**UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL**

**INSTITUTO DE INFORMÁTICA**

**PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO**

# Sequenciamento de DNA utilizando Hadoop (MR)

# (Java Vs. C++)

Trabalho Final - (TF)

Programação de Objetos Distribuídos – CMP 167

Raffael Bottoli Schemmer

# 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. Em [1] após um estudo do crescimento do número de transistores, foi identificado que a cada 18 a 24 meses dobra-se a quantidade de transistores que se podia construir sobre a mesma área de silício, resultando em uma redução de custos e aumento do desempenho dos circuitos integrados segundo uma curva exponencial. Este fato continuou a observar-se em décadas posteriores, dando origem à chamada *Lei de Moore*.

A evolução da tecnologia de semicondutores, permite aos projetistas de circuitos integrados, maior disponibilidade em área, resultando em maior capacidade de recursos, sendo eles, no escopo deste trabalho, armazenamento volátil e persistente e capacidade de processamento. Um sistema de computador tradicional atual, faz uso de diferentes tipos de circuitos integrados, uns voltados puramente para processar recursos e outros, voltados a persistir informação, seja de forma temporária ou persistente. Além dos circuitos integrados, são utilizados dispositivos de caráter eletromecânicos para o armazenamento da informação de forma persistente. De maneira análoga aos circuitos integrados, dispositivos eletromecânicos tendem por natureza tecnológica, permitir a persistência da informação na ordem de centenas a milhares de vezes mais, comparado a um circuito integrado estado da arte, 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 as restrições tecnológicas impostas, o cenário atual implica em fazer 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.

O ambiente Hadoop [3], permite dar 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 existe 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.

Nos dias de hoje, o processo de análise e sequenciamento de DNA humano, utiliza técnicas de sequenciamento de segunda geração, baseadas em equipamentos sequenciadores que utilizam tecnologia a nível de semicondutores [4]. Dado a densidade de sensores presentes sobre o equipamento semicondutor, o número de amostragens e leituras realizada pelas análises e a existência de inconsistências durante as leituras, é possível afirmar que o volume de informações produzidas pelo sequenciador, e que devem ser processadas é de grande volume. Também, as leituras tendem a serem feitas sobre um ou mais conjuntos de genes de conjuntos de pacientes, que podem variar conforme o tempo, o que aumenta o tamanho da entrada. Além dos dados amostrados pelo sequenciador, é necessário buscar em uma base de referência de DNA humano, a sequência de referência original do DNA para o gene que se quer analisar. A saber, apenas 5% desta base de referência, chamada de RefSeq presente no NCBI [5] é conhecida. Esta base de referência assume que para os genes já conhecidos, estas são as sequencias válidas e corretas.

Nesta etapa, é necessário que tanto os genes dos pacientes sejam alinhados, de maneira que apenas os genes de interesse estejam presentes nas amostras, e também, que a informação redundante seja utilizada para remover as inconsistências da leitura, que por fim são removidas das amostras. Deste primeiro procedimento, resumidamente chamado de alinhamento, deverão constar, apenas o ou os genes, de um ou mais pacientes, dependendo como o arquivo lote contendo amostras de pacientes estiver disponível. O próximo passo deve ler a grande base de referência comentada anteriormente, no sentido de buscar os genes com a sequência correta, que deverão ser comparados entre si. Ainda, é necessário buscar para os genes analisados, informações referentes a mutações em bases públicas de mutações.

As bases de mutações de genes são bancos públicos disponíveis onde pesquisadores da área médica publicam para aqueles genes, mutações encontradas em amostras de pacientes realizadas. É necessário que nos arquivos referentes aos bancos de mutações, sejam buscadas as mutações dos genes a serem analisados. A partir das amostras e do DNA de referência, é feita uma comparação entre os arquivos. Quaisquer mudanças encontradas, significam que a amostra do paciente contém uma mutação para aquele gene. A etapa a seguir representa a consulta a base de mutações, no sentido de procurar se a mutação encontrada é conhecida ou não na base de mutações. Na prática, isso significa se caso encontrada a mutação identificada, o DNA do paciente é propenso a desenvolver certos tipos de doenças associadas ao gene sequenciado. É importante lembrar, que com exceção da base de referência, as bases de mutações tendem a variar constantemente e devem ser acessadas a cada execução, no sentido de buscar novas mutações a serem comparadas, aumentando a precisão da análise.

