Análise de Escalabilidade Horizontal em um *cluster* Hbase

Omitido [1]

[1]Omitido

Abstract. *Horizontal Scalability allows resources to be distributed by the addition of nodes, being made more flexible by the NoSQL Database. This work evaluates the horizontal scaling potential of a Hbase Database cluster, through a benchmarking in several scenarios, an ideal scenario is obtained, in order to verify the efficiency and availability of applications regardless of storage demand, operations and addition of we. The results indicate that scalability is more evident, except in scan operations, in sets of 100,000 and 1,000,000 records, in increasing the number of nodes from 1 to 2 and from 2 to 3. In the search operations, there is no improvement performance from the insertion of nodes.*

Resumo. *O Escalonamento Horizontal permite que recursos sejam distribuídos pela adição de nós, sendo flexibilizado pelos Banco de Dados NoSQL. Este trabalho avalia o potencial de escalonamento horizontal de um cluster do Banco de Dados Hbase, através de um benchmarking em diversos cenários é obtido um cenário ideal, de modo a verificar a eficiência e disponibilidade de aplicações independentemente da demanda de armazenamento, operações e adição de nós. Os resultados indicam que a escalabilidade é mais evidente, exceto em operações scan, em conjuntos de 100.000 e 1.000.000 de registros, no aumento do número de nós de 1 para 2 e de 2 para 3. Nas operações de busca, não há melhora de desempenho a partir da inserção de nós.*

1. Introdução

Um dos grandes desafios computacionais consiste no armazenamento, recuperação e disponibilidade de dados de modo eficiente. Durante muitos anos, a solução majoritária foi a adoção dos Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDR), que garantem o conjunto de propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade). Contudo, os SGBDR's não satisfazem as necessidades de sistemas no âmbito de *Big Data*.

O *Big Data* se refere ao grande volume de dados gerados em diversos domínios de aplicação [Han et al. 2011]. A tecnologia relacional possui fragilidade no tratamento de dados semi-estruturados e não-estruturados, além de dificuldade de distribuição devido a atender as propriedades ACID [González-Aparicio et al. 2016]. A complexidade lógica existente na modelagem relacional somada ao alto volume de dados mostrou-se um problema, visto que pode propiciar *deadlocks*, além de problemas de concorrência, lentidão na leitura e escrita dos dados [Han et al. 2011]. Como alternativa surgiram os Bancos de Dados *NoSQL*, que atuam de modo mais eficiente com o armazenamento e manipulação de grande volume de dados, possibilitando escalar operações por diversos servidores, além de prover maior flexibilidade [Ramesh et al. 2016].

Neste contexto, o objetivo do trabalho consiste analisar o potencial de escalabilidade horizontal de um *cluster* executando o Banco de Dados *NoSQL* HBase. Foram conduzidos testes em diferentes cenários de implementação do *cluster*, com o intuito de obter um cenário ideal, em que aplicações e serviços se mantenham eficientes e disponíveis independentemente da demanda de armazenamento e consultas, apenas pela adição de nós ao ambiente distribuído. Os testes foram conduzidos por meio de um *benchmarking* com operações *read*, *write*, *scan* e *read-modify-write* utilizando o *framework Yahoo! Cloud Serving Benchmark* (YCSB) e variação do ambiente.

O HBase está integrado a plataforma Hadoop, também composta por sistema de arquivos distribuídos *Hadoop Distributed File System* (HDFS) e o modelo de suporte a programação paralela MapReduce [HBase 2019]. O Hbase permite o processamento distribuído por meio de *clusters*, atuando sobre o HDFS de modo a prover recursos semelhantes ao BigTable [Chang et al. 2008] e alta tolerância a falhas ao armazenar grandes quantidades de dados esparsos [HBase 2019].

Este trabalho está organizando do seguinte modo: Na Seção 2 são apresentados conceitos de escalabilidade em banco de dados e as ferramentas utilizadas; A Seção 3 expõe os trabalhos relacionados; Na Seção 4 é apresentada a configuração do experimento descrevendo os cenários de testes utilizados; A Seção 5 exibe a análise dos resultados obtidos; Por fim, a Seção 6 apresenta as considerações finais.

