Análise de Escalabilidade Horizontal em um Cluster Hbase

Cedryk Augusto dos Santos[1] **, Ronaldo Celso Messias Correia**[1]

[1]Faculdade de Ciências e Tecnologia
Universidade Estadual Paulista "Júlio de Mesquita Filho" (FCT/UNESP)
Departamento de Matemática e Computação

cedrykaugusto@gmail.com, ronaldo@fct.unesp.br

Resumo. *Com o surgimento do Big Data, identificou-se cenários onde a complexidade lógica existente na modelagem relacional não é necessária e novas soluções com características diferentes foram propostas, denominadas bancos de dados NoSQL. Os bancos de dados NoSQL tem como principal característica o escalonamento horizontal, em que os recursos de um sistema computacional aumenta com a adição de novos nós em um ambiente distribuído (cluster), em vez de substituição por um hardware mais potente. Diante disso, este trabalho avalia o potencial do escalonamento horizontal de um cluster executando o banco de dados NoSQL Hbase, realizando um benchmarking sobre diversos cenários, analisando se com o uso deste paradigma as aplicações podem continuar eficientes e disponíveis independente do tamanho da demanda de armazenamento e de consultas, quando há a adição novos nós no sistema. Os experimentos mostraram que a escalabilidade horizontal existe e é mais evidente, exceto para operações scan (busca), em conjuntos de dados maiores que no contexto dessa pesquisa foram de que neste trabalho foram 100.000 e 1.000.000 de registros de 1KB de tamanho cada e ao se aumentar o número de nós de 1 para 2 e, de 2 para 3 nós. No caso dos testes que executaram operações de busca, não há melhora no desempenho a medida que nós são adicionados ao sistema, devido ao método de busca implementado pelo Hbase ser linear.*

1. Introdução

Um dos grandes desafios computacionais é o armazenamento e a recuperação de dados de forma eficiente, para que possam ser acessados rapidamente e que estejam sempre disponíveis a quem os requisite. Durante muitos anos, a solução para a maioria dos problemas dessa área foram os sistemas gerenciadores de banco de dados relacionais (SGBDR), que garantem as propriedades de atomicidade, de consistência, de isolamento e de durabilidade (ACID). Porém, a maioria dos SGBDR's não atendem mais de forma satisfatória às necessidades de certos cenários identificados com surgimento do Big Data [Brito 2010].

O conceito de Big Data refere-se ao grande volume de dados gerados em diversos domínios de aplicação, como: a Web, dispositivos móveis, sensores, motores de busca e principalmente nas redes sociais; volume este que está na ordem de 1 Zettabyte/dia [Index 2017] e, as tecnologias criadas para a manipulação desses dados garantindo escalabilidade, confiabilidade e tolerância a falhas [an et al. 2011]. A partir deste conceito, identificou-se que a tecnologia relacional possui grande fragilidade no tratamento de dados classificados como semiestruturados e não estruturados (vídeos, áudios e posts em redes sociais), por exemplo, a dificuldade em acomodá-los no modelo relacional e

a dificuldade de implementação distribuída (escalabilidade horizontal) de modo a continuar atendendo às propriedades ACID. A complexidade lógica existente na modelagem relacional somada ao volume exagerado de dados mostrou-se um problema pois pode propiciar deadlocks, problemas de concorrência e lentidão na leitura e escrita dos dados [Han et al. 2011] [Brito 2010].

Como alternativa para esses cenários, surgiram os bancos de dados denominados NoSQL. Esse termo refere-se aos bancos de dados que não utilizam linguagem SQL para consulta de dados (*Not Only SQL*), não utilizam do modelo relacional e que em maioria são distribuídos. Tal tecnologia trabalha de forma mais eficiente com grande volume de dados, possibilitando escalar horizontalmente operações por diversos servidores, replicar e distribuir esses dados, adicionar dinamicamente novos atributos aos registros, entre outras. Embora o conceito das ferramentas não relacionais tenha sido apresentado com caráter comparativo, o objetivo destas não é substituir o modelo relacional em todas as situações, apenas nos casos em que é possível adotar uma maior flexibilidade de estruturação e assim, escalonar horizontalmente o sistema de forma eficiente [Brito 2010].

No contexto dente trabalho, foi utilizado o ambiente de desenvolvimento estabelecido pelo *framework open-source Hadoop*, desenvolvido pela Fundação Apache, que reúne várias ferramentas de manipulação Big Data em um ponto central, envolvendo bancos de dados NoSQL (Hbase), um sistema de arquivos distribuídos denominado Hadoop Distributed File System (HDFS) e um modelo de programação para suportar computações paralelas em grandes coleções de dados em *clusters* de computadores chamado MapReduce [Apache Hadoop 2016].

O banco de dados Hbase, utilizado neste trabalho, permite o processamento distribuído por meio de *clusters*. Funciona sobre o HDFS, proporcionando à ferramenta capacidades parecidas ao BigTable /citechang2008bigtable e alta tolerância a falhas ao armazenar grandes quantidades de dados esparsos [Hadoop Hbase 2016].

O objetivo deste trabalho é verificar o potencial desse novo paradigma de armazenamento e recuperação de dados que utiliza de computação distribuída para criar um cenário onde aplicações e serviços continuem eficientes e disponíveis independente do tamanho da demanda de armazenamento e de consultas, apenas pela adição de novos nós ao ambiente distribuído. Dessa forma, foi avaliada a escalabilidade horizontal do banco de dados Hbase por meio de *benchmarking* com operações *read*, *write*, *scan* e *read-modify-write*, utilizando o *framework* Yahoo! Cloud Serving Benchmark (YCSB) a medida em que novos nós são adicionados ao sistema e o tamanho do conjunto de dados é aumentado.

Este trabalho está organizado como segue: A Seção 2 apresenta a fundamentação teórica necessária para a compreensão do trabalho; Na Seção 3 são citados os principais trabalhos relacionados encontrados ao proceder a revisão bibliográfica; A Seção 4 apresenta os cenários de testes utilizados; A Seção 5 exibe a análise quanto aos resultados obtidos e a Seção 6 apresenta as conclusões obtidas.

2. Fundamentação Teórica

Nessa seção são apresentados os principais conceitos necessários para uma boa compreensão do artigo.

2.1. Escalabilidade

Segundo [Elmasri and Navathe 2015], a escalabilidade é a capacidade de expandir os recursos de um sistema (capacidade de armazenamento e de processamento). Aplicando ao contexto deste artigo podemos dizer que um banco de dados escalável possui a habilidade de manipular/processar quantidades de dados cada vez maiores e, garantir a disponibilidade do sistema. Porém, existem duas abordagens para expandir um sistema as quais estão representadas na Figura 1.

2.1.1. Escalabilidade Vertical

Tradicionalmente usado pelos bancos de dados relacionais, significa adicionar mais recursos (CPU / memória RAM / disco rígido) ao servidor. Possui algumas vantagens em relação a abordagem seguinte como menor consumo de energia, menor custo com arrefecimento de servidores e geralmente são mais fáceis de implementar. Suas principais desvantagens são o custo extremamente superior a escalabilidade horizontal e, a possibilidade de interrupção do serviço sendo por falha (apenas um servidor torna-se um ponto crítico do sistema) ou quando esgotadas as possibilidades de *upgrade* da máquina atual, sendo necessário transferir o sistema para outro servidor [Hwang et al. 2014].

2.1.2. Escalabilidade Horizontal

Nesta abordagem não são inseridos mais recursos em um único servidor, mas sim inseridos mais servidores para trabalhar em paralelo. Usualmente estes servidores inseridos são máquinas simples e não muito caras chamadas de *commodity hardware* mantendo o custo muito menor que a abordagem anterior. Desta forma também há a garantia de que falhas em um ou outro servidor não causarão a interrupção do sistema pois existe redundância de dados e processos em outros nós. Essa é a maneira como a maioria dos bancos não relacionais (como o Hbase) foram projetados para aumentar sua capacidade dada uma demanda maior de armazenamento ou requisições. Embora muito mais barato que a alternativa anterior, também existem questões que precisam de atenção nessa abordagem, por exemplo, espaço físico para acomodar esses servidores, alimentação elétrica, infraestrutura de arrefecimento entre outras [Hwang et al. 2014].

2.2. Apache Hadoop

A biblioteca de softwares Apache Hadoop é um conjunto de aplicações (*framework*) que permite o processamento distribuído de um grande volume de dados em *clusters* por meio de um modelo de programação simples conhecido como MapReduce, o qual é projetado para escalar horizontalmente o processamento, oferecer alta disponibilidade e recuperação de falhas [Apache Hadoop 2016].

Há grandes empresas de diferentes ramos que o utilizam, por exemplo, a Adobe, e-Bay, Facebook, LinkedIn, The New York Time, Twitter e Yahoo. Por pertencer à Apache Software Foundation o projeto adere à licença de software Apache que é bem flexível. Por isso, existem diversas versões adaptadas da ferramenta com novas funcionalidades ou direcionadas a aplicações específicas, muitas vezes desenvolvidas por alguma das empresas citadas acima [Goldman et al. 2012].

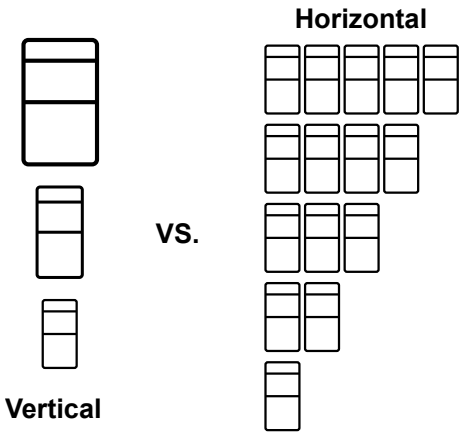


Figura 1. Escalabilidade vertical versus Escalabilidade Horizontal. Fonte: adaptado de [pc-freak.net 2017]

Um *cluster* Hadoop opera sob uma arquitetura mestre/escravo como apresentado na Figura 2. Em uma aplicação típica existem cinco processos envolvidos. O NameNode, SecondaryNameNode e o JobTraker são processos únicos para toda a aplicação sendo o NameNode e JobTracker executados pelo nó mestre e o SecondaryNameNode por um nó mestre "reserva" caso o principal falhe. O DataNode e TaskTraker são processos escravos que podem ter várias instâncias em um ou vários nós. Os processos NameNode, SecondaryNameNode e DataNode fazem parte da execução do HDFS (Hadoop Distributed File System) enquanto JobTraker e TaskTraker parte da execução MapReduce [Goldman et al. 2012].

