Análise de Escalabilidade Horizontal em um *cluster* Hbase

Cedryk Augusto dos Santos [1], Bruno Santos de Lima [1], Leandro Ungari Cayres [1], Rogério Eduardo Garcia [1], Celso Olivete Junior [1], Danilo Medeiros Eler [1], Ronaldo Celso Messias Correia [1]

[1]Faculdade de Ciências e Tecnologia – Universidade Estadual Paulista (
Presidente Prudente – SP – Brasil

cedrykaugusto@gmail.com,{bruno.s.lima,leandro.ungari,rogerio.garcia,celso.olivete,dar

Abstract. NoSQL Databases facilitate horizontal scheduling, allowing the distribution of resources with the addition of nodes. This work evaluates the potential of horizontal scheduling of an HBase database through a benchmarking in several scenarios, setting an ideal scenario, in order to verify the efficiency and availability of applications regardless of the demand for storage, operations and addition nodes. The results show that scalability is more evident, except in scan operations, in sets of 100,000 and 1,000,000 records of 1KB each, increasing the number of nodes from 1 to 2 and from 2 to 3. In search operations , there is no improvement in performance from the insertion of nodes.

Resumo. *Os Bancos de Dados* NoSQL *facilitam escalonamento horizontal, permitindo a distribuição de recursos com a adição de nós. Este trabalho avalia o potencial do escalonamento horizontal de um* cluster *do Banco de Dados Hbase, através de um* benchmarking *em diversos cenários, configurando um cenário ideal, de modo a verificar a eficiência e disponibilidade de aplicações independentemente da demanda de armazenamento, operações e adição nós. Os resultados demonstram que a escalabilidade é mais evidente, exceto em operações* scan*, em conjuntos de 100.000 e 1.000.000 de registros de 1KB cada, no aumento do número de nós de 1 para 2 e, de 2 para 3. Nas operações de busca, não há melhora no 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 para a maioria dos problemas dessa área foram os 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* [Brito 2010].

O conceito de *Big Data* refere-se ao grande volume de dados gerados em diversos domínios de aplicação [Han et al. 2011, Index 2013]. A partir desse conceito, identificou-se que a tecnologia relacional possui fragilidade no tratamento de dados classificados como semiestruturados e não estruturados, além de dificuldade de implementação distribuída, de modo 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, Brito 2010]. Como alternativa surgiram os Bancos de Dados *NoSQL*, os quais atuam de modo mais eficiente com o armazenamento e manipulação de grande volume de dados, possibilitando escalar horizontalmente operações por diversos servidores, além de prover maior flexibilidade [Mohamed et al. 2014, Ramesh et al. 2016].

Neste contexto, o objetivo deste trabalho consiste em verificar o potencial desse novo paradigma de armazenamento e recuperação de dados utilizando computação distribuída, de modo a criar um cenário, em que aplicações e serviços se mantenham eficientes e disponíveis independentemente do tamanho da demanda de armazenamento e consultas, somente através da adição de novos nós ao ambiente distribuído. Desse modo, através de um experimento foi avaliada a escalabilidade horizontal do Banco de Dados Hbase, foi executado um *benchmarking* com operações *read*, *write*, *scan* e *read-modify-write* utilizando o *framework Yahoo! Cloud Serving Benchmark* (YCSB) a partir da inserção de novos nós ao sistema e crescimento do conjunto de dados.

O ambiente de desenvolvimento Hadoop foi criado pela *Apache Foundation* e reúne ferramentas de manipulação *Big Data*, envolvendo o Banco de Dados Hbase, o sistema de arquivos distribuídos *Hadoop Distributed File System* (HDFS) e o modelo de suporte a programação paralela em grandes coleções de dados chamado MapReduce [HBase 2019]. O Banco de Dados Hbase permite o processamento distribuído por meio de *clusters*, atuando sobre o HDFS de modo a agregar à ferramenta capacidades semelhantes ao BigTable [Chang et al. 2008] e alta tolerância a falhas ao armazenar grandes quantidades de dados esparsos [HBase 2019].