2. Fundamentação Teórica

2.1. Escalabilidade em Banco de Dados

Escalabilidade é a capacidade de expandir os recursos (armazenamento e processamento) de um sistema [Elmasri and Navathe 2010]. Um Banco de Dados escalável possui a capacidade de manipular maior quantidade de dados e garantir a disponibilidade do sistema. Existem duas abordagens para escalabilidade: *Vertical* e *Horizontal*.

A *Escalabilidade Vertical* consiste em adicionar mais recursos ao servidor, a qual prove menor consumo de energia, implementação facilitada e menores problemas de arrefecimento. Contudo, possui custo extremamente superior a escalabilidade horizontal, além de possibilitar a interrupção do serviço por falha devido a um único servidor [Hwang et al. 2014]. Em contraponto, na *Escalabilidade Horizontal*, os recursos são distribuídos em diferentes servidores, em geral máquinas simples, com o propósito de redução de custos. Existe garantia de recuperação em caso de falhas, devido a presença de redundância de dados e processos em diversos nós. Essa arquitetura é presente em diversos Banco de Dados *NoSQL* [Hwang et al. 2014].

2.2. Apache Hadoop

O Apache Hadoop consiste em um (*framework*) para processamento distribuído de grande volume de dados em *clusters*. Utiliza o modelo *MapReduce*, projetado para escalonamento horizontal, oferecendo alta disponibilidade e recuperação de falhas [HBase 2019]. Um *cluster* Hadoop opera sob a arquitetura mestre/escravo – Figura 1. Existem cinco processos, o *NameNode* e o *JobTraker* são processos executados pelo nó-mestre, enquanto *SecondaryNameNode* pelo nó-mestre alternativo, em caso de falha. O *DataNode* e *TaskTraker* atuam como processos escravos de múltiplas instâncias.

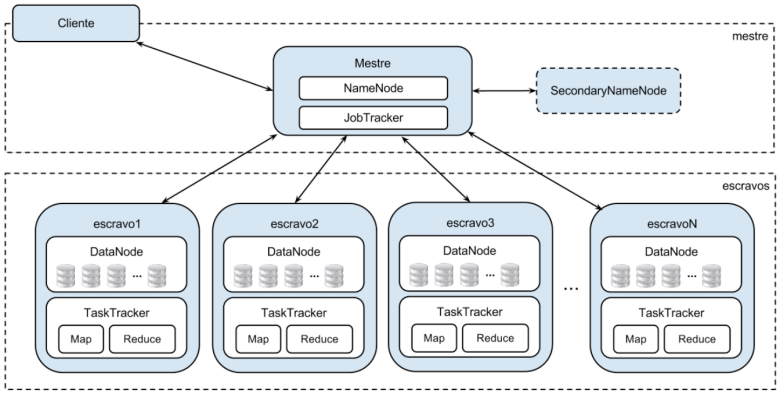


Figura 1. Estrutura dos processos do Hadoop [Goldman et al. 2012].

2.3. HBase

O Hbase é um banco de dados *NoSQL*, distribuído, tolerante a falhas de código aberto e altamente escalável, com foco em aplicações que necessitam de leitura e escrita de acesso aleatório com tempo constante [HBase 2019]. O projeto foi inspirado no *BigTable* da Google, ambos utilizam o modelo de dados orientado a colunas. Esse não possui nenhuma linguagem de consulta estruturada, que fornece uma API Java que possibilita realizar operações básicas como *put*, *get*, *update* e *delete*, e também o uso da função *scan*, para selecionar quais as colunas retornadas ou o número de versões de cada célula, porém consultas mais complexas utilizam *jobs* do MapReduce.

Os dados do HBase são armazenados como arquivos do HDFS, um sistema de arquivos distribuídos tolerante a falhas, voltado para *hardwares* de baixo custo e aplicações com grande volume de dados. Dessa forma, os servidores escravos (*region servers*) são dispostos nos nós que executam os processos *DataNode* do HDFS provendo localidade nos dados que transitam de um para o outro.

2.4. Yahoo! Cloud Serving Benchmark

O YCSB (*Yahoo!Cloud Serving Benchmark*) é um *framework* de *benchmarking* para bancos de dados distribuídos, que permite a adição de novas implementações [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, enquanto a escalabilidade é medido o impacto na performance quando o número de servidores do sistema cresce [Cooper et al. 2010].