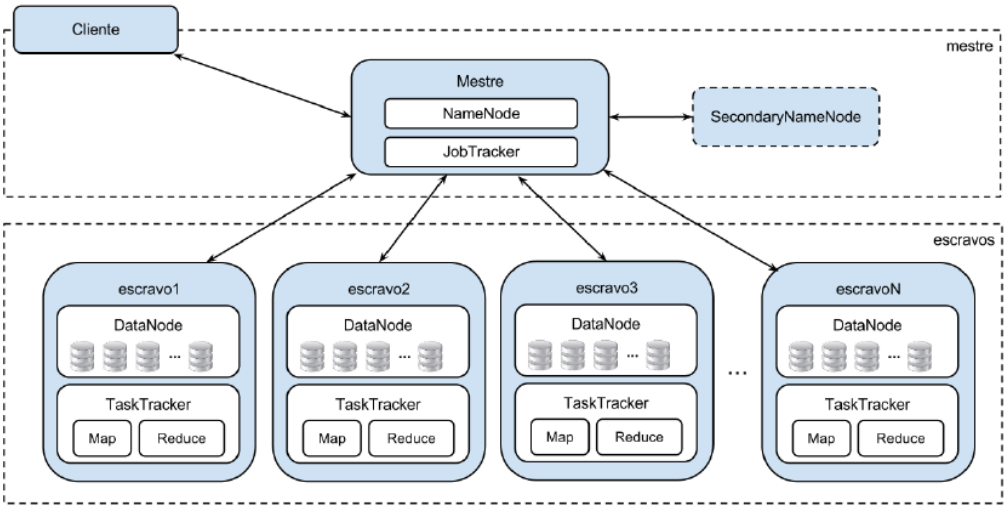


Figura 2. Representação da estrutura dos processos do Hadoop. Fonte: [Goldman et al. 2012]

2.2.1. HDFS

Sendo um dos principais componentes do *framework* Apache Hadoop, o HDFS é um sistema de arquivos distribuídos projetado para ser altamente tolerante a falhas e, operar

com hardware de baixo custo que tenham suporte à máquina virtual Java. É extremamente adequado para aplicações com grande volume de dados. Foi originalmente desenvolvido para o motor de busca Apache Nutch [Hadoop HDFS 2016].

Assim como um sistema de arquivos típico, como por exemplo os sistemas de arquivos dos sistemas operacionais, fornece operações de armazenamento, organização, nomeação, configuração de permissão de acesso, compartilhamento e recuperação de forma transparente ao usuário, ou seja, ocultando toda a complexidade. Porém a diferença do HDFS para um sistema de arquivos de um SO é que os arquivos são armazenados e compartilhados por máquinas remotas ligadas em rede.

É responsabilidade do processo NameNode abrir, fechar ou renomear arquivos e diretórios, realizar o controle de acesso dos arquivos pelos usuários, armazenar metadados e coordenar a fragmentação dos arquivos em blocos e a distribuição dos mesmos dentro dos processos DataNodes. Aos processos DataNodes cabe a tarefa de armazenas os dados de fato, responder as solicitações de leitura e escrita dos clientes ou de suas aplicações e criar, excluir ou replicar blocos sob orientação do NameNode. Além de armazenar, os DataNodes precisam se reportar constantemente ao NameNode para que o mapeamento dos blocos seja atualizado e constatar que está sem falhas [Hadoop HDFS 2016].

2.2.2. Hbase

Hbase é um banco de dados não relacional, distribuído, tolerante a falhas, de código aberto e altamente escalável. É usado principalmente em aplicações que necessitam de leitura e escrita de acesso aleatório com tempo constante sobre um grande volume de dados [Hadoop Hbase 2016].

O projeto Hbase foi inspirado pelo projeto do banco de dados BigTable da Google. Ambos utilizam o modelo de dados orientado a colunas, constituído por Namespaces (conjunto de tabelas, corresponde a uma base de dados relacional), tabelas, linhas, colunas, famílias de colunas e células [Cunha 2015]. Não possui nenhuma linguagem de consulta estruturada como o SQL, mas fornece uma API Java que possibilita realizar operações básicas como get, delete, put e update. Possibilita também o uso da função *scan*, para selecionar quais as colunas a serem retornadas ou o número de versões de cada célula. Consultas mais complexas ficam a cargo de jobs do MapReduce [Cunha 2015].

Geralmente ocorre uma confusão conceitual sobre a diferença entre o HBase e o HDFS. O HDFS é um sistema de arquivos que não fornece pesquisa de registros individuais, enquanto o HBase é executado em cima do HDFS e utiliza indexação para recuperar rapidamente os dados armazenados no HDFS [Hadoop Hbase 2016].

O HBase é organizado em uma estrutura mestre/escravo onde, na parte mestre do sistema existe o servidor *HMaster* e o ZooKeeper onde o primeiro controla os servidores escravos chamados *Region Servers* e o segundo é responsável por monitorar e manter o *cluster* ativo. *Region Servers* são responsáveis por operações de leitura e escrita [McDonald 2015].

Todos os dados do HBase são armazenados como arquivos do HDFS. Dessa forma, os *Region Servers* são dispostos nos nós que executam os processos *DataNode* do HDFS provendo localidade nos dados que transitam de um para o outro

[McDonald 2015].

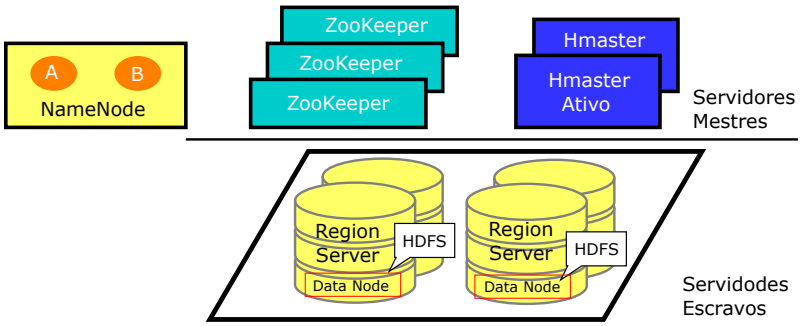


Figura 3. Estrutura do HBase. Fonte: adaptado de [McDonald 2015]

2.3. Yahoo! Cloud Serving Benchmark

O YCSB (Yahoo! Cloud Serving Benchmark) é um *framework* desenvolvido para facilitar o *benchmarking* de diferentes bancos de dados distribuídos (ou não relacionais). É altamente extensível para outros bancos de dados que ainda não estejam implementados em seu projeto e, seu código é aberto para que os desenvolvedores possam adaptá-lo como for necessário para avaliar seu sistema [Cooper et al. 2010]. O YCSB possibilita a avaliação de duas "camadas"de *benchmark*: performance e escalabilidade. Em performance, mede-se a latência (tempo de execução) das requisições enviadas ao banco e, em escalabilidade é medido o impacto na performance quando o número de servidores do sistema cresce [Cooper et al. 2010].

É composto por um conjunto de *workloads* denominado *Core Package* e uma aplicação desenvolvida em JAVA chamada *YCSB Client*. *Workloads* são combinações de operações de leitura e escrita que atuam sobre um conjunto de dados de certo tamanho e distribuição. É possível personalizar uma *workloads* alterando as combinações de operações leitura e escrita, tamanho do conjunto de dados e a distribuição dos dados no banco, a fim de tornar a execução do teste mais próxima a alguma operação do mundo real [Cooper et al. 2010].

Para executar uma *workload* , o *YCSB Client* interpreta o arquivo descritor da *workload*, gera os dados automaticamente e submete as requisições ao banco de dados. O componente que interpreta e executa as *workloads* (*workload executor*) pode ser dividido em múltiplas *threads* aumentando o número de requisições simultâneas que podem ser enviadas. As *threads* do *YCSB Client* então se comunicam com a interface do banco de dados, no qual são traduzidos os comandos descritos na *workload* para a sintaxe do banco [Cooper et al. 2010].

Existem duas fases para execução do teste: A *load phase*, onde o conjunto de dados especificado é gerado e inserido e; A *transaction phase* onde são executadas as operações definidas. Ao final do processo é gerado um relatório com a latência e o desempenho alcançado em termos de operações por segundo das tarefas executadas. A estrutura do *YCSB Client* é mostrada na Figura 4.

Utilizou-se do termo *threads* no contexto deste trabalho para se referir as *threads* do *YCSB Client*.

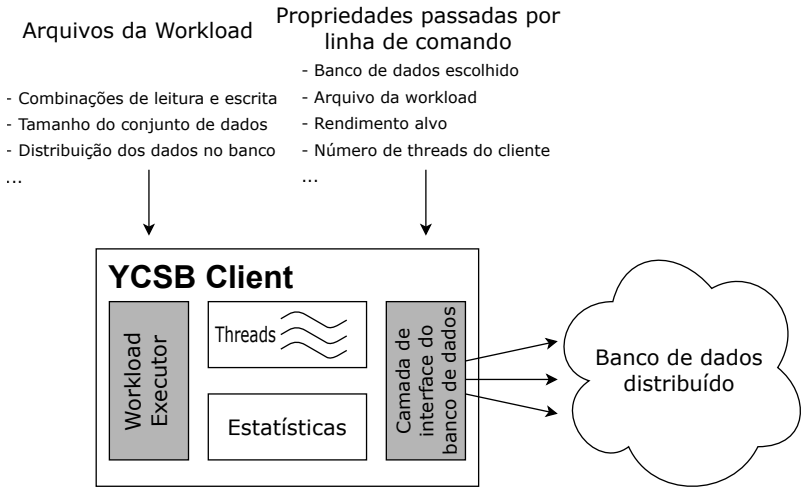


Figura 4. Estrutura do *YCSB Client*. Fonte: adaptado de [Cooper et al. 2010]

3. Trabalhos Relacionados

A partir do levantamento bibliográfico realizado para este trabalho, foram encontrados alguns trabalhos relacionados à avaliação do desempenho da ferramenta Hbase por meio de *benchmarking* com outros bancos de dados ou ainda colocando alguma condição sobre os dados a serem manipulados. Esses são descritos em sequência.

Ao apresentar o *framework* YCSB, [Cooper et al. 2010] submeteram a *benchmarking* as bases de dados não relacionais Cassandra, Hbase, Yahoo!'s PNUTS e o *sharded* MySQL para exemplificar seu uso de maneira prática. Ao realizarem os testes avaliando as camadas de performance e escalabilidade [Cooper et al. 2010] concluem que, assim como suposto pelas descrições dos desenvolvedores, Cassandra e Hbase apresentam maior latência para operações *read* e menor latência para operações *update* e **write** em relação ao PNUTS e MySQL; O PNUTS e Cassandra possuem uma escalabilidade melhor que o HBase quando o número de servidores no *cluster* aumenta proporcionalmente com a carga de trabalho. Escalabilidade esta, que daremos maior foco ao decorrer do artigo; Cassandra, Hbase e PNUTS são aptos a crescer elasticamente durante a execução de uma carga de trabalho, porém o PNUTS apresenta uma latência melhor e mais estável para tal.

Os autores [Jogi and Sinha 2016] comparam o banco de dados relacional MySQL com os bancos não relacionais Cassandra e Hbase quanto a operações *heavy write*. O teste foi realizado por meio de uma aplicação web REST (*Representacional State Transfer*) em Java que recebe dados gerados pela ferramenta *web nGrinder* em formato JSON e os salvava no banco. O desempenho de cada banco foi calculado pelo *nGrinder* em termos de TPS (Transações por segundo) ao longo de 10 minutos de testes. Chegou-se à conclusão de que o Cassandra tem o melhor desempenho entre os três bancos com a maior velocidade de escrita. O Hbase por sua vez, se mostrou aproximadamente duas vezes mais rápido que o banco relacional MySQL. Segundo [Jogi and Sinha 2016] o Cassandra apresenta tamanho desempenho para operações *heavy write* por incorporar ao mesmo tempo características do Big Table do Google e do Dynamo da Amazon.

O trabalho [Swaminathan and Elmasri 2016], prioriza a análise da escalabilidade dos bancos de dados Hbase, Cassandra e MongoDB. Para obter os resultados foi utilizado

o *framework* YCSB aplicando as cargas de trabalho 50% *read* – 50% *write*, 100% *read*, 100% *blind write*, 100% *read–modify–write* e 100% *scan*, variando o conjunto de dados a ser manipulado entre 1, 4, 10 e 40 GB a medida que foi aumentado o tamanho do *cluster* em 2, 3, 5, 6, 12 e 13 nós. O objetivo dos testes foi evidenciar as vantagens e desvantagens de cada ferramenta para um cenário específico dadas as suas diferenças de design.

