processo de compilação e execução do ambiente exigiu dezenas de etapas e parâmetros adicionais, sendo fortemente dependente da instalação de bibliotecas externas. Tais empecilhos impedem e ou dificultam o uso e a aplicação de C++ no desenvolvimento do trabalho e na execução e avaliação dos resultados.

3. Resultados e Avaliação

A proposta original, consistia em fazer uso do Cluster GradeP, onde originalmente o trabalho do aluno Bruno Filho foi executado e avaliado. Por várias limitações quanto a disponibilidade e instalação de bibliotecas no cluster gradeP, este trabalho propôs a criação de um Cluster distribuído, utilizando máquinas de alta performance do GPPD em caráter dedicado a execução dos resultados. O ambiente distribuído utilizado era composto de duas máquinas Intel Q8200 de 2.33GHz possuindo 4GBytes de memória RAM e discos Sata-I de 250GBytes. A conexão de rede foi feita usando um link de 100Mbps full duplex entre ambas as máquinas.

Este trabalho fez uso de um nó para a execução centralizada e de dois nós para a execução distribuída. Como trabalho da entrada foram usadas duas amostras de tamanhos iguais a 1GBytes e 4GBytes respectivamente. A avaliação proposta mediu o tempo de execução em segundos, o consumo de memória RAM em MBytes e a quantidade de Mappers e Reducers utilizada pelo escalonador. Assumindo a disponibilidade de tempo dedicada a este trabalho, apenas os resultados referentes ao Java foram corretamente normalizados. A execução da linguagem C++ não normalizou os resultados das execuções realizadas dado o alto custo necessário e tempo despendido para a realização da implementação, configuração e execução dos resultados deste trabalho.

A primeira métrica avaliada trata do tempo de execução ilustrado pela Figura 2. Com relação a implementação Java, houve uma redução no tempo de execução ao fazer uso de um segundo nó na configuração distribuída. Tais reduções foram observadas tanto no cenário de 1GBytes como no cenário de 4GBytes. A implementação da linguagem C++ obteve um resultado contrário ao da implementação Java com relação ao tempo medido, apresentando um maior tempo de execução conforme o segundo nó foi adicionado. Ainda e mais crítico apresentam-se as diferenças no tempo das estratégias utilizadas sendo C++ em um intervalo de ordem de 20 a 80 vezes mais lento em ambos os cenários de entrada comparados a Java.

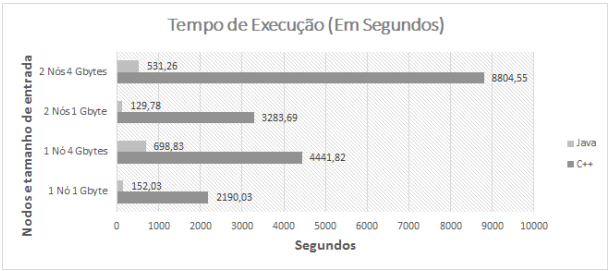

Figura 2. Tempo de execução.

A segunda métrica trata do consumo de memória ilustrado pela Figura 3. Para as duas implementações propostas, considerando tanto a variação no número de nós e no tamanho das entradas, a implementação C++ apresentou um comportamento contrário a Java, variando o consumo conforme o número de nós era variado, diferente do Java que consumiu mais memória conforme o tamanho da entrada era variado. Na maior parte dos cenários avaliados C++

apresentou um consumo próximo porém inferior aos cenários Java equivalentes.

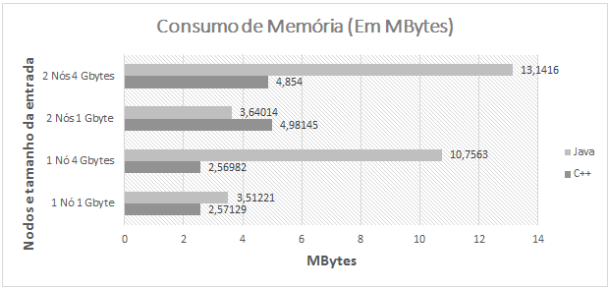

Figura 3. Consumo de memória.