Esse possui um conjunto de *workloads* denominado *Core Package* e uma aplicação chamada YCSB *Client*. As *workloads* consistem de combinações de operações de leitura e escrita, na execução de uma *workload*, o YCSB *Client* cria um conjunto de dados e submete as requisições ao banco de dados [Cooper et al. 2010].

3. Trabalhos Relacionados

[Jogi and Sinha 2016] realizam a comparação MySQL com Cassandra e Hbase em operações *heavy* write, utilizando uma aplicação *web* REST (*Representacional State Transfer*) para recebimento dos dados e armazenamento no Banco de Dados. Concluise que o Cassandra apresentou melhor velocidade de escrita, enquanto o Hbase foi duas

vezes mais rápido que o MySQL, isso ocorre devido a incorporação de características do *BigTable* do Google e do DynamoDB pelo Cassandra. [Swaminathan and Elmasri 2016] analisaram a escalabilidade nos Bancos de Dados: Hbase, Cassandra e MongoDB, utilizando o *framework* YCSB com diferentes cargas de trabalho e conjunto de dados, no intuito de evidenciar as vantagens e desvantagens de cada em cenários específicos.

Segundo [Waage and Wiese 2014], a confiabilidade para o armazenamento "em nuvem" é um dos pontos chaves para adoção de tecnologias não-relacionais. Foi proposto que os dados sejam criptografados, assim ele avaliou o impacto da criptografia, foi realizado um estudo com os Bancos de Dados Cassandra e Hbase. Para tal foi utilizado o *framework* YCSB em que *workloads* foram aplicadas a dados não-encriptados e encriptados usando o algoritmo *Advanced Encryption Standard* (AES) com chaves de diferentes comprimentos. Foi relatada uma redução no desempenho médio do *cluster*, o qual é independente do tamanho da chave de encriptação.

Na apresentação do YCSB, [Cooper et al. 2010] aplicaram *benchmarking* para o Cassandra, Hbase, Yahoo!'s PNUTS e o Sharded MySQL, como exemplo de uso do *framework*. Observou-se que o Cassandra e Hbase apresentam maior latência para operações *read* e menor latência para *update* e *write* em relação ao PNUTS e MySQL, enquanto o PNUTS e Cassandra possuem melhor escalabilidade em detrimento ao HBase, quando o número de servidores aumenta proporcionalmente a carga de trabalho.

Diferentemente da literatura, este trabalho avalia o potencial do escalonamento horizontal de um *cluster* Hbase, através de um *benchmarking*, de modo a criar um cenário que aplicações e serviços se mantenham eficientes e disponíveis independentemente da demanda de armazenamento e consultas, em termos de eficiência e disponibilidade.

4. Configuração do Ambiente de Experimentação

Os testes foram executados em um *cluster* composto por 7 computadores com Linux CentOS 6.6, sendo um deles o nó mestre denominado hpcdmc e os seis caracterizados como nós escravos denominados de n_1 a n_6. **Configuração do nó mestre (hpcdmc):** 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 (n_1 a n_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 cache.

O experimento utilizou os softwares: Hadoop 2.7.3; HBase 1.2.4; ZooKeeper 3.4.10; YCSB 0.12.0, em que foi priorizada estabilidade na escolha das versões.

Para os testes iniciais foram organizados cinco cenários. Com base nesses foi analisado a melhor alternativa para a implementação do *cluster* e avaliar sua escalabilidade, ou seja, como serão dispostos os processos cliente, mestre e escravo, e assim foi idealizado e avaliado o sexto cenário.

Nos cinco primeiros cenários foram executados os testes em um ambiente pseudo distribuído, com duas *workloads*: 100% *write* e 100% *read* – 100.000 registros, tendo cada registro 1KB e distribuição uniforme. Cada teste sobre uma *workload* foi executado 3 vezes para uma dada quantidade de *threads* do YCSB *Client*, foi obtido a média da latência (tempo de execução em milissegundos) e do desempenho médio (operações/segundo) para obter o resultado final do teste.