De acordo com [Waage and Wiese 2014], o fato de estas diversas ferramentas não relacionais existentes oferecerem a possibilidade de armazenagem "em nuvem"e, esses ambientes não garantir a confidencialidade dos dados armazenados, é um obstaculo para uma maior adoção das mesmas. Desta forma, [Waage and Wiese 2014] propõem que os dados sejam criptografados antes de armazenados em bases de dados na nuvem e, realizam um estudo do impacto que essa alteração no conjunto de dados causa ao desempenho dos bancos de dados Cassandra e Hbase. Tal estudo foi realizado com o uso do *framework* YCSB onde as *workloads* foram aplicadas a dados não encriptados e, encriptados usando o algoritmo *Advanced Encryption Standard* (AES) com chaves de 128, 192 e 256 bits de comprimento, relatando uma redução no desempenho médio do *cluster* e que, esse custo é relativamente o mesmo independente do tamanho da chave de encriptação.

4. Abordagem de comparação

Os testes realizados nesse trabalho foram executados em um *cluster* com sete computadores, sendo o nó mestre denominado hpcdmc e os seis nós escravos denominados de n01 a n06, cujas configurações estão descritas abaixo.

Configuração do mestre (hpcdmc):

- Sistema Operacional Linux CentOS 6.6
- 2x Processador Intel Xeon E5-2620 2.0GHz 6 núcleos
- 4x Memória Ram Kingston DDR3 8GB 1333MHz
- 8x Sata3 de 2TB

Configuração dos nós escravos e cliente (n01 a n06):

- Sistema Operacional Linux CentOS 6.6
- 2x Processador Intel Xeon E5-2690 2.90GHz 8 núcleos
- 4x Memória Ram Kingston DDR3 8GB 1600MHz
- 8x Sata3 de 500GB e 16MB de cahce

As versões dos softwares utilizados para a execução dos testes foram escolhidas por serem as mais atuais e estáveis[1] consultadas no site dos desenvolvedores no início do trabalho. São elas:

- Hadoop 2.7.3
- HBase 1.2.4
- ZooKeeper 3.4.10
- YCSB 0.12.0

Os testes foram organizados em seis cenários onde, os testes executados nos cinco primeiros cenários analisam a melhor alternativa para a implementação do *cluster*, ou seja, como serão dispostos os processos cliente, mestre e escravo. Nesses foram executados os testes em um ambiente pseudo distribuído, com apenas duas *workloads*: 100% *write*

[1] Versões mais atuais, mas ainda em fase beta não foram consideradas

– 100.000 registros e 100% *read* – 100.000 registros, tendo cada registro 1KB e com distribuição uniforme. Cada teste sobre uma *workload* foi executado 3 vezes para uma dada quantidade de *threads* do *YCSB Client* e então foi realizada a média da latência (tempo de execução em milissegundos) e do desempenho médio (operações/segundo) para obter o resultado final do teste.

Nos testes executados no sexto cenário, configurado de acordo com as conclusões obtidas nos testes dos cenários anteriores, é analisada a escalabilidade do HBase em um *cluster* totalmente distribuído, sendo esse, o foco principal do trabalho. Cada teste sobre uma *workload* também foi executado 3 vezes sendo o resultado final a média das saídas.

4.1. Cenário 1

No cenário 1 o *cluster* é configurado de modo pseudo distribuído, ou seja, os processos são executados como se houvesse uma distribuição entre máquinas em redes, porém estão todos no mesmo nó. Dessa forma, os processos mestre e escravo do Hadoop e HBase, os processos do ZooKeeper e do *YCSB Client* são executados todos no hpcdmc. A distribuição dos processos em um cluster pseudo distribuído é representada pela Figura 5.

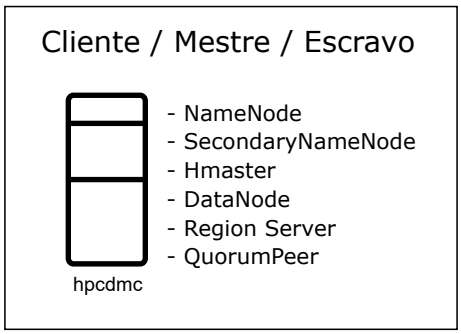


Figura 5. Configuração do cenário de testes 1. Fonte: o autor

4.2. Cenário 2

No cenário 2, considerando que a execução de todos os processos mestre e escravo junto com o *YCSB Client* pode comprometer a quantidade de requisições enviadas ao banco, alocou-se o YCSB em um nó separado para verificar essa hipótese. Dessa forma, os processos meste e escravo do Hadoop e HBase e os processos do ZooKeeper continuaram a ser executados no hpcdmc, mas o processo *YCSB Client* foi executado no nó n01 conforme é apresentado na Figura 6.

4.3. Cenário 3

Dada a limitação do número de *threads* do *YCSB Client* que podem ser executadas no n01, pelo número de núcleos do processador, no cenário 3, a execução da *workload* foi dividida entre nós n01 e n02. Assim, cada nó foi encarregado pela metade dos registros e metade do número de *threads* do *YCSB Client* total do teste. Exemplo: na execução da *workload* 100% *read* – 100.000 registros e 64 *threads*, cada nó gerou 50.000 requisições *read* com 32 *threads* ativas no YCSB. A saída de cada teste foi a soma do desempenho médio (operações/segundo) e a maior latência (tempo de execução em milissegundos) de ambos. O hpcdmc permaneceu pseudo distribuído conforme é apresentado na Figura 7.

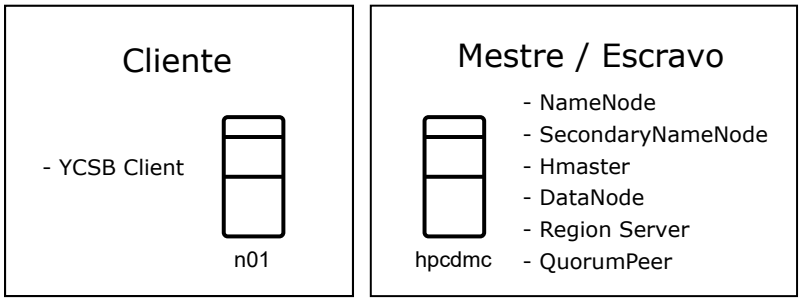


Figura 6. Configuração do cenário de testes 2. Fonte: o autor

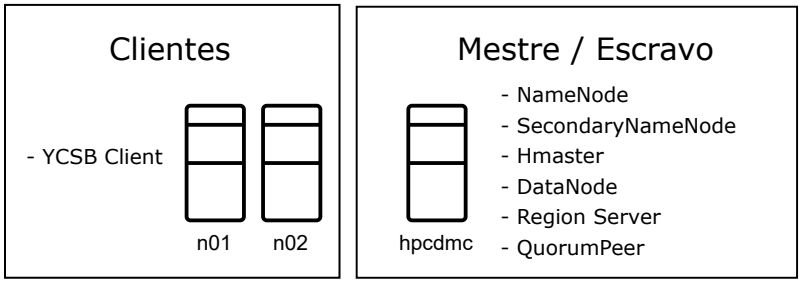


Figura 7. Configuração do cenário de testes 3. Fonte: o autor

4.4. Cenário 4

O cenário 4 se aproxima de uma situação mais adequada quanto a configuração do *cluster*, onde os processos cliente, mestre e escravo estão completamente distribuídos. O n01 é responsável pelo *YCSB Client*, o hpcdmc pelos processos mestre do Hadoop, HBase e dos processos do ZooKeeper, enquanto o n02 executou os processos escravos do Hadoop e HBase conforme é apresentado na Figura 8.

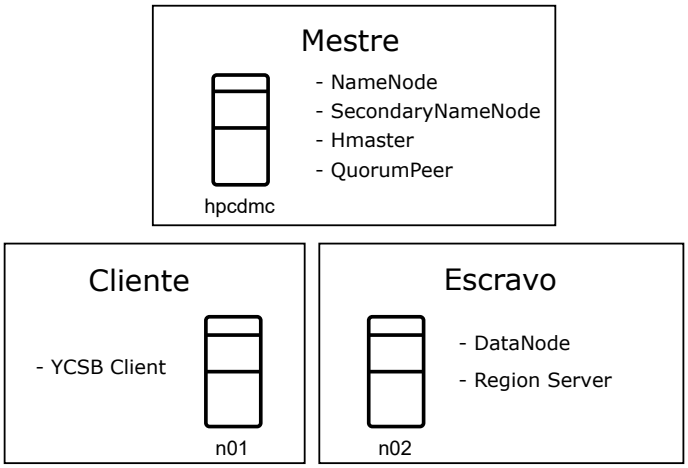


Figura 8. Configuração do cenário de testes 4. Fonte: o autor

4.5. Cenário 5

No cenário 5, o objetivo dos testes é analisar se é necessário mais de um servidor gerando requisições para saturar um *cluster* completamente distribuído. Assim, o n01 e n02 dividiram a execução das *workloads*, o hpcdmc foi responsável pelos processos mestre do

Hadoop, HBase e os processos do ZooKeeper e o n03 executou os processos escravos do Hadoop e HBase conforme é apresentado na Figura 9.

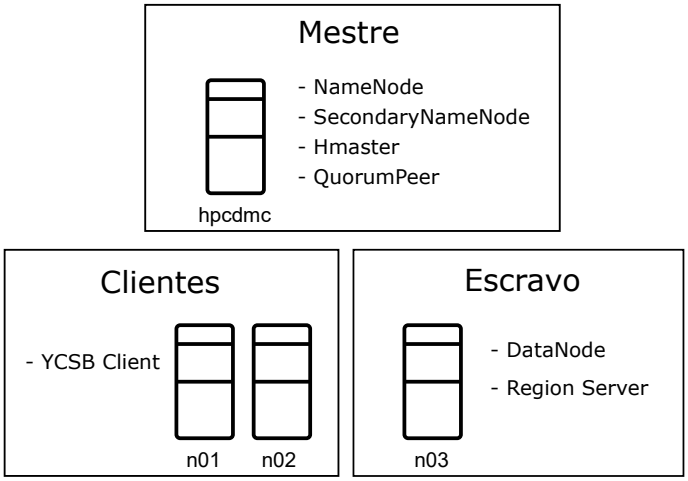


Figura 9. Configuração do cenário de testes 5. Fonte: o autor

4.6. Cenário 6

Os resultados dos testes realizados nos cenários anteriores apoiam as decisões quanto a estrutura geral do cenário 6 ilustrado na Figura 10. Foram executadas as *workloads*: *write*, *read* e *scan*[2], variando entre 1.000, 10.000, 100.000 e 1.000.000 registros com tamanho igual a 1KB e com distribuição uniforme, variando o número de escravos de 1 a 5 e o número de *threads* do *YCSB Client* fixo.