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 para a implementação do problema de análise e sequenciamento de DNA humano, detalhado de forma resumida no parágrafo anterior. No ano de 2013, o então aluno de graduação Bruno Filho [6], realizou a proposta de implementação do problema de análise e sequenciamento de DNA humano, conforme explicado anteriormente. Nele é utilizado o modelo de Map Reduce, para implementação das etapas de sequenciamento das amostras com a base de referência do gene e da busca das mutações encontradas no sequenciamento em bases de mutações do gene, previamente buscadas e resumidas, dado as limitações encontradas e apresentadas pelo trabalho.

## 1.1. Motivação para o desenvolvimento do trabalho.

A partir dos estudos realizados, e da implementação desenvolvida pelo aluno Bruno Filho, este trabalho se propõem sobre um primeiro momento, a entender o que exatamente foi desenvolvido, e também, conseguir executar e reproduzir os experimentos realizados pelo trabalho. Durante a disciplina

de POD 2014/I, diversas foram as discussões a respeito dos prós e dos contras no uso da técnica de troca de mensagens do Java, baseado em invocação remota de métodos (RMI), onde objetos Java são serializados entre computadores distintos na rede. Ainda, o Hadoop é escrito em Java, e executa sobre a máquina virtual Java, o que lhe garante portabilidade de código fonte e suporte heterogêneo, tanto de sistema operacional como de conjunto de instruções, desde que, para estes cenários, exista uma máquina virtual Java.

Na contramão das facilidades técnicas oferecidas pelo Java, sejam elas o da máquina virtual e da portabilidade de código e plataforma, está a linguagem C++, também suportada pelo Hadoop. Na proposta utilizando C++ só é possível escrever código utilizando bibliotecas C++, sendo que a maneira como é feita a comunicação, é utilizando uma biblioteca de troca de mensagens chamada Hadoop Pipe, que utiliza a tecnologia de Sockets para comunicação. Sockets é considerado pela literatura como uma biblioteca mais voltada ao nível de rede, exigindo maior domínio do desenvolvedor quanto a chamada da conexão, e não possuindo tamanha flexibilidade quanto RMI na troca de mensagens.

Assumindo, apenas pelo senso comum, e partindo das discussões realizadas em sala de aula quanto as possíveis diferenças entre a biblioteca de troca de mensagem utilizada por C++ no uso de Sockets e de Java no uso de RMI, e também, pelo Hadoop dar suporte a ambas as tecnologias, este trabalho assume como motivação, realizar um estudo dos mecanismos de troca de mensagens, avaliando quais são seus impactos sobre uma aplicação desenvolvida para o Hadoop, que é fortemente dependente da rede e da troca de mensagens e na visão deste trabalho, pode se beneficiar, tanto em ganhos no tempo de execução, como no consumo de recursos envolvendo rede e troca de mensagens, como uso de memória.

## 1.2. Objetivos do trabalho.

Para atender as expectativas do trabalho, é necessário que uma série de atividades sejam desenvolvidas. A primeira delas, envolve entender o trabalho do aluno Bruno Filho, de maneira a conhecer o problema de análise e sequenciamento de DNA, como também, conseguir configurar e instalar o ambiente Hadoop, para entender e reproduzir os códigos fonte desenvolvidos pelo trabalho. A segunda atividade tem relação com o entendimento e estudo da biblioteca Hadoop Pipes, e também, que este trabalho consiga configurar o ambiente Hadoop para dar suporte a execução da linguagem C++. A terceira e principal atividade reside na rescrita da aplicação Java em C++, onde bibliotecas alternativas do C++ deverão ser pesquisadas para substituir as utilizadas pelo Java para implementação do problema. Por fim e não mais importante, este trabalho deverá instalar e configurar um cenário de máquina distribuída utilizando o Hadoop, para realizar a avaliação quantitativa, onde serão medidos os tempos de execução, aceleração e o consumo de recursos como rede, memória e o número de processos Map e Reduce utilizados para as implementações Java e C++.

## 1.3. Organização do restante do texto.

O restante do texto está organizando conforme a estrutura de tópicos definida a seguir:

- Capítulo II: Descreve os conceitos do trabalho relacionados ao ambiente Hadoop e ao problema de análise e sequenciamento de DNA. Também, descreve a implementação do problema comentando quais foram os desafios encontrados pelo trabalho.

- Capítulo III: Apresenta a plataforma distribuída e quais foram as dificuldades encontradas na configuração do ambiente, seguida dos resultados, comparando através de métricas quantitativas as duas implementações avaliadas.