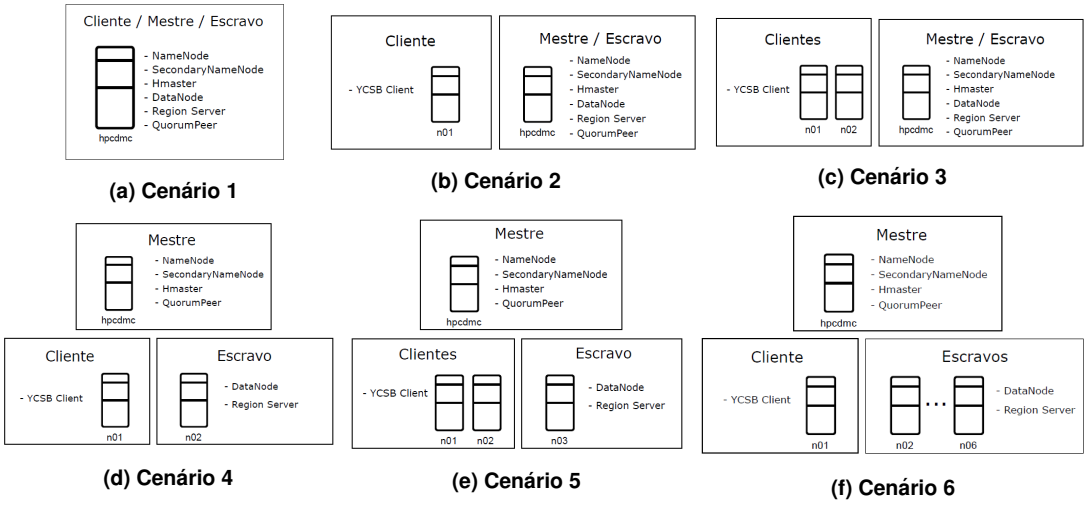


Figura 2. Configuração dos seis cenários de testes.

No sexto cenário, cuja configuração foi obtida dos testes anteriores, é analisada a escalabilidade do HBase em um *cluster* totalmente distribuído. Cada teste foi executado 3 vezes, cujo resultado a média das saídas. A seguir é descrita a organização dos cenários:

Cenário 1: o *cluster* é configurado 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 – Figura 2a.

Cenário 2: 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. Assim, 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 n_1 – Figura 2b.

Cenário 3: dada a limitação de *threads* do YCSB *Client*, que podem ser executadas no n_1, pelo número de núcleos do processador, no cenário 3 a execução da *workload* foi dividida entre nós n_1 e n_2. 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* e 64 *threads*, cada nó gerou 50.000 requisições *read* com 32 *threads* ativas no YCSB. A saída de cada foi a soma do desempenho médio (operações/segundo) e a maior latência (tempo de execução em milissegundos). O hpcdmc permaneceu pseudo distribuído – Figura 2c.

Cenário 4: consiste na aproximação de uma situação mais adequada quanto a configuração do *cluster*, em que os processos cliente, mestre e escravo estão completamente distribuídos. O n_1 é responsável pelo YCSB *Client*, o hpcdmc pelos processos mestre do Hadoop, HBase e dos processos do ZooKeeper, enquanto o n_2 executou os processos escravos do Hadoop e HBase – Figura 2d.

Cenário 5: analisou a necessidade de mais de um servidor gerando requisições na saturação do *cluster*. Assim, o n_1 e n_2 dividiram a execução das *workloads*, o hpcdmc foi responsável pelos processos mestre do Hadoop, HBase e os processos do ZooKeeper

e o n_3 executou os processos escravos do Hadoop e HBase – Figura 2e.

Cenário 6: Os resultados dos testes anteriores apoiaram as decisões quanto a estrutura do cenário 6 – Figura 2f. Foram executadas as *workloads*: *write*, *read* e *scan*, variando entre 1.000, 10.000, 100.000 e 1.000.000 registros de tamanho 1KB e com distribuição uniforme, variando o número de escravos de 1 a 5 e o número de *threads* do YCSB *Client* fixo.

5. Análise dos Resultados

Essa Seção apresenta a análise dos resultados com a execução dos testes. Inicialmente são descritas as análises dos resultados obtidos nos cinco primeiros cenários, em seguida apresentamos as análises dos resultados dos testes aplicados em um *cluster* totalmente distribuído, cenário 6.