Além dessa introdução, 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; A Seção 4 apresenta a configuração do experimento abordando os cenários de testes utilizados; A Seção 5 exibe a análise dos resultados obtidos; Por fim, a Seção 6 apresenta as considerações finais e indica trabalhos futuros.

2. Fundamentação Teórica

2.1. Escalabilidade em Banco de Dados

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

A *Escalabilidade Vertical*, tradicionalmente usado pelos Bancos de Dados Relacionais, consiste em adicionar mais recursos (CPU / Memória RAM / Disco Rígido) ao servidor. Caracterizada pelo menor consumo de energia, e menores problemas de arrefecimento, sendo de implementação facilitada. Suas desvantagens são: custo extremamente superior a escalabilidade horizontal, possibilidade de interrupção do serviço por falha devido a um único servidor(ponto crítico do sistema), além de quando esgotadas as possibilidades de *upgrade* da máquina atual é necessário transferir o sistema para outro servidor [Hwang et al. 2014].

Em contraponto, na *Escalabilidade Horizontal*, os recursos não são centralizados em um servidor, mas sim distribuídos em diversos servidores. Os servidores incluídos consistem em máquinas simples, com o propósito de redução de custos. Existe garantia de recuperação em caso de falhas, a paralisação de uma máquina não causa a interrupção do sistema, devido a presença de redundância de dados e processos em diversos nós. Essa arquitetura é muito semelhante em diversos Banco de Dados *NoSQL* [Hwang et al. 2014], assim como no HBase, utilizado neste trabalho.

2.2. Apache Hadoop

A biblioteca de softwares Apache Hadoop consiste em um conjunto de aplicações (*framework*) para processamento distribuído de grande volume de dados em *clusters*. Esse utiliza o modelo de programação simples *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. Em uma aplicação típica existem cinco processos envolvidos. O *NameNode*, *SecondaryNameNode* e o *JobTraker* são processos únicos para toda a aplicação, em que o *NameNode* e *JobTracker* são executados pelo nó-mestre, enquanto *SecondaryNameNode* pelo nó-mestre alternativo, em caso de falha. O *DataNode* e *TaskTraker* atuam como processos escravos de multiplas instâncias. Por fim, os processos *NameNode*, *SecondaryNameNode* e *DataNode* fazem parte da execução do HDFS, enquanto *JobTraker* e *TaskTraker* parte da execução MapReduce [Goldman et al. 2012].