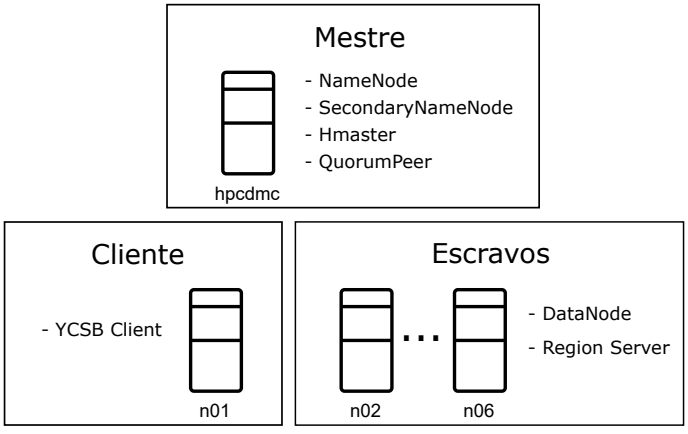


Figura 10. Configuração do cenário de testes 6. Fonte: o autor

5. Resultados

Considerando os resultados obtidos nos cenários de 1 a 5, foi realizada uma análise e configurado o *cluster* da melhor forma possível de acordo com os computadores disponíveis para explorar da melhor forma a disposição dos processos e executar o *cluster* com todo o seu potencial. Após a montagem do *cluster* ideal foram realizados os testes no cenário

[2]Scan: Operação de busca por determinado registro no banco de dados

6 cuja a análise dos resultados relatou o impacto da adição de nós ao *cluster* e o aumento do conjunto de dados, no desempenho do banco de dados Hbase considerando o desempenho médio (número de operações executadas por segundo) e a latência do teste (tempo de execução em milissegundos).

5.1. Resultados dos testes dos cenários de 1 a 5

Nos resultados obtidos nos testes dos cenários 1 e 2 existe um crescimento no desempenho médio do *cluster* nas execuções com até 64 *threads*, como ilustram as Figuras às 11 e 13. Para os testes do cenário 1, de 1 para 64 threads ocorreu um aumento de aproximadamente 90% no desempenho médio e, nos testes do cenário 2 ocorreu, para a mesma quantidade de *threads*, um aumento de aproximadamente 93%.

Ao instanciar mais que 64 *threads*, no cenário 1 ocorreu uma queda no desempenho médio de aproximadamente 5% nos testes com 128 *threads* e, no cenário 2 ocorreram quedas nas execuções de 128, 256 e 512 *threads*, sendo a mais expressiva de aproximadamente 10% nas execuções com 256 *threads*. Ambas as porcentagens foram calculadas comparando os resultados às execuções com 64 *threads*.

Da mesma forma, existe uma queda na latência nos testes dos cenários 1 e 2 até as execuções com 64 *threads*, como ilustram as Figuras 12 e 14. Isso é claro, dado que a medida que o desempenho médio aumenta, ou seja, o *cluster* executa mais operações por segundo, o tempo de execução do teste diminui. No cenário 1, a queda da latência entre as execuções com 1 e 64 *threads* foi de aproximadamente 90% e, para o cenário 2 de aproximadamente 93%.

Assim, constatou-se que o desempenho máximo do *cluster* foi obtido para esses dois cenários nas execuções do *YCSB Client* com 64 *threads* e, a partir do cenário 3 foram executados apenas os testes de 64 a 512 *threads* afim de constatar se esses resultados foram ótimos locais ou globais. Os testes do cenário 1, para 256 e 512 *threads* não puderam ser executados por estouro da pilha de memória do nó hpcdmc ao executar o *YCSB Client* ja que, além do *YCSB Client* o nó hpcdmc também estava executando os processos mestre e escravo do Hbase e Hadoop e os processos do ZooKeeper.

A análise dos resultados obtidos pela execução testes com a *workload* 100% *write* – 100.000 registros, mostrou que o melhor desempenho médio foi alcançado pelo cenário 5 para as execuções com 64 e 128 *threads*, sendo maior que os resultados dos cenários 4, 3, 2 e 1 aproximadamente 5,7%, 19,4%, 25,4% e 50,3% nas execuções com 64 *threads* e, 8,2%, 27,4%, 28,6% e 54% nas execuções com 128 *threads*, respectivamente. Para as execuções com 256 e 512 *threads* o melhor desempenho médio foi obtido pelo cenário 4 sendo maior que os cenários 5, 3 e 2 aproximadamente 0,8%, 22,6% e 35,6% nas execuções com 256 *threads* e, 58%, 19,9% e 21,2% nas execuções com 512 *threads* respectivamente.

O cenário 5 apresentou tal desempenho pois o *cluster* foi completamente distribuído, então os processos mestres e escravo do Hbase e Hadoop, os processos do ZooKeeper e YCSB não concorreram por recursos de uma mesma máquina uns com os outros. O mesmo ocorreu no cenário 4, mas a divisão da execução das *workloads* entre o n01 e n02 apresentou uma pequena melhora no cenário 5 no desempenho do *cluster* executando o YCSB com 64 e 128 *threads*, já nas execuções com 256 e 512 *threads* tal divisão sobrecarregou demais o n03 fazendo com que o desempenho médio diminuísse e a latência

aumentasse, tornando os resultados dos testes do cenário 4 melhores.

Analisando os testes da *workload* 100% *write* – 100.000 registros, pode-se concluir que o cenário 5, para execuções com 64 e 128 *threads* teve uma execução mais rápida que os testes dos cenários 4, 3, 2 e 1 aproximadamente 5,6%, 19,7%, 25,4% e 50,2% com 64 *threads* e, 7,8%, 27%, 28,1% e 53,7% com 128 *threads* respectivamente. Enquanto que para as execuções com 256 e 512 *threads* o cenário 4 teve uma execução mais rápida que os testes dos cenários 5, 3 e 2 aproximadamente 2,4%, 22,8% e 35,8% com 256 *threads* e, 95,8%, 20,3% e 21,5% com 512 *threads*. Os resultados são apresentados na Figura 12.

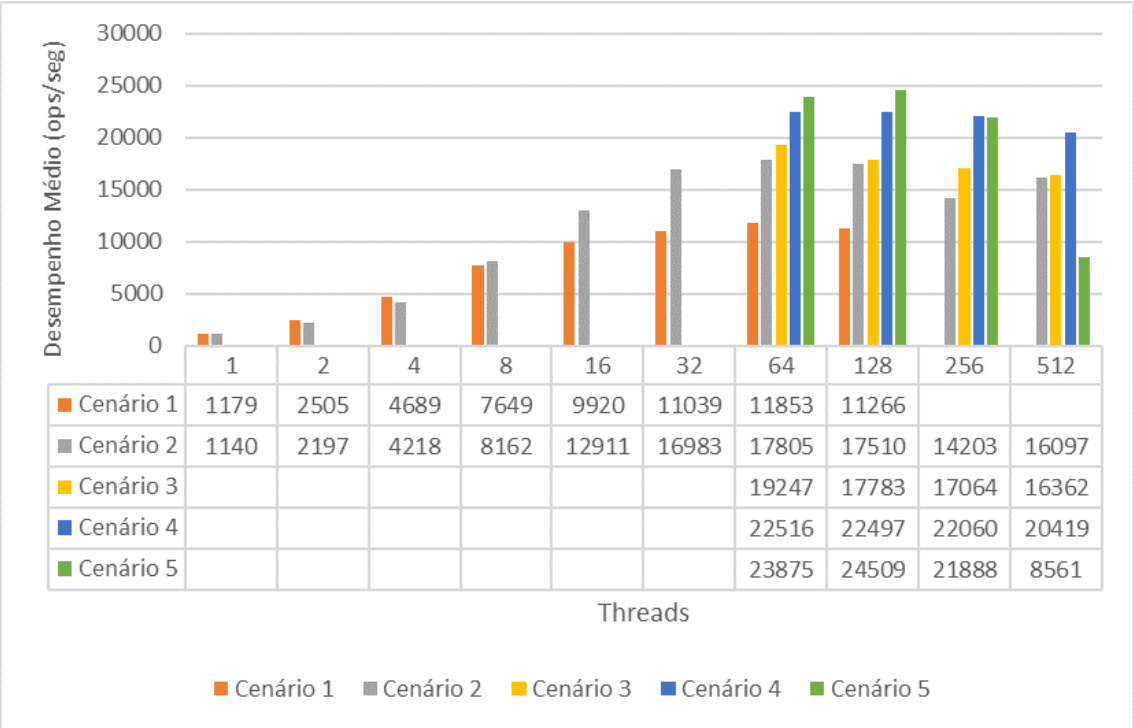
Analisando os resultados dos testes da *workload* 100% *read* – 100.000 registros, para todos os números de *threads* o cenário 5 obteve o melhor desempenho médio sendo maior que os resultados dos testes dos cenários 4, 3, 2 e 1 aproximadamente 26,8%, 22,2%, 48,3% e 62,6% nas execuções com 64 *threads* e, 26,9%, 44,3%, 45,7% e 64,4% nas execuções com 128 *threads* respectivamente. E maior que os resultados dos testes dos cenários 4, 3 e 2 aproximadamente 28,4%, 23% e 50,4% nas execuções com 256 *threads* e, 26,5%, 18,2% e 47,3% nas execuções com 512 *threads* respectivamente.

Para as operações de leitura especificadas na *workload* 100% *read*, a divisão de requisições entre o n01 e n02 não sobrecarregou o n03 para execuções com 256 e 512 *threads* como nos testes da *workload* 100% *write*, então o cenário 5 obteve os melhores resultados para cada execução entre 64 e 512 *threads*. Os testes do cenário 3, que também dividiram as requisições entre o n01 e n02 obtiveram os segundos melhores resultados para cada execução entre 64 e 512 *threads*. Portanto para operações de leitura, o *cluster* é mais eficiente, considerando o desempenho médio e a latência, se as requisições são realizadas de mais de um cliente. Os resultados são ilustrados na Figura 13.

Na *workload* 100% *read* – 100.000 registros, cujo os dados são apresentados pela Figura 14, a latência dos testes executados no cenário 5 foi menor que os cenários 4, 3, 2 e 1 aproximadamente 26,2%, 22,8%, 47,9% e 62,3% nas execuções com 64 *threads* e, 26,6%, 23,6%, 45,6% e 64,3% nas execuções com 128 *threads* respectivamente. E também, menor que os cenários 4, 3 e 2 aproximadamente 27,9%, 22,8% e 50% nas execuções com 256 *threads* e, 25,5%, 17,3% e 46.6% nas execuções com 512 *threads* respectivamente.