A terceira e última métrica avalia a quantidade de processos do tipo Map e Reduce criados pelas implementações a partir da variação do número de nós e das entradas, sendo os resultados ilustrados pelas Figuras 4 e 5. De maneira análoga ao comportamento apresentado na métrica que avaliou o consumo de memória, Java varia consideravelmente o número de Mappers e Reducers apenas quando o tamanho da entrada é variado. Ainda, observa-se uma relação no Java que para cada 10 Mappers um Reducer é criado. Já no C++, o número de Mappers aumenta apenas quando o número de nós é variado, sendo que não houve variação no número de Reducers independente as entradas ou número de nós utilizados.

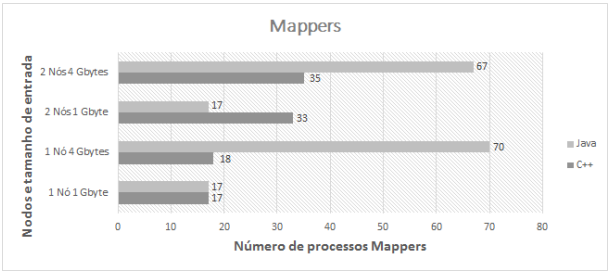

Figura 4. Número de processos Mappers.

4. Conclusões

A partir dos resultados, é possível avaliar e concluir que a abordagem baseada em C++, tanto com relação a avaliação qualitativa que trata das métricas do desenvolvimento e do ambiente de execução, apontam as deficiências do C++ em relação ao Java. Também e mais importante, os resultados quantitativos apresentam que o C++ não é capaz de atingir desempenho aceitável ou próximo ao Java, não possuindo escalabilidade conforme a variação do número de

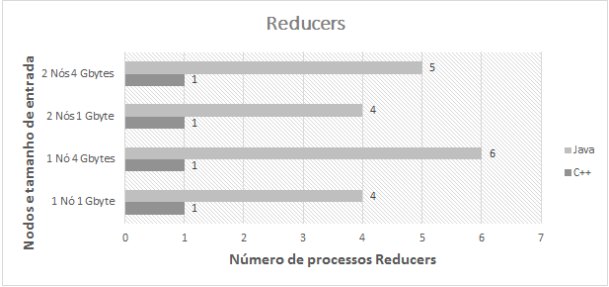

Figura 5. Número de processos Reducers.

recursos distribuídos. As métricas avaliadas apontam possíveis inconsistências quanto ao consumo de memória, e ao escalonamento dos processos Mapper e Reducers, que além de muito inferiores aos números avaliados no Java, eram apenas referenciados em momentos contrários aos utilizados pelo Java.

Os autores deste trabalho assumem a partir da pouca ou inexistente disponibilidade quanto a material e APIs para o desenvolvimento de código C++ para o ambiente Hadoop encontrados durante o desenvolvimento do trabalho, relação direta do baixo interesse pelo uso da linguagem e falta de atualização dos mecanismos de escalonamento de processos. Tais afirmações justificam os comportamentos anômalos e análogos aos detectados na alocação de memória e criação de Mappers e Reducers, que podem ter relação direta com o tempo de execução e falta de escalabilidade da implementação C++.

Como trabalhos futuros os autores deste trabalho sugerem um estudo detalhado do escalonador de processos do Hadoop, no sentido de entender quais aspectos refletem as reais inconsistências detectadas no tempo de execução medido. Ainda, que os testes sejam executados de forma que o número de processos Mapper e Reducer seja o mesmo para ambas linguagens. Por fim, que um ambiente de cluster com disponibilidade de nós seja utilizado para revalidar os resultados aferidos.

Referências

[1] B. Filho. Aplicação do mapreduce na análise de mutações genéticas de pacientes., 2013.

[2] O. Inc. Parallel Hardware Architecture. http://docs.oracle.com/cd/A87860$_0$1/$doc/paraserv$.817/a76968/$pshwarch.htm$, 2008. [$Online$; $accessed$30 − $July$ − 2014].

[3] NHGRI. NHGRI. National Human Genome Research Institute Website. http://docs.oracle.com/cd/A87860$_0$1/$doc/paraserv$.817/a76968/$pshwarch.htm$, 2014. [$Online$; $accessed$30 − $July$ − 2014].

[4] T. White. *Hadoop: The definitive guide*. "O'Reilly Media, Inc.", 2012.