5.1. Cenários de 1 a 5

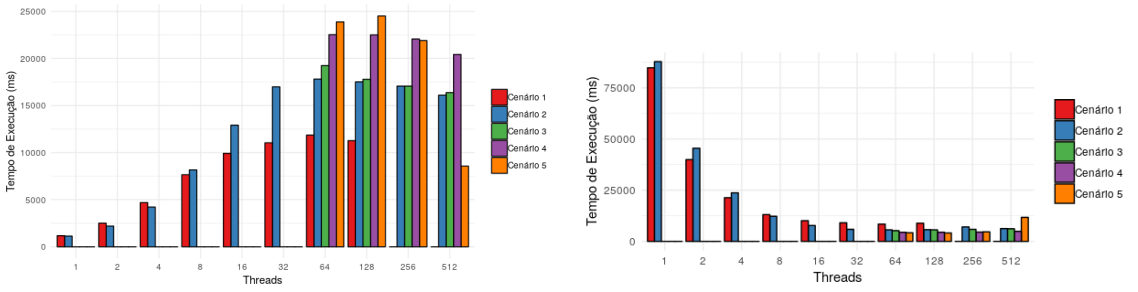
Analisando os resultados dos teste aplicados aos cenários 1 e 2 foi identificado um crescimento no desempenho médio do *cluster* nas execuções com o uso de até 64 *threads* – Figuras 3a e 4a. No cenário 1, ao utilizar entre uma e 64 *threads* ocorreu um aumento de aproximadamente 90% no desempenho médio, enquanto no cenário 2 ocorreu, para a mesma quantidade de *threads*, um aumento de aproximadamente 93%.

Na instanciação de mais de 64 *threads*, ao analisar os resultados obtidos com os testes aplicados ao cenário 1, observa-se queda no desempenho médio de aproximadamente 5% com o uso de 128 *threads*. Considerando o cenário 2, também há 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*.

Observa-se queda na latência nos testes dos cenários 1 e 2 até as execuções com 64 *threads* – Figuras 3b e 4b, 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%.

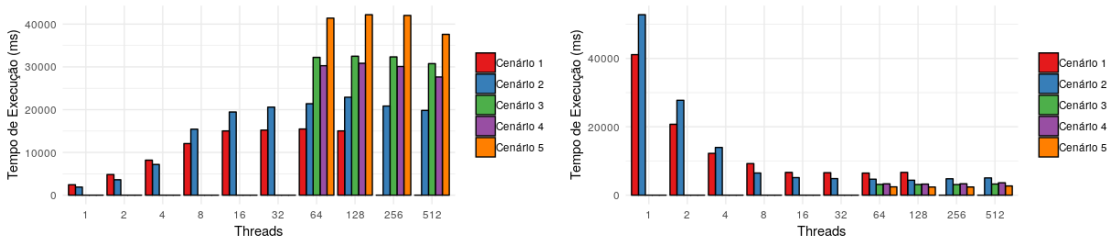
Constata-se que o desempenho máximo do *cluster* nos dois primeiros cenários foi obtido nas execuções do YCSB *Client* com o uso de 64 *threads*, a partir do cenário 3 foram executados apenas os testes com o uso de 64 a 512 *threads*, de modo a observar se esses resultados foram ótimos locais ou globais. 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* já 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 dos testes com *workload* 100% *write* – 100.000 registros, mostrou que o melhor desempenho médio foi alcançado pelo cenário 5 para as execuções com o uso de 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. Nas execuções com o uso de 256 e 512 *threads* o melhor desempenho médio foi obtido pelo cenário 4 sendo maior que os cenários 5, 3 e 2 em 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.



(a) Desempenho médio **(b) Latência**

Figura 3. *Workload* 100% *write* 100.000 registros nos cenários de 1 a 5.



(a) Desempenho médio **(b) Latência**

Figura 4. *Workload* 100% *read* 100.000 registros nos cenários de 1 a 5.