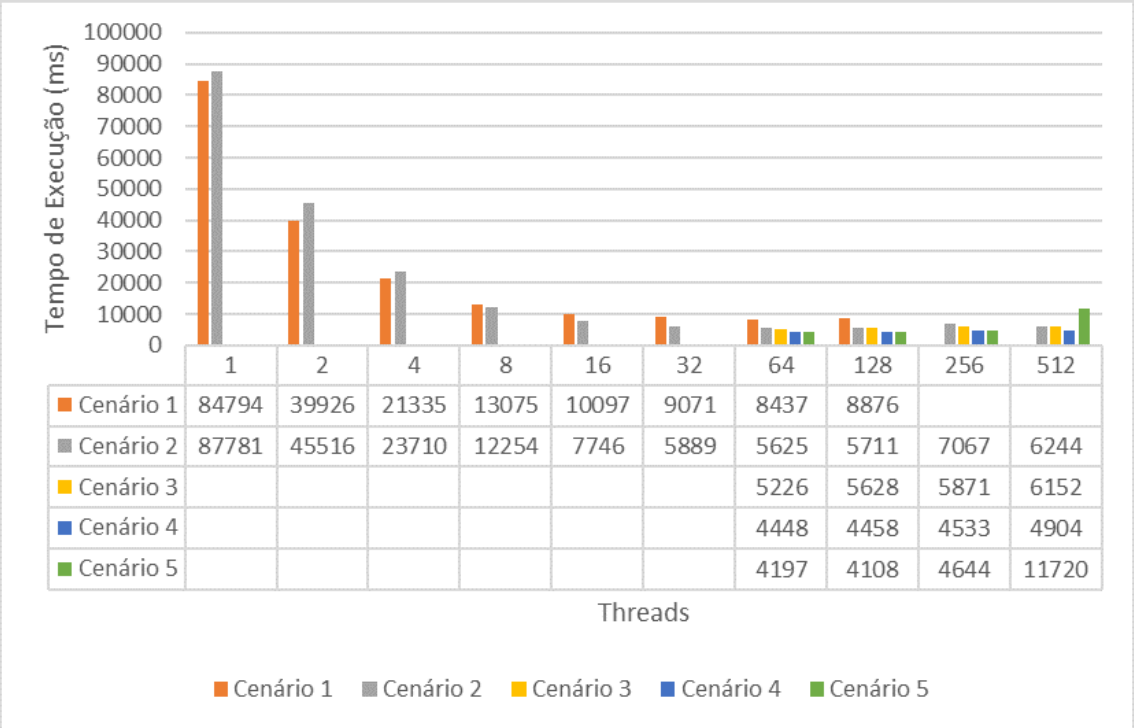
Dos resultados obtidos observou-se que, para todos os cenários as execuções com 64 e 128 *threads* são as mais altas e a variação referente a um mesmo cenário, considerando essas duas quantidades de *threads* não são expressivas, sendo de aproximadamente 1%, para os testes da *workload* 100% *write* e 100% *read*, tando para o desempenho médio quanto para a latência.

O aumento do número de *threads* além de 128 causou uma queda do desempenho médio nos testes dos cenários 5, 4, 3 e 2 de até 65%, 9,3%, 15% e 9,6% para os testes da *workload* 100% wirte e, 10,9%, 10,4%, 5,3 e 13,5% para os testes da *workload* 100% *read* respectivamente. Comparando então os 5 cenários onde a execução do *YCSB Client* ocorreu com 64 *threads* temos que em ambas as *workloads* 100% *write* e 100% *read* o cenário 5 obteve o maior desempenho médio do *cluster*, seguido pelo cenário 4 na *workload* 100% *write* com uma diferença de aproximadamente 5,7% e, seguido pelos cenários 3 e 4 na *workload* 100% *read* com uma diferença de aproximadamente 22,2% e



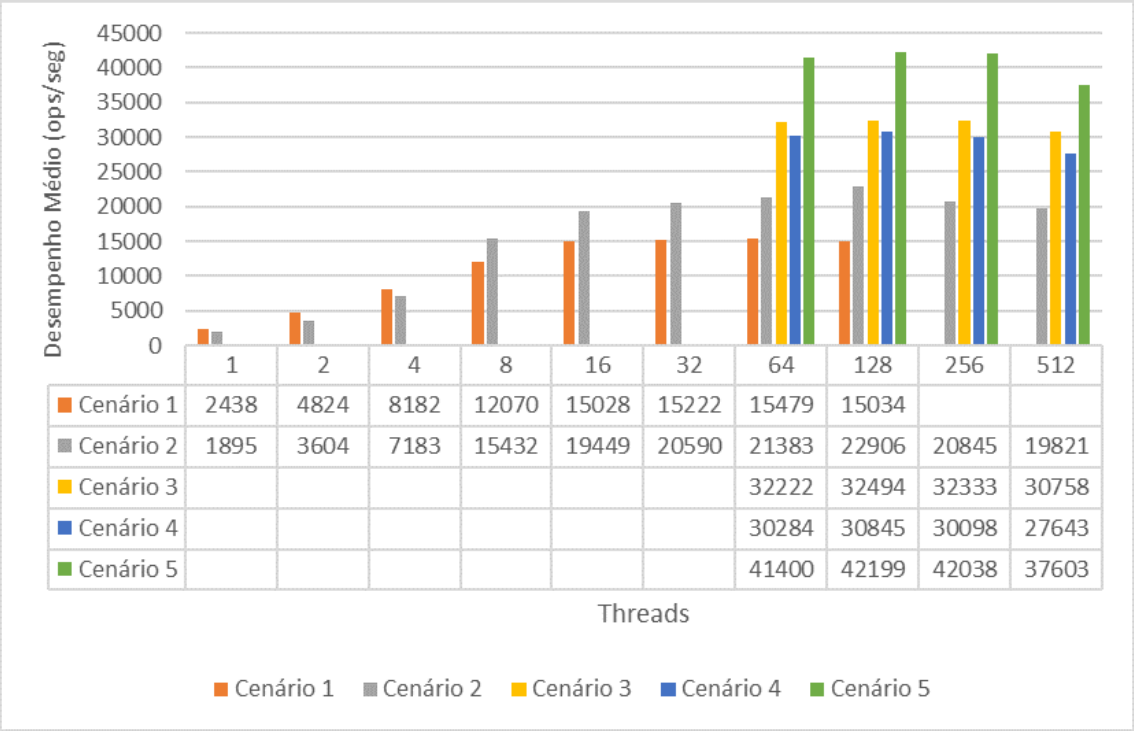
	1	2	4	8	16	32	64	128	256	512
Cenário 1	1179	2505	4689	7649	9920	11039	11853	11266		
Cenário 2	1140	2197	4218	8162	12911	16983	17805	17510	14203	16097
Cenário 3							19247	17783	17064	16362
Cenário 4							22516	22497	22060	20419
Cenário 5							23875	24509	21888	8561

Figura 11. Desempenho médio da *workload* 100% *write* – 100.000 registros nos cenários de 1 a 5. Fonte: o autor



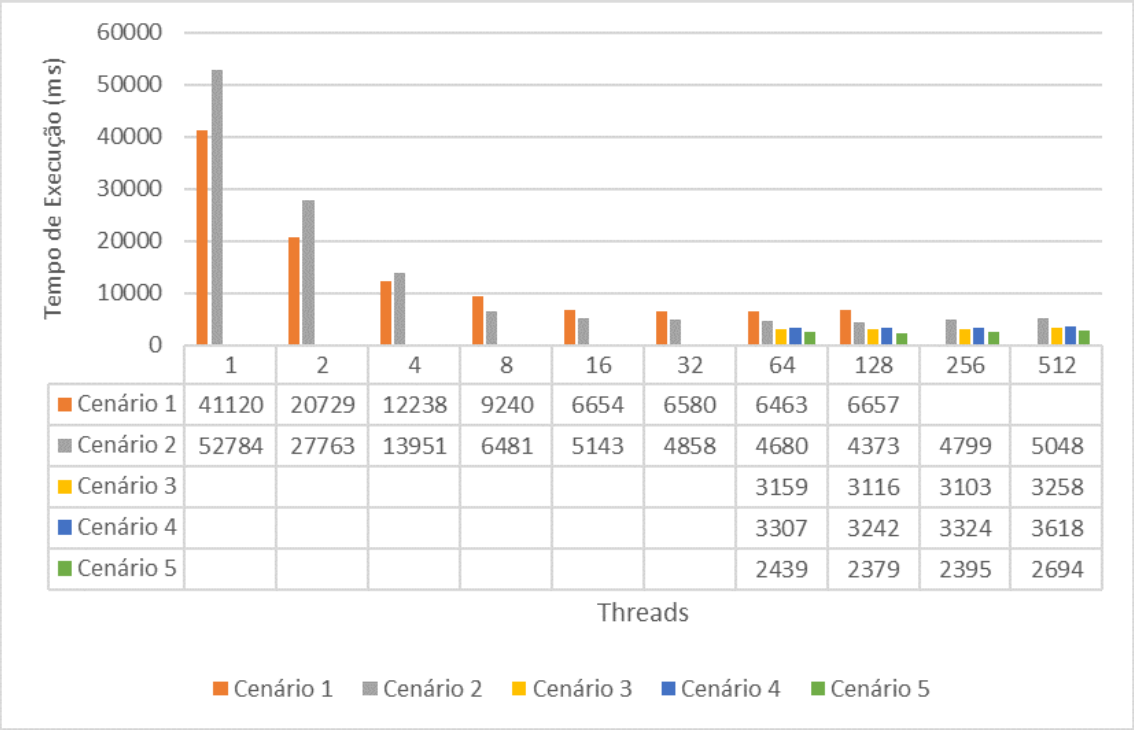
	1	2	4	8	16	32	64	128	256	512
Cenário 1	84794	39926	21335	13075	10097	9071	8437	8876		
Cenário 2	87781	45516	23710	12254	7746	5889	5625	5711	7067	6244
Cenário 3							5226	5628	5871	6152
Cenário 4							4448	4458	4533	4904
Cenário 5							4197	4108	4644	11720

Figura 12. Latência da *workload* 100% *write* – 100.000 registros nos cenários de 1 a 5. Fonte: o autor



	1	2	4	8	16	32	64	128	256	512
Cenário 1	2438	4824	8182	12070	15028	15222	15479	15034		
Cenário 2	1895	3604	7183	15432	19449	20590	21383	22906	20845	19821
Cenário 3							32222	32494	32333	30758
Cenário 4							30284	30845	30098	27643
Cenário 5							41400	42199	42038	37603

Figura 13. Desempenho médio da *workload* 100% *read* – 100.000 registros nos cenários de 1 a 5. Fonte: o autor



	1	2	4	8	16	32	64	128	256	512
Cenário 1	41120	20729	12238	9240	6654	6580	6463	6657		
Cenário 2	52784	27763	13951	6481	5143	4858	4680	4373	4799	5048
Cenário 3							3159	3116	3103	3258
Cenário 4							3307	3242	3324	3618
Cenário 5							2439	2379	2395	2694

Figura 14. Latência da *workload* 100% *read* – 100.000 registros nos cenários de 1 a 5. Fonte: o autor

26,8% respectivamente.

Levando em consideração que os testes do cenário 4 obtiveram o segundo maior desempenho médio nos testes da *workload* 100% *write* com uma diferença de menos de 6% comparado aos resultados do cenário 5, obtiveram o terceiro maior desempenho médio nos testes da *workload* 100% *read* com uma diferença de aproximadamente 6% comparado com os resultados cenário 3 e, considerando também que ambos os cenários 5 e 3 utilizam 2 nós para a execução do *YCSB Client*, reduzindo o número de nós disponíveis para os testes de adição de nós, a implementação do *cluster* para os testes do cenário 6 foi realizada de acordo com o cenário 4 e com 64 *threads* de execução no *YCSB Client*.

5.2. Resultados dos testes do cenário 6

Pela análise dos resultados obtidos dos testes da *workload* 100% *write* referente ao desempenho médio pode-se notar que a escalabilidade horizontal foi mais evidenciada quando o conjunto de dados foi igual a 1.000.000 de registros, sendo o desempenho médio do *cluster* com 5 *region servers* maior do que os testes executados com 4, 3, 2, e 1 *region servers* em aproximadamente 5,1%, 6,5%, 17,1% e 47% respectivamente. Os resultados dos testes em que o conjunto de dados foi igual a 100.000 registros também apresentou certa escalabilidade, sendo o desempenho médio com 5 *region servers* maior do que o desempenho médio dos testes executados com 4, 3, 2, e 1 *region servers* em aproximadamente 9,9%, 10,4%, 15,7% e 43,4% respectivamente.