- Capítulo IV: Descreve as conclusões obtidas por este trabalho.

# 2. Conceitos e implementação.

Este capítulo apresenta de forma resumida a implementação do problema. A Figura 1 ilustra o modelo da aplicação desenvolvida. O campo (dados de entrada) refere-se aos dados das amostras dos pacientes, que contém o DNA a ser sequenciado. Os dados utilizados por este trabalho são providos a partir de uma tabela do tipo excel (xls) que contem cinco genes para um conjunto de 100 pacientes. Uma vez que as estruturas de leitura de dados do Hadoop não dão suporte a tabelas do tipo (xls), são utilizados scripts python para converter a tabela (xls) para texto (txt) (script excel2text_format.py), e a seguir, réplicas desse arquivo são feitas para aumentar o tamanho da entrada (script text2BigInput.py). Ainda, o script que converte a tabela captura da mesma o DNA de referência para os respectivos genes, também informados na tabela. A busca nos bancos públicos de genes pelos genes das tabelas informados é feita pelo script (filter_CCDS_DB_per_gene.py), neste caso para o banco CCDS que respectivamente lê o arquivo do gene, que deve ser manualmente informado pelo usuário, separando campos estratégicos em um formato aceito pelo código da aplicação.

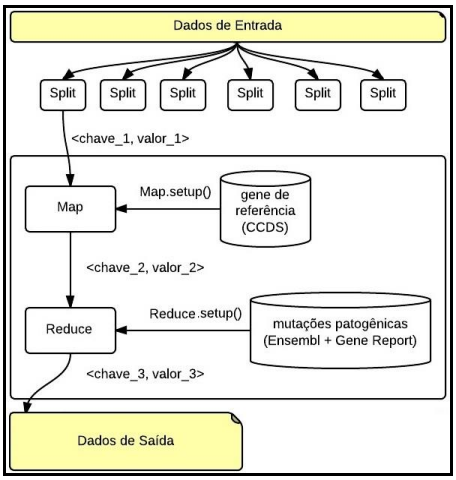


**Figura 1: Modelo da aplicação de sequenciamento de DNA usando Map Reduce.**

A aplicação realiza na atividade Map, a busca pelas linhas do arquivo de entrada, previamente persistido pelo usuário no sistema de arquivos distribuído do Hadoop para ser processado. Em cada linha estará presente um gene de um paciente a ser sequenciado. O processo de sequenciamento envolve comparar as sequencias de DNA da entrada, com o arquivo de referência do gene, também persistido pelo usuário no sistema de arquivos distribuído do Hadoop, sendo na implementação Java, alocado em uma memória cache do Hadoop, de maneira que este arquivo esteja presente em todos os nós do sistema distribuído onde processos Map forem executados. Caso diferenças na comparação forem encontradas durante o sequenciamento, o processo Map deverá gerar uma saída do tipo <*chave*, *valor*> que será repassada a um processo Reduce.

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 ira estar 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.

Para a implementação C++, foi necessário apenas no lugar da cache distribuída, uma vez não suportado, persistir os dados em estruturas do tipo String, estaticamente no código fonte. A codificação C++ de forma resumida, consumiu 88% maior volume de linhas de código comparado a Java. O maior consumo 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++ encontram-se na falta de documentação e APIs para codificação de funcionalidades (Apenas existindo a Util e a Pipes). O processo que mais exigiu tempo e esforço no desenvolvimento foi referente a compilação e depuração do código, principalmente com relação a configuração do ambiente, sendo necessário recompilar as bibliotecas Util e Pipes, culpa de erros existentes em suas configurações. A compilação também, exigiu dezenas de parâmetros, sendo fortemente dependente da instalação de bibliotecas externas.

# 3. Resultados.

Este capítulo apresenta as questões voltadas a instalação e configuração da infraestrutura utilizada pelo trabalho, seguida da avaliação quantitativa das linguagens utilizadas para implementação do problema.

## 3.1. Infraestrutura utilizada pelo trabalho.

Uma vez que a implementação do problema para as duas abordagens utilizadas foi concluída, este trabalho realizou a instalação e configuração da infraestrutura referente a máquina distribuída para avaliação deste trabalho. A proposta original, consistia em fazer uso do Cluster GradeP, onde originalmente o trabalho do aluno Bruno Filho foi executado e avaliado. Os testes iniciais do código Java foram feitos sobre a configuração original e estendida desta mesma máquina, que para execução do código C++ ofereceu restrições quanto a configuração existente do Hadoop e a disponibilidade de bibliotecas necessárias para compilar o código C++ para executar na máquina alvo. Por esta limitação, este trabalho propôs a criação de um Cluster distribuído, utilizando quatro máquinas de alta performance do GPPD.