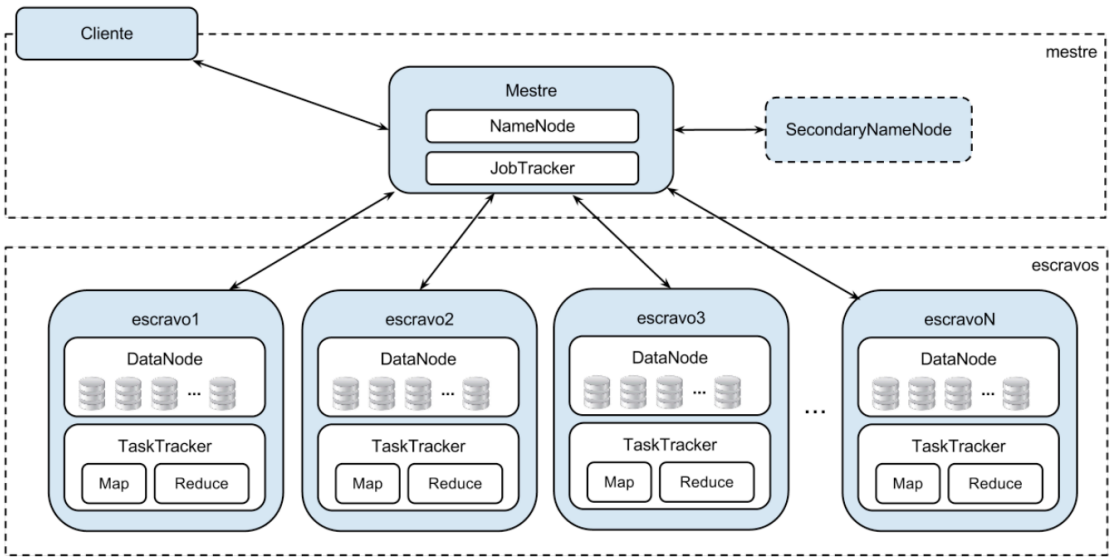


Figura 1. Representação da estrutura dos processos do Hadoop [Goldman et al. 2012].

2.3. HDFS

O HDFS consiste em um sistema de arquivos distribuídos projetado para ser tolerante a falhas, voltado para *hardwares* de baixo custo e aplicações com grande volume de dados. Assim como um sistema de arquivos, o HDFS fornece operações de armazenamento, organização, nomeação, atribuição de permissões de acesso, compartilhamento

e recuperação de forma transparente ao usuário, porém, o diferencial está no armazenamento dos arquivos, os quais compartilhados por máquinas remotas ligadas em rede.

O processo *NameNode* tem responsabilidade pela abertura, fechamento ou renomeação de arquivos e diretórios; realizar o controle de acesso dos arquivos, armazenar metadados, coordenar a fragmentação dos arquivos em blocos, assim como a distribuição dos mesmos dentro dos processos *DataNodes*. Os processos *DataNode* tem a tarefa de armazenar os dados, responder as solicitações de leitura e escrita dos clientes e aplicações, criar, remover ou replicar blocos sob orientação do *NameNode*. Além de armazenar, os *DataNode* precisam se comunicar constantemente com o *NameNode* para manter o mapeamento dos blocos atualizado e livre de falhas [Hadoop 2016].

2.4. HBase

O Hbase é um Banco de Dados *NoSQL*, distribuído, tolerante a falhas, de código aberto e altamente escalável. Tem como foco aplicações que necessitam de leitura e escrita de acesso aleatório com tempo constante em grandes volumes de dados [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, porém fornece uma API Java que possibilita realizar operações básicas como *put*, *get*, *update* e *delete*. 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].

Todos os dados do HBase são armazenados como arquivos do HDFS. 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.

2.5. Yahoo! Cloud Serving Benchmark

O YCSB (*Yahoo! Cloud Serving Benchmark*) é um *framework* de código-aberto para apoio a *benchmarking* de diferentes Bancos de Dados distribuídos, a qual permite a extensão para Banco de Dados não implementados previamente [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, 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 chamada YCSB *Client*. As *workloads* consistem de combinações de operações de leitura e escrita que atuam sobre um conjunto de dados de certo tamanho e distribuição. Na execução de uma *workload*, o YCSB *Client* realiza a interpretação do arquivo descritor, criando o conjunto de dados e submete as requisições ao banco de dados [Cooper et al. 2010].

3. Trabalhos Relacionados

Na literatura alguns trabalhos relacionados também avaliam o desempenho da ferramenta Hbase por meio de *benchmarking* com outros Bancos de Dados ou ainda colocando alguma condição sobre os dados a serem manipulados.

Em seu estudo, [Jogi and Sinha 2016] realizam a comparação MySQL com Cassandra e Hbase em operações *heavy* write. Foi elaborada uma aplicação *web* REST (*Representacional State Transfer*) para recebimento dos dados e armazenamento no Banco de Dados. Em conclusão, o Cassandra apresentou melhor desempenho entre os três bancos quanto a velocidade de escrita, enquanto o Hbase foi duas vezes mais rápido que o MySQL. Para os autores [Jogi and Sinha 2016], o comportamento do Cassandra ocorre devido a incorporação de características do *BigTable* do Google e do DynamoDB.