A partir da análise dos resultados dos testes da *workload* 100% *write*, ilustrados pela Figura 16, pode-se observar que os testes que o conjunto de dados foi igual a 1.000.000 de registros obtiveram uma latência menor que os testes com 4, 3, 2 e 1 *region server* em aproximadamente 5%, 6,3%, 17,1% e 47% respectivamente. Os resultados dos testes com 100.000 registros, onde a escalabilidade foi menos evidenciada, as latências tiveram poucas alterações comparados os resultados dos testes com 4, 3, 2 e 1 *region server*, sendo menor em aproximadamente 12,5%, 11,7%, 15,9% e 43,4% respectivamente.

Os testes em que os conjuntos de dados foram iguais a 1.000 e 10.000 registros não tiveram alterações tão significativas no desempenho médio com a adição de novos nós, sendo a variação mais expressiva para o conjunto de dados de 1.000 registros de aproximadamente 4% aumentando de 1 para 2 *region servers* e, de aproximadamente 8% para o conjunto de dados de 10.000 aumentando de 1 para 2 *region servers*, considerando tanto o desempenho médio quanto a latência como ilustrado pelas Figuras 17 e 18.

Pela análise dos resultados dos testes da *workload* 100% *read*, assim como nos resultados acima, nota-se que a escalabilidade horizontal foi mais evidenciada quando o conjunto de dados foi igual a 1.000.000 de registros. Porém nas execuções com 5 *region servers* só existiu alteração significativa no desempenho médio quando comparadas com as execuções com 2 e 1 *region servers*, sendo maior em aproximadamente 10,4% e 39,1% respectivamente. Para as execuções com 4 e 3 *region servers* a variação foi de menos de 1,5% como apresenta a Figura 18.

Pela análise dos resultados dos testes com 100.000 registros nota-se o mesmo comportamento, sendo o desempenho médio das execuções com 5 *region servers* maior que os testes com 2 e 1 *region servers* aproximadamente 5,4% e 29% respectivamente. Para as execuções com 4 e 3 *region servers* a variação foi de menos de 2,4%.

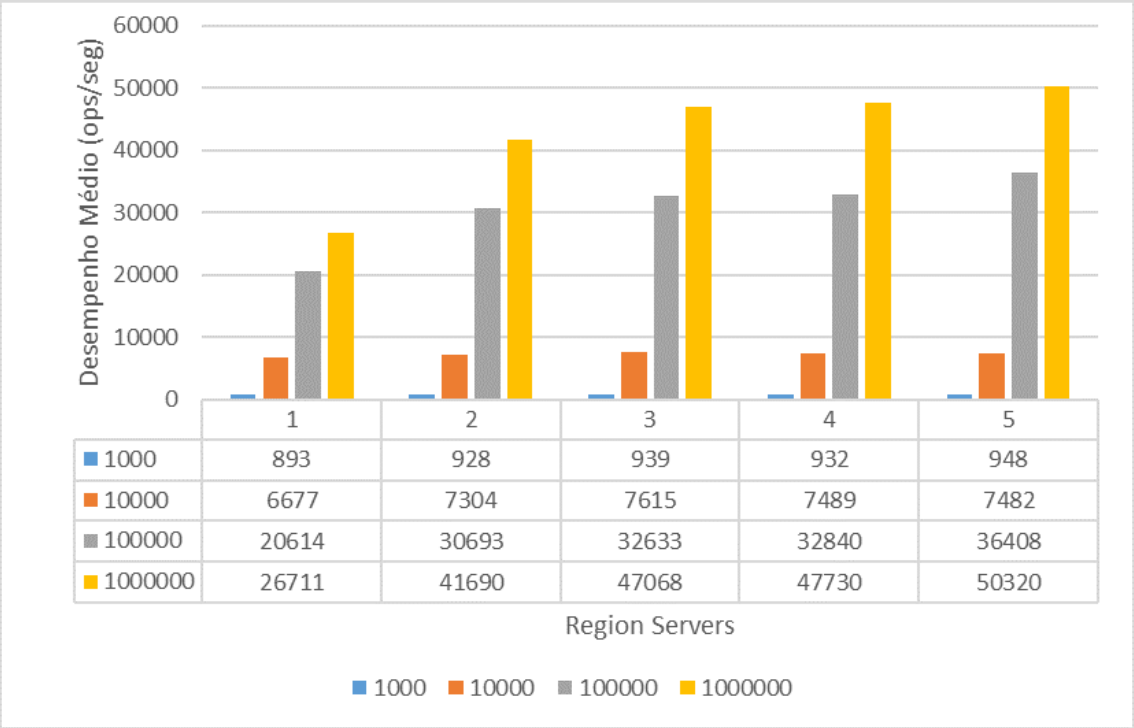


Figura 15. Desempenho médio da *workload* 100% *write* variando o tamanho do conjunto de dados e o número de *region servers* do *cluster*. Fonte: o autor

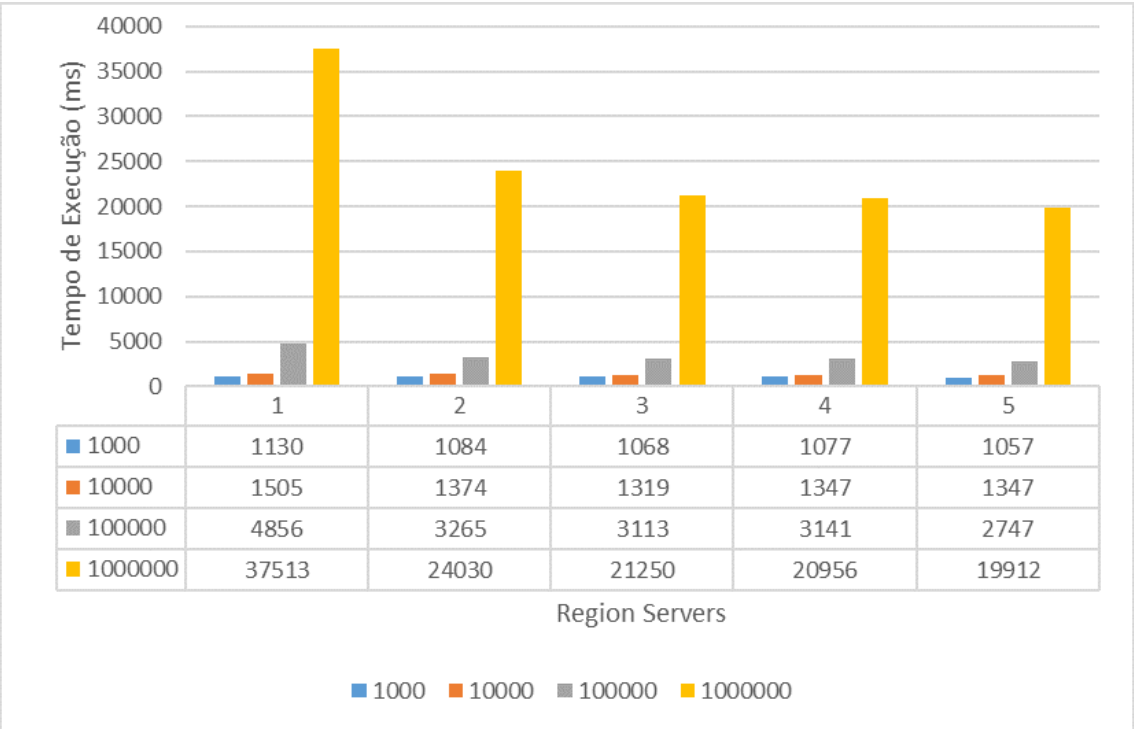


Figura 16. Latência da *workload* 100% *write* variando o tamanho do conjunto de dados e o número de *region servers* do *cluster*. Fonte: o autor

Os testes em que os conjuntos de dados foram iguais a 1.000 e 10.000 registros não tiveram alterações significativas, sendo a variação mais expressiva para o conjunto de dados de 1.000 registros de menos de 1,5% aumentando de 1 para 2 *region servers* e para o conjunto de 10.000 registros de menos de 7,5% também aumentando de 1 para 2 *region servers* .

A latência dos testes da *workload* 100% *read* como pode-se analisar nos dados apresentados pela Figura 18, também não apresentou alterações significativas para os testes cujos conjuntos de dados foram iguais 1.000.000 e 100.000 registros quando variados os *region servers* entre 3 e 5, sendo essa variação de menos de 1,3% para ambos os conjuntos. Desta forma, os testes executados com 5 *region servers* apresentaram uma latência menor que os testes com 2 e 1 *region servers* em aproximadamente 10,5% e 40,2% com 1.000.000 de registros e, em aproximadamente 5,4% e 29% com 100.000 registros respectivamente.

Os testes em que os conjuntos de dados foram iguais a 1.000 e 10.000 registros não tiveram alterações significativas no desempenho médio, sendo a variação de 1 para 5 *region servers* de menos de 3,3% para o conjunto de 1.000 registros e, de menos de 10,7% para o conjunto de 10.000 registros.

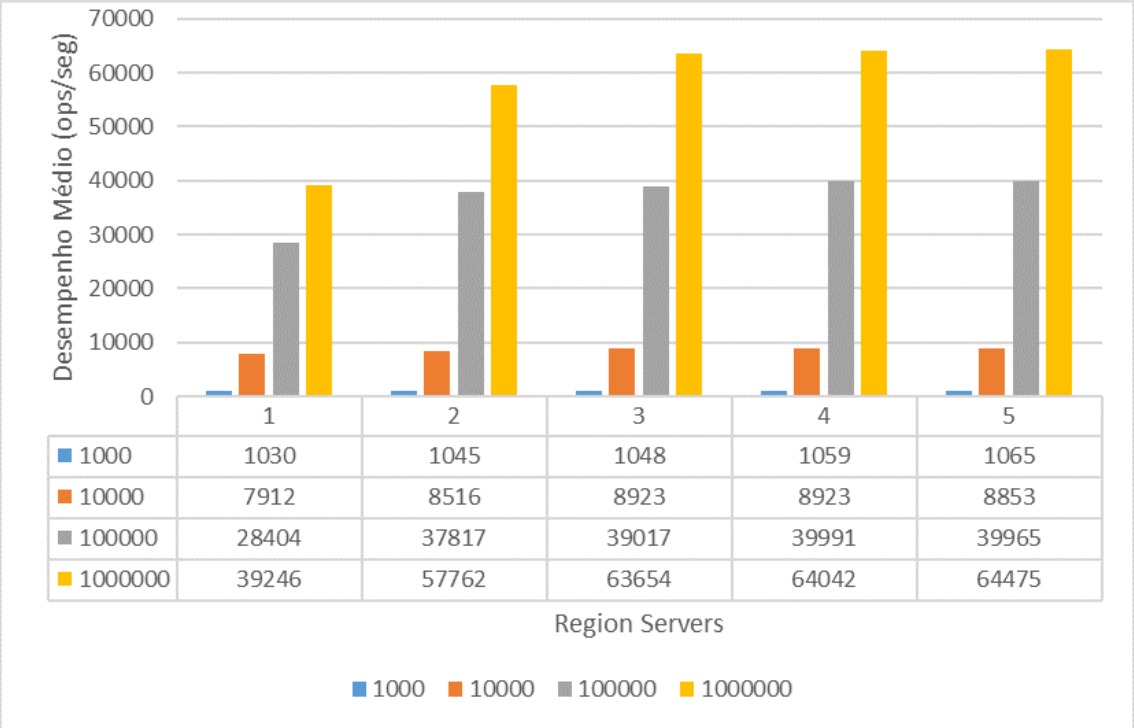
Pela análise dos resultados quanto ao desempenho médio dos testes da *workload* 100% *read/modify/write*, observou-se novamente que, a escalabilidade horizontal é mais evidenciada nos cenários onde o conjunto é igual a 1.000.000 de registros, sendo o desempenho médio dos testes com 5 *region servers* maior que os resultados dos testes com 4, 3, 2 e 1 *region servers* aproximadamente 4,5%, 14,5%, 29,7% e 58,6% respectivamente. Para os testes onde os conjuntos de dados foram iguais a 100.000 registros obteve-se nas execuções com 5 *region servers* um desempenho médio maior que os testes com 4, 3, 2 e 1 *region server* de aproximadamente 5,6%, 7,1%, 20,1% e 47,7% respectivamente.