Em cada uma das quatro máquinas utilizadas, foi feita uma instalação manual do Hadoop, versão 1.0.4. A configuração do Hadoop exige que apenas a máquina mestre, responsável por executar a estrutura (Namenode) do HDFS contenha no arquivo slaves da raiz do Hadoop quais são as demais máquinas existentes no Cluster, que obrigatoriamente deverão estar com o Hadoop, contendo apenas uma estrutura de (Datanode) em execução. Todo o processo de leitura dos dados de entrada, persistência distribuída e escalonamento das tarefas da aplicação durante a execução é feito de forma automática pelo Hadoop.

Das máquinas disponíveis utilizadas, três delas possuíam processadores Intel Q8200 de 2.33GHz de frequência de operação, possuindo 4GBytes de memória RAM. A quarta máquina, utilizada como sendo a configuração mestre e persistindo a estrutura (Namenode) possuía processador Intel i5 3570 de 3.2GHz de frequência de operação, possuindo 6GBytes de memória RAM. Este trabalho utilizou HDD temporários, com objetivo de não poluir as instalações do laboratório já existentes, sendo que todas elas possuíam discos Sata-I, sendo duas delas de 250GBytes, uma de 80GBytes e outra de 120GBytes.

## 3.2. Avaliação Quantitativa (Java Vs. C++).

Com relação a avaliação quantitativa executada, é importante assumir de antemão quais foram os cenários de entrada variados e algumas políticas adotadas durante a execução dos resultados. Este trabalho fez uso de um nó para a execução sequencial e de dois nós para a execução distribuída. O tamanho da entrada foi variado em amostras de 1GByte e 4GByte respectivamente. O processo de

avaliação e medição dos resultados fez uso das saídas de log do Hadoop para medir o tempo de execução e a quantidade de Mappers e Reducers, sendo o consumo de memória e de rede medidos a partir de ferramentas nativas do Linux. Este trabalho não normalizou as execuções, ou seja, o resultado obtido é assumido da primeira execução realizada. Adotou-se esta estratégia a partir do alto custo e tempo despendido para a realização da execução dos resultados deste trabalho.

- ✓ **Tempo de execução.**

A métrica que trata do tempo de execução, aponta que para 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 1GByte como no cenário de 4GBytes. A implementação C++ obteve um resultado inverso, apresentando um maior tempo de execução conforme o segundo nó foi adicionado. As diferenças no tempo de execução das duas estratégias de programação utilizadas são expressivas, sendo a implementação C++ mais lenta que a Java em uma ordem de diferença de 20 a 80 vezes dependendo o tamanho da entrada e o número de nós utilizados.


**Figura 2: Tempo de execução das implementações Java e C++ com variação no número de nós e no tamanho da entrada.**

- ✓ **Aceleração (Speed Up).**

A métrica que trata do fator de aceleração, calculada a partir da divisão do tempo sequencial pelo tempo paralelo da configuração distribuída, aponta a eficiência das execuções aferidas. O cálculo da aceleração assume como ideal, quando a divisão do tempo sequencial pelo paralelo seja sempre igual ou próxima do número de nós utilizados. Para a implementação Java, observa-se existir uma pequena aceleração, que aumenta conforme o tamanho do problema cresce. A implementação C++ apresenta o inverso da implementação Java, demostrando piora na aceleração conforme um maior tamanho de entrada é utilizado. A aceleração demostra a escalabilidade da aplicação, que conforme vista, não acontece no código C++, nem para maiores tamanhos de entrada como maior número de nós utilizados.