[Swaminathan and Elmasri 2016] conduziram uma análise de escalabilidade nos Bancos de Dados: Hbase, Cassandra e MongoDB. Para isso foi utilizado o *framework* YCSB com diferentes cargas de trabalho e conjunto de dados, no intuito de evidenciar as vantagens e desvantagens de cada ferramenta para um cenário específico.

Segundo [Waage and Wiese 2014], a confiabilidade para o armazenamento "em nuvem" é um dos pontos chaves para adoção de tecnologias não-relacionais. Desse modo, é proposto que os dados sejam criptografados de modo prévio. Para avaliar o impacto da criptografia, foi realizado um estudo com os Bancos de Dados Cassandra e Hbase. Para tal fo 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 128, 192 e 256 bits de comprimento. Em suma, foi relatada uma redução no desempenho médio do *cluster*, o qual é independente do tamanho da chave de encriptação.

Na apresentação do *framework* YCSB, [Cooper et al. 2010] aplicaram *benchmarking* nos Banco de Dados: Cassandra, Hbase, Yahoo!'s PNUTS e o Sharded MySQL, para exemplificar o uso do *framework* de maneira prática. A partir de testes de performance e escalabilidade, observaram que o Cassandra e Hbase apresentam maior latência para operações *read* e menor latência para operações *update* e *write* em relação ao PNUTS e MySQL, enquanto o PNUTS e Cassandra melhor escalabilidade, em detrimento ao HBase, quando o número de servidores no *cluster* aumenta proporcionalmente a carga de trabalhos.

Diferentemente dos estudos presentes na literatura, este trabalho avalia o potencial do escalonamento horizontal de um *cluster* do Banco de Dados Hbase, através de um *benchmarking* sobre diversos cenários, de modo a criar um cenário, em que aplicações e serviços se mantenham eficientes e disponíveis independentemente do tamanho da demanda de armazenamento e consultas. Concomitantemente, também é verificado aspectos de eficiência e disponibilidade.

4. Configuração do Ambiente de Experimentação

Os testes conduzidos foram executados em um *cluster* composto por sete computadores todos utilizando o Sistema Operacional Linux CentOS 6.6, sendo um deles o nó mestre denominado hpcdmc e os seis caracterizados como nós escravos denominados de n01 a n06, as demais configurações são descritas a seguir:

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 (n01 a n06):

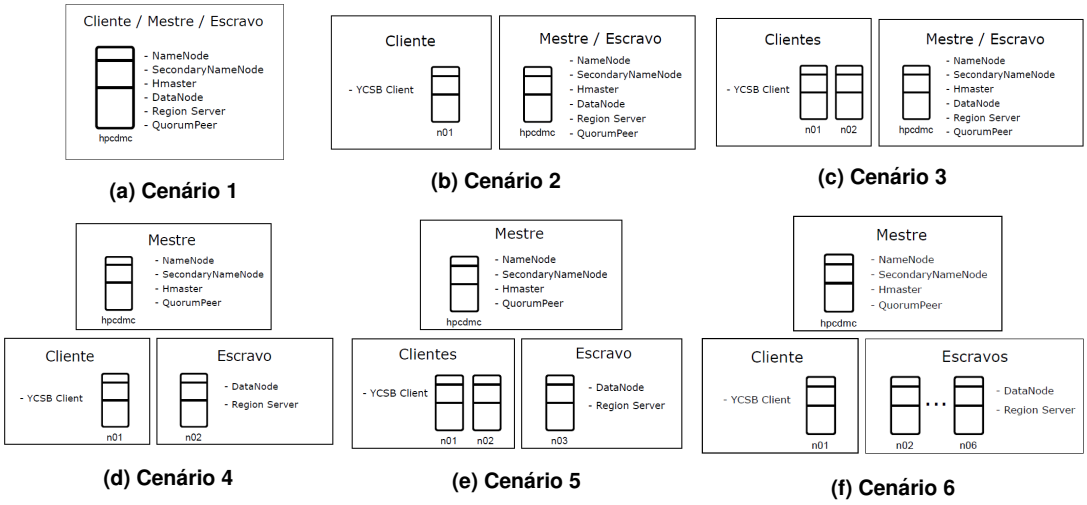


Figura 2. Configuração dos seis cenários de testes.