Os testes onde os conjuntos de dados foram iguais a 1.000 registros não tiveram alterações significativas no desempenho médio, sendo a variação mais expressiva de menos de 6% aumentando de 1 para 2 *region servers*. Para o conjunto de 10.000 registro, a variação mais expressiva foi de aproximadamente 18,5% quando aumentado de 1 para 2 *region servers*. As demais adições de nós apresentaram uma variação de menos de 4%.

A Figura 20 apresenta os resultados dos testes da *workload* 100% *read/modify/write* quanto a latência. Novamente, o comportamento da latência acompanha o comportamento do desempenho médio de maneira inversamente proporcional e portanto, os testes com 1.000.000 de registros e 5 *region servers* obtiveram uma latência menor que os testes com 4, 3, 2 e 1 *region servers* em aproximadamente 4,4%, 14,4%, 29,9% e 58,7% respectivamente. Do mesmo modo, os testes com 100.000 registros e 5 *region servers* obtiveram uma latência menor que os testes com 4, 3, 2 e 1 *region servers* em aproximadamente 2,6%, 7,7%, 20,1% e 47,9% respectivamente.

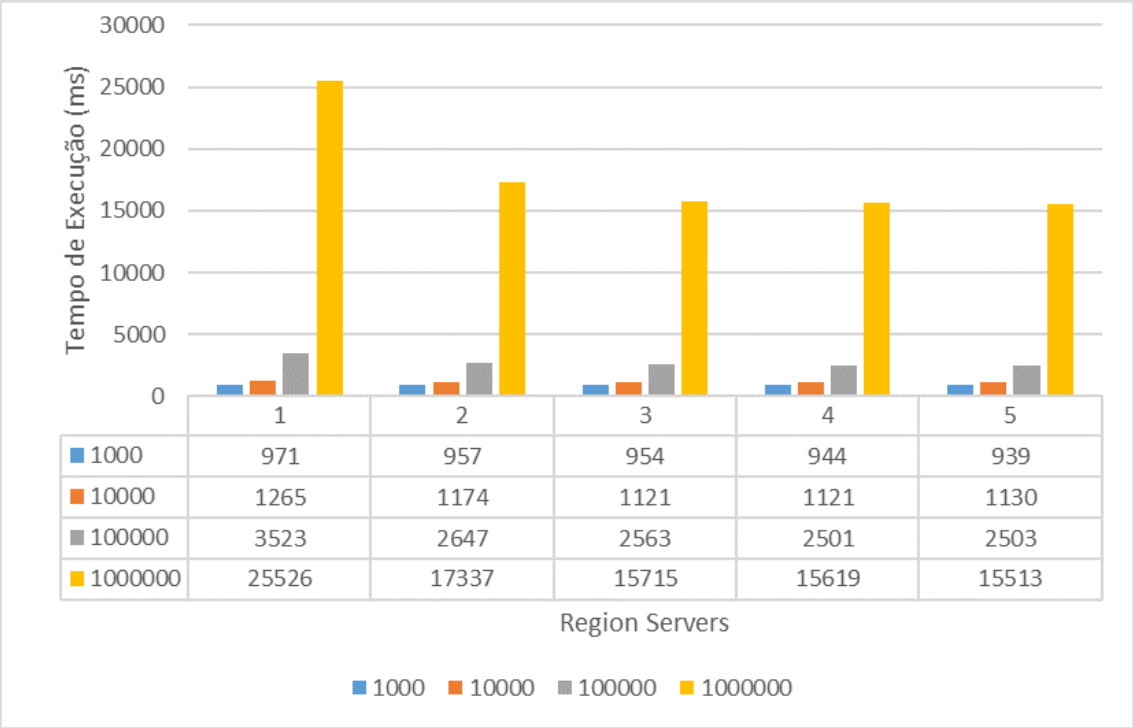
Os testes da *workload* 100% *read/modify/write* em que os conjuntos de dados foram iguais a 1.000 e 10.000 registros não tiveram alterações significativas na latência, seguindo a mesma porcentagem calculada nos testes de desempenho médio.

Os resultados dos testes da *workload* 100% *scan* referentes ao desempenho médio e latência são apresentados nas Figuras 21 e 22. Exceto nos testes cujos conjuntos de dados foram iguais a 1.000 registros, não houveram alterações significativas no desempe-



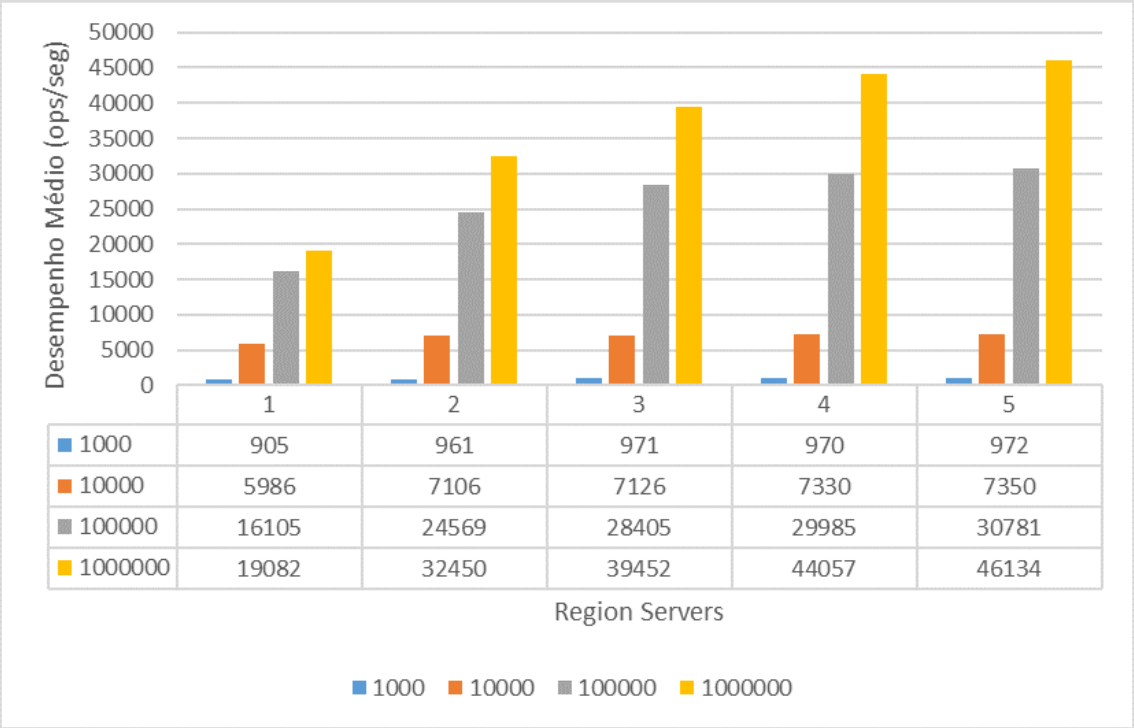
	1	2	3	4	5
1000	1030	1045	1048	1059	1065
10000	7912	8516	8923	8923	8853
100000	28404	37817	39017	39991	39965
1000000	39246	57762	63654	64042	64475

Figura 17. Desempenho médio da *workload* 100% *read* variando o tamanho do conjunto de dados e o número de *region servers* do *cluster*. Fonte: o autor



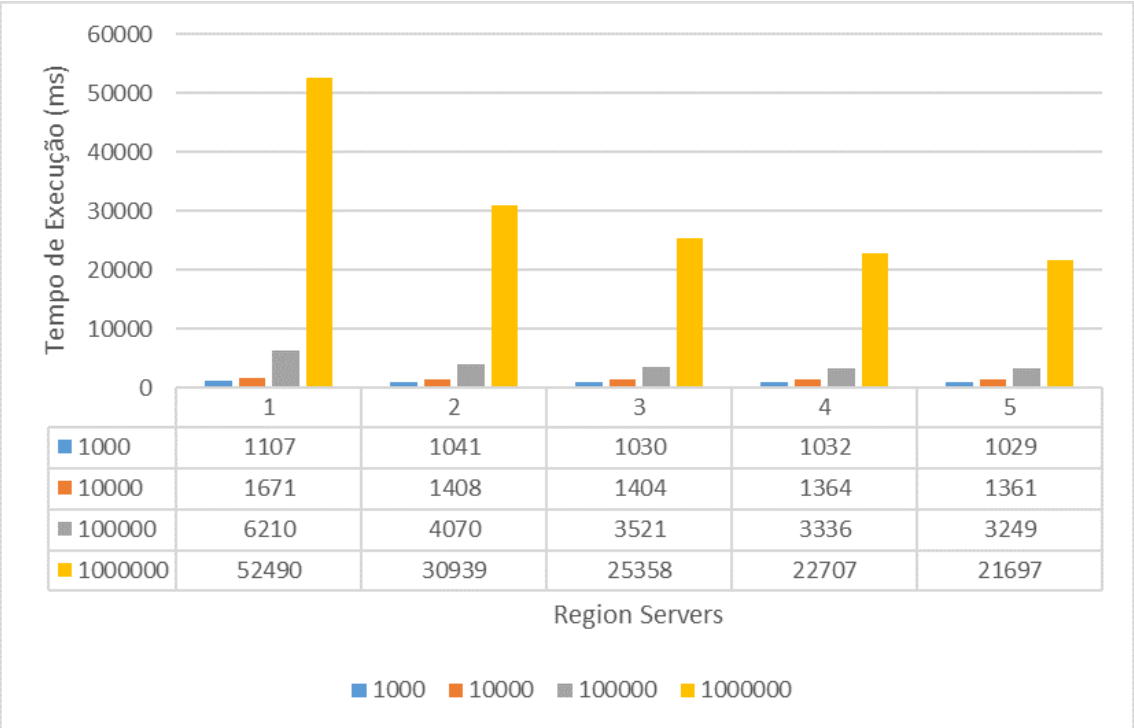
	1	2	3	4	5
1000	971	957	954	944	939
10000	1265	1174	1121	1121	1130
100000	3523	2647	2563	2501	2503
1000000	25526	17337	15715	15619	15513

Figura 18. Latência da *workload* 100% *read* variando o tamanho do conjunto de dados e o número de *region servers* do *cluster*. Fonte: o autor



	1	2	3	4	5
1000	905	961	971	970	972
10000	5986	7106	7126	7330	7350
100000	16105	24569	28405	29985	30781
1000000	19082	32450	39452	44057	46134

Figura 19. Desempenho médio da *workload* 100% *read/modify/write* variando o tamanho do conjunto de dados e o número de *region servers* do *cluster*. Fonte: o autor



	1	2	3	4	5
1000	1107	1041	1030	1032	1029
10000	1671	1408	1404	1364	1361
100000	6210	4070	3521	3336	3249
1000000	52490	30939	25358	22707	21697