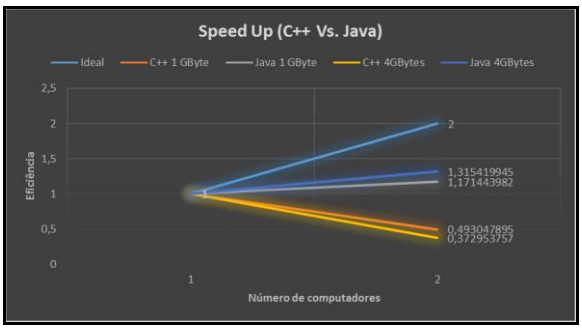


**Figura 3: Fator de aceleração para as implementações Java e C++, onde é variado o tamanho da entrada e o número de nós utilizados.**

✓ **Consumo de memória.**

Além do tempo, este trabalho durante as execuções, anotou o consumo de memória das execuções, ilustrado pela Figura 4. A implementação Java apresenta no geral maior consumo de memória que C++, variando o consumo de memória em cenários onde o tamanho da entrada é aumentado. A implementação C++ de maneira contrária, varia o consumo de memória apenas quando o número de nós distribuídos é variado.

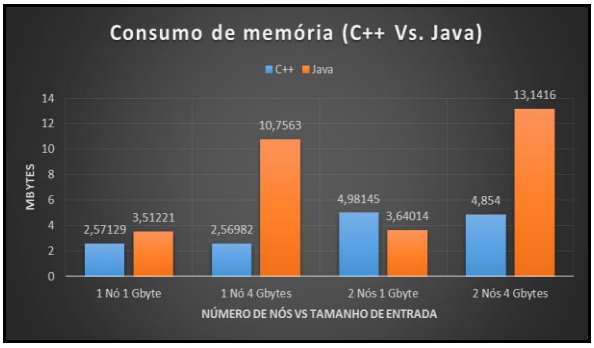


**Figura 4: Consumo de memória das implementações Java e C++, variando o tamanho da entrada e o número de nós utilizados.**

✓ **Número de Mappers e Reducers.**

Outra métrica avaliada, considera o número de processos Mappers e Reducers utilizados pelas execuções, conforme ilustra a Figura 5. É interessante observar que as implementações utilizam diferentes estratégias com relação a esta métrica, onde Java aumenta consideravelmente o número de Mappers utilizados conforme o tamanho da entrada é variado, não existindo variação conforme mais nós são utilizados. C++ por outro lado, varia o número de Mappers apenas quando há variação no número de nós. Observa-se que a relação entre Mappers e Reducers no Java segue uma relação de 10 Mappers para 1 Reducer, onde também, o Java aumenta o número de Reducers, conforme o tamanho da entrada é aumentado, não existindo variação neste número conforme o uso de mais de um nó. Para a implementação C++, apenas um Reduce foi utilizado. Estas métricas apontam um possível indício que justifica a baixa escalabilidade do C++ comparado ao Java, justificando as diferenças no tempo de execução medidos, conforme ilustra a Figura 2.

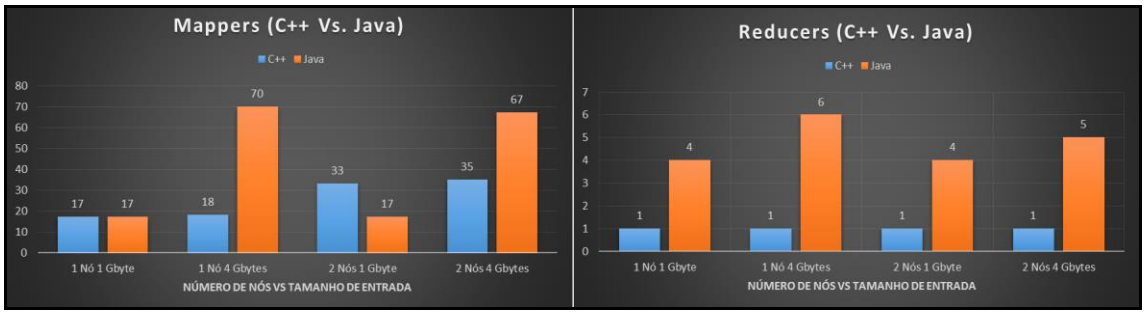


**Figura 5: Número de processos Mappers e Reducers utilizados pelo C++ e pelo Java, conforme variação no tamanho das entradas e no número de nós.**

✓ **Consumo de rede.**

O consumo de rede mede qual foi a quantidade de mensagens enviadas e recebidas durante as execuções. A Figura 6 ilustra o consumo de rede para as duas implementações propostas. Assume-se a partir dos resultados que no geral, uso intensivo de envio (UpStream) comparado a recebimento

(DownStream). Observa-se que para a implementação Java, o aumento no uso do link de envio e recebimento, ocorre aumentando o tamanho do problema, diferente do C++, que varia apenas conforme aumentado o número de nós. Este efeito da mesma forma que outras métricas avaliadas, pode ter relação e também pode ajudar a justificar o alto tempo de execução do C++ comparado ao Java.