- 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

Para a execução do experimento foram utilizados os seguintes softwares: Hadoop 2.7.3; HBase 1.2.4; ZooKeeper 3.4.10; YCSB 0.12.0. As versões dos softwares utilizados para a execução dos testes foram escolhidas por serem as mais estáveis e atuais, até o momento do desenvolvimento deste trabalho, de acordo com as informações obtidas pela documentação dos desenvolvedores de cada aplicação.

Primeiramente foram organizados cinco cenários, de organização para o *cluster*, nos quais foram executados os testes iniciais. Com base nesses cinco cenários e nos testes realizados 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. De acordo com os resultados dessas análises foi idealizado um sexto cenário, considerado o cenário ideal para a configuração do *cluster*.

Para os 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.

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 – Figura 2a.

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 – Figura 2b.

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 – Figura 2c.

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 – Figura 2d.

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 distribuío. 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 – Figura 2e.

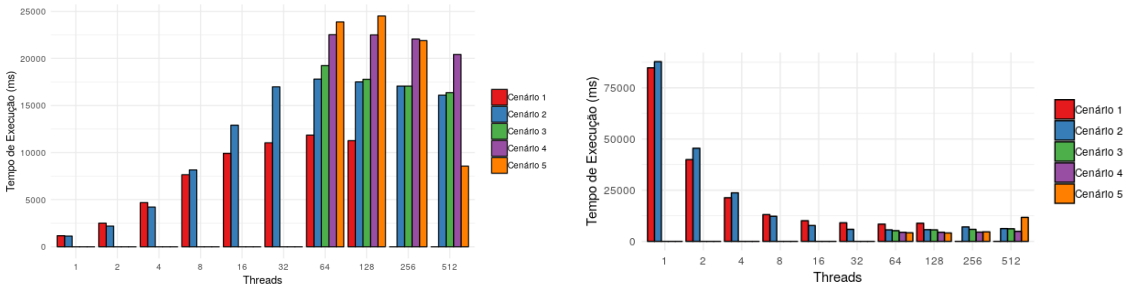
4.6. Cenário 6

Os resultados dos testes realizados nos cenários anteriores apoiam as decisões quanto a estrutura geral 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 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.

5. Resultados e Discussões

5.1. Testes dos Cenários de 1 a 5

Nos resultados obtidos nos cenários 1 e 2 indicam um crescimento no desempenho médio do *cluster* nas execuções com até 64 *threads* – Figuras 3a e 4a. No cenário 1, entre 1



(a) Desempenho médio. **(b) Latência.**

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

e 64 *threads* ocorreu um aumento de aproximadamente 90% no desempenho médio e, enquanto no cenário 2 ocorreu, para a mesma quantidade de *threads*, um aumento de aproximadamente 93%.

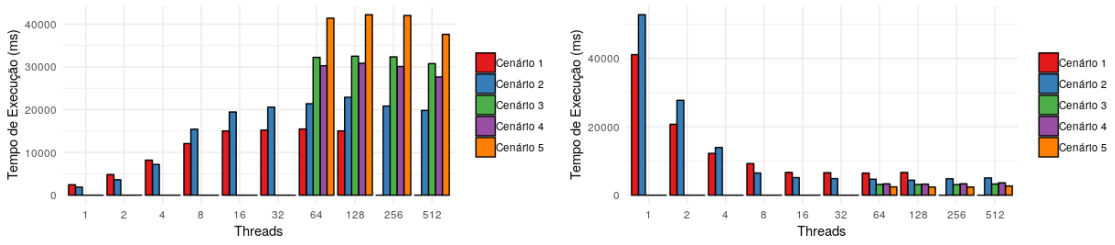
Na instanciação de mais de 64 *threads*, no cenário 1 observa-se queda no desempenho médio de aproximadamente 5% nos testes com 128 *threads* e, no cenário 2, também quedas nas execuções de 128, 256 e 512 *threads*, sendo a mais expressiva de aproximadamente 10% nas execuções com 256 *threads*. Ambas as porcentagens foram calculadas comparando os resultados às execuções com 64 *threads*.