O cenário 5 apresentou esse desempenho pois o *cluster* foi completamente distribuído, então os processos mestres e escravo do Hbase e Hadoop, 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 n_1 e n_2 apresentou uma pequena melhora no cenário 5 no desempenho do *cluster* executando o YCSB com 64 e 128 *threads*, já nas execuções com o uso de 256 e 512 *threads* tal divisão sobrecarregou demais o n_3 fazendo com que o desempenho médio diminuísse e a latência aumentasse, tornando os resultados do cenário 4 melhores.

A partir dos testes com a *workload* 100% *write* – 100.000 registros, pode-se concluir que o cenário 5, para execuções com o uso de 64 e 128 *threads* obteve uma execução mais rápida se comparado com os resultados obtidos com os testes dos cenários 4, 3, 2 e 1 aproximadamente 5,6%, 19,7%, 25,4% e 50,2% com o uso de 64 *threads* e, 7,8%, 27%, 28,1% e 53,7% com 128 *threads* respectivamente. Enquanto que as execuções com o uso de 256 e 512 *threads* o cenário 4 apresentou resultados nos quais a execução foi mais rápida que os resultados obtidos com os testes dos cenários 5, 3 e 2 aproximadamente 2,4%, 22,8% e 35,8% com o uso de 256 *threads* e, 95,8%, 20,3% e 21,5% com o uso de 512 *threads* – Figura 3b.

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

vamente.

Para as operações de leitura na *workload* 100% *read*, a divisão de requisições entre o n_1 e n_2 não sobrecarregou o n_3 para execuções com o uso de 256 e 512 *threads* como nos testes com a *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 n_1 e n_2 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 – Figura 4a.

Na *workload* 100% *read* – 100.000 registros – Figura 4b, a latência obtida com os 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 o uso de 64 *threads* e, 26,6%, 23,6%, 45,6% e 64,3% nas execuções com o uso de 128 *threads* respectivamente. E também, menor que os cenários 4, 3 e 2 aproximadamente 27,9%, 22,8% e 50% nas execuções com o uso de 256 *threads* e, 25,5%, 17,3% e 46.6% nas execuções com o uso de 512 *threads* respectivamente.

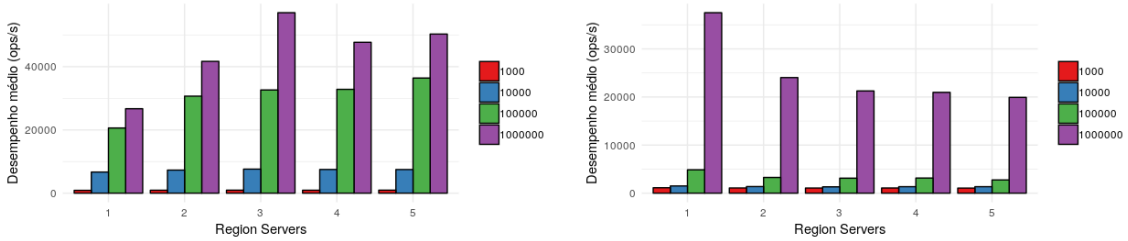
Observou-se também que para todos os cenários as execuções com o uso de 64 e 128 *threads* são as mais altas e a variação referente a um mesmo cenário, considerando essas duas quantidades de *threads* não são expressivas, sendo de aproximadamente 1%, para os testes da *workload* 100% *write* e 100% *read*, para latência e desempenho médio. 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% *write* e, 10,9%, 10,4%, 5,3 e 13,5% para os testes da *workload* 100% *read* respectivamente. Comparando então os 5 cenários em que 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 26,8% respectivamente.

O cenário 4 obteve o segundo maior desempenho médio com a *workload* 100% *write* com menos de 6% comparado ao cenário 5, obtendo o terceiro maior desempenho médio nos testes da *workload* 100% *read* com uma diferença de aproximadamente 6% comparado com os resultados obtidos com os testes aplicados ao 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. Cenário 6