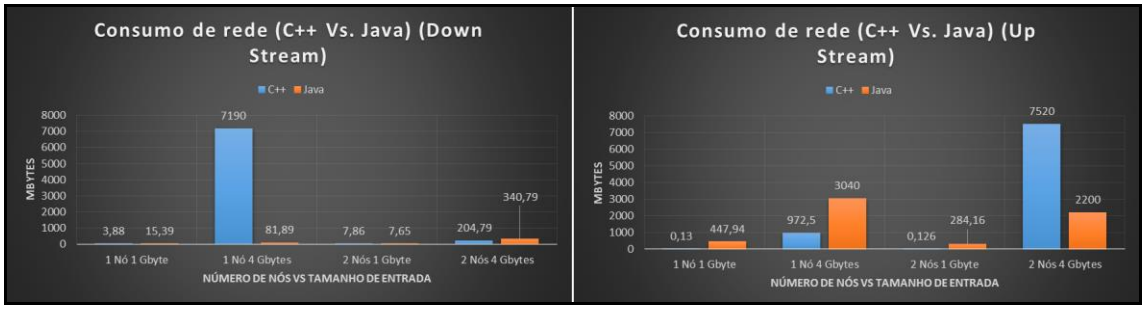


**Figura 6: Consumo de rede (DownStream e UpStream) das implementações Java e C++, conforme variação no tamanho das entradas e no número de nós.**

# 4. Conclusões.

Este trabalho apresenta um estudo comparativo de duas implementações, uma Java e outra C++ de um problema de análise e sequenciamento de DNA humano. Estas implementações são codificadas sobre o modelo de programação Map Reduce, tendo como ambiente de execução o Hadoop. Este trabalho faz uso da capacidade distribuída de persistência e processamento distribuído do Hadoop e avalia as implementações realizadas sobre diferentes tamanhos de entrada e variações no número de computadores distribuídos utilizados.

A partir dos resultados, conclui-se que a implementação C++, não é capaz de atingir desempenho próximo, ao avaliado para os mesmos cenários em Java. Ainda, o mesmo comportamento foi observado na aceleração, que assume a escalabilidade do problema conforme mais computadores são usados para resolve-lo. As métricas que tratam do consumo de memória e de rede, apontam que Java, faz uso intensivo destes recursos conforme é variado o tamanho do problema, diferente do C++ que apenas consome tais recursos apenas quanto o número de computadores é aumentado. Outro aspecto interessante detectado, têm relação ao volume de transmissão medido, que apresenta-se muito maior que o de recepção, para ambas as implementações, demostrando um possível aspecto da rede a ser melhorado ou otimizado, capaz de resultar em melhorias ao ambiente Hadoop.

Com relação ao número de processos Mappers e Reducers utilizados, detecta-se que para as duas implementações propostas, o Java só varia o número de Mappers e Reducers, conforme é variado o tamanho da entrada, sendo que C++, varia o número de processos apenas quando o número de nós é variado. Ainda, não houve variação no número de Reducers em C++, permanecendo sempre em um processo, podendo estes serem possíveis pontos causadores da diferença no tempo da execução das diferentes implementações propostas. Para a implementação Java, foi possível observar um padrão de variação no número de Mappers para Reducers em 10:1, sendo que para cada 10 Mappers, um Reduce era criado.

# 5. Referências Bibliográficas.

[1] Moore, G. “Cramming More Components Onto Integrated Circuits”. Electronics, 38(8), April 1965, pp.114-117.

[2] “Parallel Hardware Architecture”. Capturado em: http://docs.oracle.com/cd/A87860_01/doc/paraserv.817/a76968/pshwarch.htm

[3] White, T. Hadoop: the definitive guide (2nd ed.). [S.l.]: O’Reilly Media, Inc., 2010.

[4] Sequenciadores IonTorrent – Ion Proton System. Capturado em: http://products.invitrogen.com/ivgn/product/4476610.

[5] NHGRI. National Human Genome Research Institute Website. Capturado em: http://www.genome.gov.

[6] Filho, B.; “Aplicação do MapReduce na Análise de Mutações Genéticas de Pacientes.” Trabalho de Graduação, PPGC-UFRGS. Julho 2013. 41p.