Da mesma forma, observa-se queda na latência nos testes dos cenários 1 e 2 até as execuções com 64 *threads* – Figuras 3b e 4b. Dado a medida que o desempenho médio aumenta, ou seja, o *cluster* executa mais operações por segundo, o tempo de execução do teste diminui. Para o 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* foi obtido para os 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*, 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 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 esse 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



(a) Desempenho médio. **(b) Latência.**

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

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.

A partir dos testes da *workload* 100% *write* – 100.000 registros, pode-se concluir que o cenário 5, para execuções com 64 e 128 *threads* obteve 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* – Figura 3b.

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 – Figura 4a.

Na *workload* 100% *read* – 100.000 registros – Figura 4b, 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.

Com os resultados 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

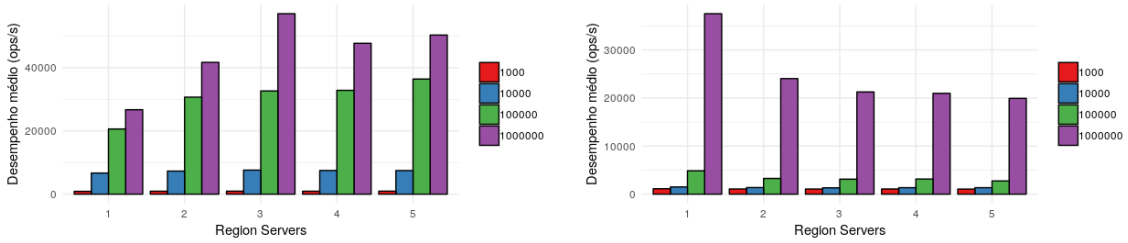
essas duas quantidades de *threads* não são expressivas, sendo de aproximadamente 1%, para os testes da *workload* 100% *write* e 100% *read*, tando para o desempenho médio quanto para a latência.

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% *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.

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. Testes do Cenário 6

Inicialmente, a respeito do desempenho médio – Figura 5a, 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 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.



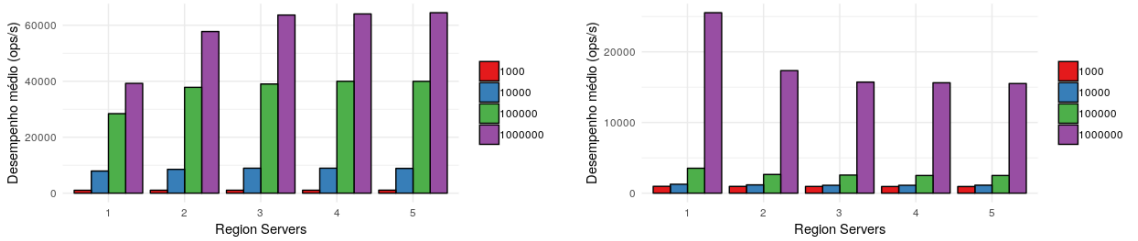
(a) Desempenho médio. **(b) Latência.**

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

A partir dos resultados dos testes da *workload* 100% *write* – Figura 5b, 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 – Figura 6a, quanto a latência – Figura 6b.