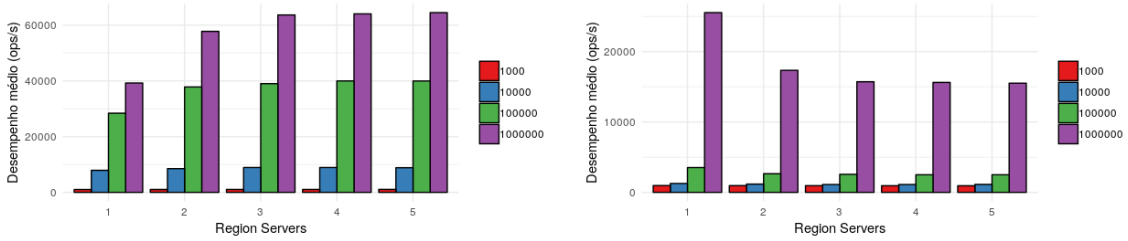
Inicialmente, a respeito do desempenho médio (Figura 5a), a escalabilidade horizontal foi mais evidenciada com 1.000.000 de registros, sendo o desempenho do *cluster* com 5 *region servers* maior do que os testes executados com 4, 3, 2, e 1 *region servers* em aproximadamente 5,1%, 6,5%, 17,1% e 47% respectivamente. A análise dos resultados dos testes para 100.000 registros também demonstraram 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) Desempenho médio **(b) Latência**

Figura 5. *Workload* 100% *write* variando o tamanho do conjunto de dados e o número de *region servers* do *cluster* .



(a) Desempenho médio **(b) Latência**

Figura 6. *Workload* 100% *read* variando o tamanho do conjunto de dados e o número de *region servers* do *cluster*.

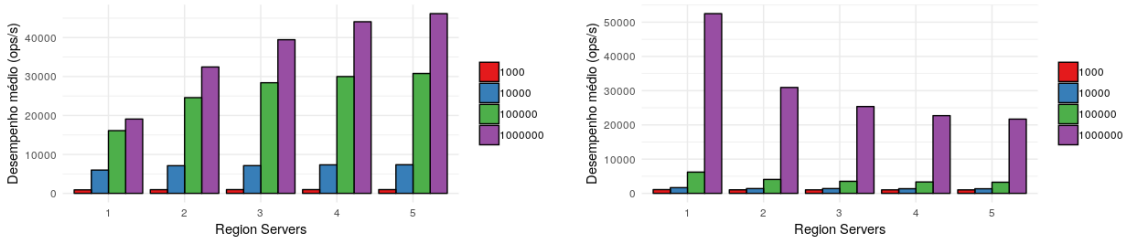
A partir da análise dos resultados dos testes da *workload* 100% *write* – Figura 5b, observa-se que com 1.000.000 registros a latência foi menor que os testes com 4, 3, 2 e 1 *region server* em aproximadamente 5%, 6,3%, 17,1% e 47% respectivamente. Com base nos testes com 100.000 registros, a escalabilidade foi menos evidenciada, as latências tiveram poucas alterações comparados com 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.

Nos testes com conjunto de dados de 1.000 e 10.000 registros não houve alterações significativas no desempenho médio com a adição de nós, sendo a variação mais expressiva para 1.000 registros de aproximadamente 4% aumentando de 1 para 2 *region servers*, aproximadamente 8% para 10.000 aumentando de 1 para 2 *region servers*, considerando o desempenho médio – Figura 6a e a latência – Figura 6b.

Para a *workload* 100% *read*, assim como nos resultados acima, nota-se que a escalabilidade horizontal foi mais evidenciada quando o conjunto com 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% – Figura 6a. Pelos 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%. 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* – Figura 6b, também não apresentou alterações significativas para os testes com 1.000.000 e 100.000 registros quando variados os *region servers* entre 3 e 5, sendo essa variação menor que 1,3% para ambos. Desta forma, os testes executados com 5 *region servers* apresentaram menor latência 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. Para 1.000 e 10.000 registros não houve alterações significativas no desempenho médio, sendo a variação de 1 para 5 *region servers* menor que 3,3% para 1.000 registros e, de menos de 10,7% para 10.000 registros.