Figura 20. Latência da *workload* 100% *read/modify/write* variando o tamanho do conjunto de dados e o número de *region servers* do *cluster*. Fonte: o autor

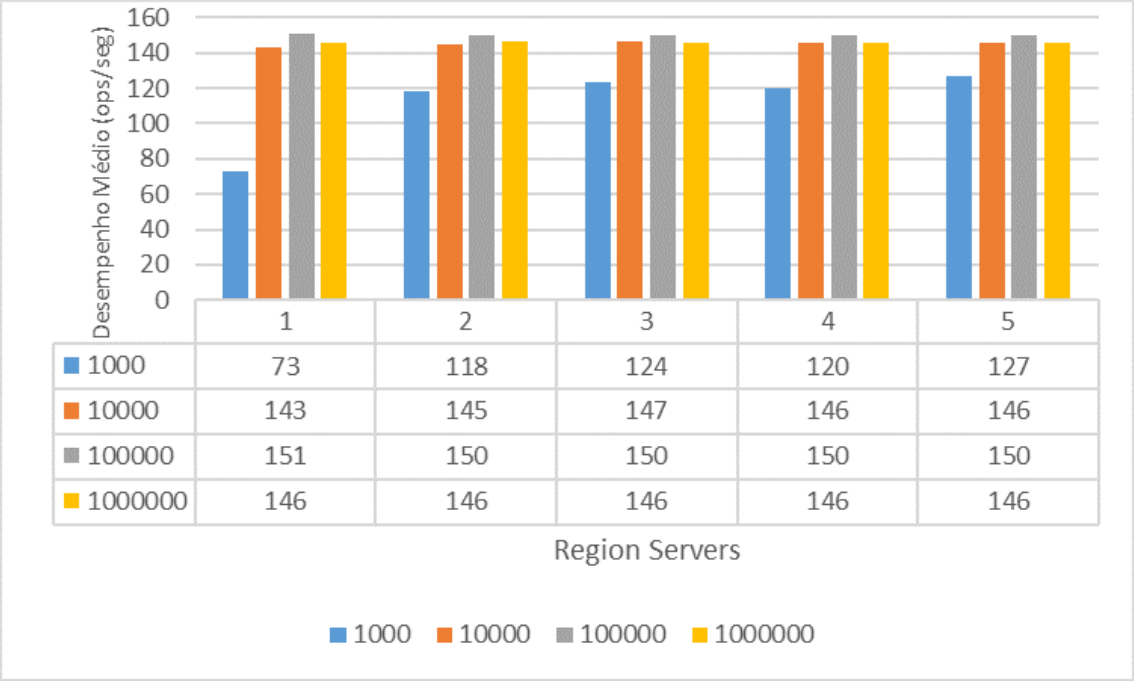


Figura 21. Desempenho médio da *workload* 100% *scan* variando o tamanho do conjunto de dados e o número de *region servers* do *cluster*. Fonte: o autor

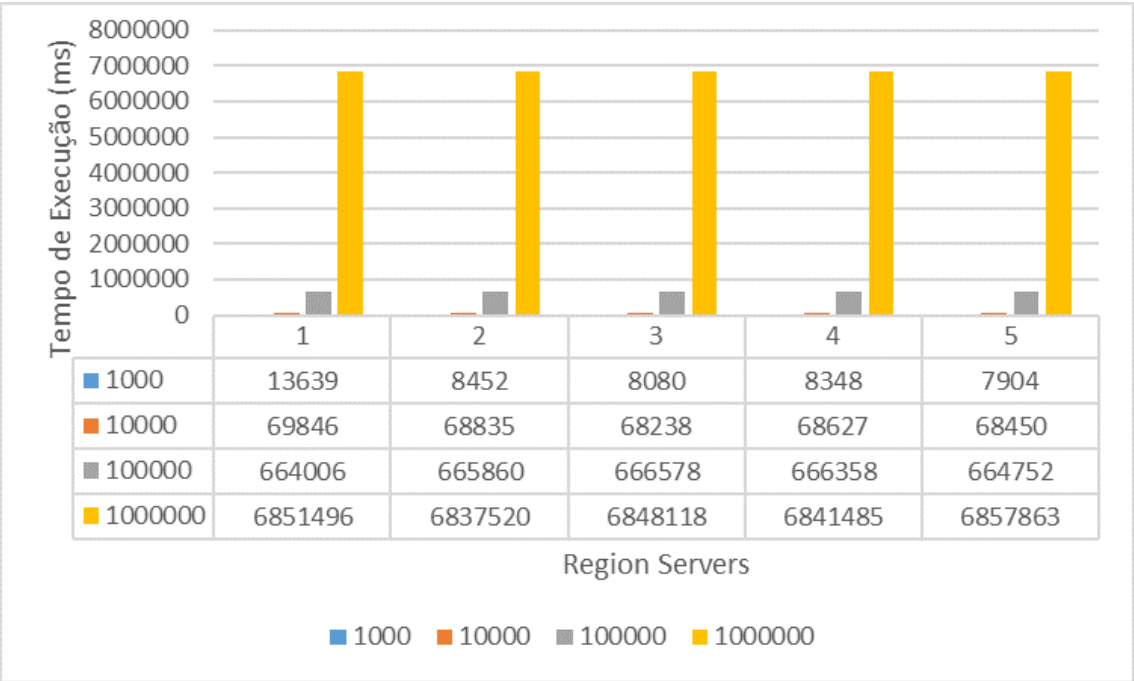


Figura 22. Latência da *workload* 100% *scan* variando o tamanho do conjunto de dados e o número de *region servers* do *cluster*. Fonte: o autor

nho médio do *cluster* sendo a variação mais expressiva de todos os conjuntos menor que 1,4%. Portanto também não houve alteração significativa na latência em nenhum testes executados.

Para os testes cujo o conjunto de dados foi igual a 1.000 registros, o desempenho médio dos testes executados com 5 *region servers* foi maior que do as execuções com 4, 3, 2 e 1 *region servers* em aproximadamente 5,5%, 2,4%, 7% e 42,5% respectivamente. A latência das execuções com 5 *region servers* foi menor que do as execuções com 4, 3, 2 e 1 *region servers* em aproximadamente 5,3%, 2,2%, 6,5% e 42% respectivamente.

6. Conclusão

Este trabalho analisou a escalabilidade horizontal de um *cluster* Hbase, submetendo o banco de dados a *benchmarking* com operações *write*, *read*, *read/modify/write* e *scan*, apoiado pela ferramenta YSCB. Os testes variaram o número de nós escravos e o tamanho do conjunto de dados e foram medidos o desempenho médio (operações por segundo) e a latência (tempo de execução em milissegundos).

Os resultados obtidos mostram que o escalonamento horizontal é mais evidente, considerando as *workloads* 100% *wirte*, *read* e *read/modify/write*, para conjuntos de dados maiores, que neste trabalho foram 100.000 e 1.000.000 de registros de 1KB de tamanho cada. E também, o aumento do desempenho médio do *cluster* é mais significativo aumentando os *region servers* de 1 para 2 e de 2 para 3. A partir de 3 *region servers* o aumento se torna menos expressivo com uma variação média menor que 8% em todos os casos.

Desta forma, a melhora no desempenho do *cluster* Hbase, considerando o desempenho médio e a latência das operações, depende do tamanho do conjunto de dados e que, quanto maior o tamanho desse conjunto, mais evidente se torna essa melhora.Por exemplo, o aumento do desempenho médio nos testes da *workload* 100% *write* – 1.000.000 de registros, aumentando de 1 para 5 *region servers* foi de aproximadamente 47%, enquanto para os conjuntos de tamanho 100.000, 10.000 e 1.000 registros, esse aumento foi de aproximadamente 43%, 11% e 5% respectivamente.

A melhora também varia de acordo com as operações realizadas no banco de dados, por exemplo, o melhor aproveitamento do *cluster* levando em conta o desempenho médio nas execuções com 5 *region servers* foi alcançado pelos testes da *workload* 100% *read*, sendo maior que os testes das *workloads* 100% *write* e 100% *read/modify/write* em aproximadamente 22% e 28% respectivamente.

Os testes da *workload* 100% *scan* mostraram que não há melhora no desempenho do *cluster* para busca de registros independente do tamanho do conjunto de dados devido a metodologia de busca linear implementada pela ferramenta Hbase, que não ganha eficiência a medida que novos nós são adicionados ao sistema. Exceto nos testes em que o conjunto de dados foi igual a 1.000 registros, todos testes apresentaram os mesmos resultados quanto a desempenho médio e latência. Os testes em que o conjunto de dados foi igual a 1.000 teve um aumento no desempenho médio de 42%. Isso ocorreu pois o conjunto de dados por ser pequeno, foi menos fragmentado entre os servidores escravos que os outros conjuntos.

Trabalhos futuros a este podem explorar o aumento do fator de replicação, que neste trabalho foi zero e, analisar o impacto que as operações adicionais de replicação poderão causar à eficiência do *cluster*, também considerando a adição de nós ao sistema.

Referências

Apache Hadoop (2016). Apache hadoop. `http://hadoop.apache.org`. Accessed: 2016-08-25.

Brito, R. W. (2010). Bancos de dados nosql x sgbds relacionais: Análise comparativa. *Infobrasil*.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). Benchmarking cloud serving systems with ycsb. In *IEEE International System-on-Chip Conference (SoCC)*.

Cunha, J. P. (2015). *Column-based databases: estudo exploratório no âmbito das bases de dados NoSQL*. PhD thesis.

Elmasri, R. and Navathe, S. B. (2015). *Fundamentals of Database Systems*. Pearson, 6th edition.

Goldman, A., Kon, F., Junior, F. P., Polato, I., and de Fátima Pereira, R. (2012). Apache hadoop: conceitos teóricos e práticos, evolução e novas possibilidades. *XXXI Jornadas de atualizações em informática,*.

Hadoop Hbase (2016). Apache hbase reference guide. `http://hbase.apache.org/book.html`. Acessado em: 2016-08-25.

Hadoop HDFS (2016). Hdfs architecture guide. `https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html`. Acessado em: 2016-08-25.

Han, J., Haihong, E., Le, G., and Du, J. (2011). Survey on nosql database. *2011 6th International Conference on Pervasive Computing and Applications*, pages 363–366.

Hwang, K., Shi, Y., and Bai, X. (2014). Scale-out vs. scale-up techniques for cloud performance and productivity. *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pages 763–768.

Index, C. V. N. (2017). The zettabyte era-trends and analysis. *Cisco white paper*.

Jogi, V. D. and Sinha, A. (2016). Performance evaluation of mysql, cassandra and hbase for heavy write operation. *2016 3rd International Conference on Recent Advances in Information Technology (RAIT)*, pages 586–590.

McDonald, C. (2015). An in-depth look at the hbase architecture. `https://www.mapr.com/blog/in-depth-look-hbase-architecture`. Acessado em: 2016-08-27.

pc-freak.net (2017). What is vertical scaling and horizontal scaling – vertical and horizontal hardware / services scaling. `https://goo.gl/rMxi99`. Acessado em: 2017-07-04.

Swaminathan, S. N. and Elmasri, R. (2016). Quantitative analysis of scalable nosql databases. *2016 IEEE International Congress on Big Data (BigData Congress)*, pages 323–326.

Waage, T. and Wiese, L. (2014). Benchmarking encrypted data storage in hbase and cassandra with ycsb. *Foundations and Pratices of Security (FPS)*, 1:311–325.