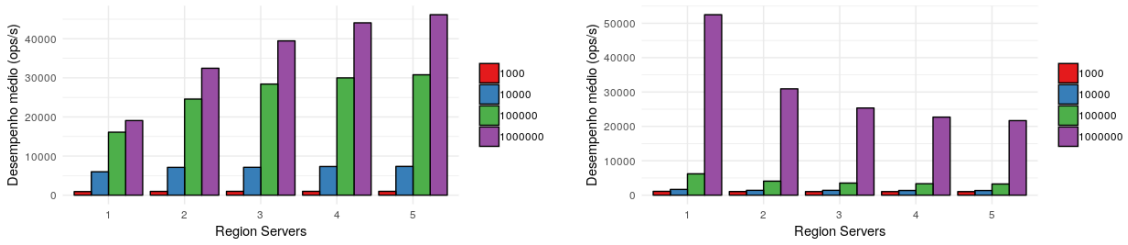
(a) Desempenho médio. **(b) Latência**

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

Para a *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% – 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 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 – Figura 7a, observou-se novamente que, a escalabilidade hori-



(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*.

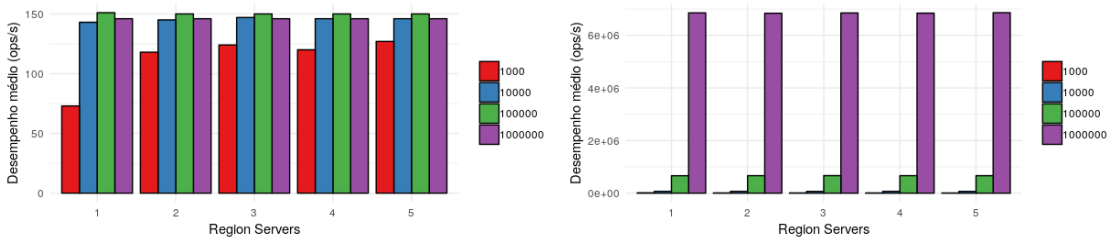
zontal é mais evidenciada nos cenários onde o conjunto é igual a 1.000.000 de registros, sendo o desempenho médio dos testes com 5 *region servers* maior que os resultados dos testes com 4, 3, 2 e 1 *region servers* aproximadamente 4,5%, 14,5%, 29,7% e 58,6% respectivamente. Para os testes onde os conjuntos 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%.

Considerando os resultados dos testes da *workload* 100% *read/modify/write* quanto a latência – Figura 7b. 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. Considerando os resultados dos testes da *workload* 100% *scan* referentes ao desempenho médio – Figuras 8a e latência – Figura 8b. Exceto nos testes cujos conjuntos de dados foram iguais a 1.000 registros, não houveram alterações significativas no desempenho 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.



(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*.

6. Considerações Finais

Este trabalho analisou a escalabilidade horizontal de um *cluster* Hbase, submetendo o banco de dados a um *benchmarking* apoiado pela ferramenta YSCB. Foram conduzidos testes com diferentes números de nós-escravos e tamanho do conjunto de dados.

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 aumento do 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 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. Contudo, os testes da *workload* 100% *scan* mostraram que não há melhora no desempenho do *cluster* para busca de registros, de modo independente do tamanho do conjunto de dados, devido a busca linear implementada pela ferramenta Hbase não ganhar eficiência a medida que novos nós são adicionados.

Os trabalhos futuros podem explorar o aumento do fator de replicação, não empregado neste trabalho, ao analisar o impacto que as operações adicionais de replicação podem causar à eficiência do *cluster*, considerando a inserção de novos nós.

Referências

Brito, R. W. (2010). Bancos de dados nosql x sgbds relacionais: análise comparativa. *Faculdade Farias Brito e Universidade de Fortaleza*.

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.

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

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.

Hadoop, A. (2016). Hdfs architecture. `https://hadoop.apache.org/docs/r3.1.2/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html`. Online; accessed 03 May 2019.

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.

Index, C. V. N. (2013). 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. In *2016 3rd International Conference on Recent Advances in Information Technology (RAIT)*, pages 586–590. IEEE.

Mohamed, M., G. Altrafi, O., and O. Ismail, M. (2014). Relational vs. nosql databases: A survey. *International Journal of Computer and Information Technology (IJCIT)*, 03:598.

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.