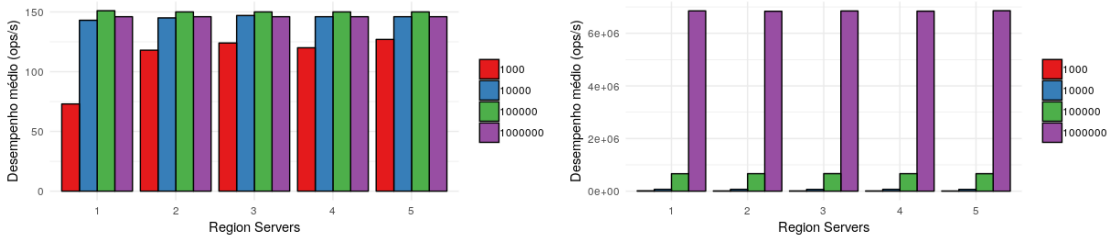


(a) Desempenho médio **(b) Latência**

Figura 7. *Workload* 100% *read/modify/write* variando o tamanho do conjunto de dados e o número de *region servers* do *cluster*.

Quanto ao desempenho médio dos testes da *workload* 100% *read*/modify/write – Figura 7a, observou-se novamente que, a escalabilidade horizontal é mais evidenciada nos cenários com 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 com 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 com 1.000 registros não apresentaram resultados com alterações signifi-



(a) Desempenho médio **(b) Latência**

Figura 8. *Workload* 100% scan variando o tamanho do conjunto de dados e o número de *region servers* do *cluster*.

cativas no desempenho médio, a variação mais expressiva é de menos de 6% aumentando de 1 para 2 *region servers*. No conjunto de 10.000 registros, a variação mais expressiva foi de 18,5% quando aumentado de 1 para 2 *region servers*. As demais adições variaram menos que 4%.

Considerando a *workload* 100% *read/modify/write* o comportamento da latência – Figura 7b, acompanha o desempenho médio de modo inversamente proporcional, assim, 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, com 100.000 registros e 5 *region servers* houve 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 com a *workload* 100% *read/modify/write* de 1.000 e 10.000 registros não tiveram alterações significativas na latência, seguindo a mesma porcentagem calculada nos testes de desempenho médio. O desempenho médio – Figura 8a e latência – Figura 8b para os testes da *workload* 100% *scan*, mostra-se que, exceto com 1.000 registros, que não houve alterações significativas no desempenho médio do *cluster* sendo a variação mais expressiva de todos os conjuntos menor que 1,4%.

Nos testes com 1.000 registros, o desempenho médio das execuções 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. Considerações Finais

Os resultados obtidos mostram que o escalonamento horizontal é mais evidente, considerando as *workloads* 100% *write*, *read* e *read/modify/write*, para conjuntos superiores a cem mil registros. Também foi identificado que o desempenho médio do *cluster* é mais significativo no aumento de *region servers* de 1 para 2 e de 2 para 3, sendo menos expressivos a partir de 3 *region servers* (inferior a 8%).

Desta forma, a melhora no desempenho do *cluster* Hbase, considerando o desempenho médio e a latência das operações, é diretamente proporcional ao tamanho do conjunto de dados, de modo mais evidente.

O ganho de desempenho varia de acordo com as operações. Por exemplo, o melhor aproveitamento do *cluster* para 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. Contudo, os testes da *workload* 100% *scan* mostram que não há melhora no desempenho para busca, independentemente do conjunto de dados, devido a implementação da busca linear implementada pela ferramenta Hbase.

Referências

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM.

Elmasri, R. and Navathe, S. (2010). *Fundamentals of database systems*. Addison-Wesley Publishing Company.

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 atualizaçoes em informatica*, pages 88–136.

González-Aparicio, M. T., Younas, M., Tuya, J., and Casado, R. (2016). A new model for testing crud operations in a nosql database. In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pages 79–86.

Han, J., Haihong, E., Le, G., and Du, J. (2011). Survey on nosql database. In *2011 6th international conference on pervasive computing and applications*, pages 363–366. IEEE.

HBase, H. (2019). Apache hbase reference guide. `http://hbase.apache.org/book.html`. Online; accessed 03 May 2019.

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

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

Ramesh, D., Khosla, E., and Bhukya, S. N. (2016). Inclusion of e-commerce workflow with nosql dbms: Mongodb document store. In *2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, pages 1–5.

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

Waage, T. and Wiese, L. (2014). Benchmarking encrypted data storage in hbase and cassandra with ycsb. In *International Symposium on Foundations and Practice of Security*, pages 311–325. Springer.