UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

SAVIO DE CASTRO LIMA

PROVISIONAMENTO E PROCESSAMENTO DE CONSULTAS NO CASSANDRA EM UM AMBIENTE DE NUVEM CONSIDERANDO UM SLA

QUIXADÁ
2014

SAVIO DE CASTRO LIMA

PROVISIONAMENTO E PROCESSAMENTO DE CONSULTAS NO CASSANDRA EM UM AMBIENTE DE NUVEM CONSIDERANDO UM SLA

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Bacharelado em Sistemas de Informação da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: Computação

Orientadora MSc. Ticiana Linhares Coelho da Silva

QUIXADÁ
2014

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca do Campus de Quixadá

L732p Lima, Sávio de Castro
Provisionamento e processamento de consultas no Cassandra em um ambiente de nuvem considerando um SLA / Sávio de Castro Lima. – 2014.
37 f. : il. color., enc. ; 30 cm.

Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2014.
Orientação: Profa. Me. Ticiana Linhares Coelho da Silva
Área de concentração: Computação

1. Computação em nuvem 2. Acordo de nível de serviço 3. Clusters I. Título.

CDD 004.36

SAVIO DE CASTRO LIMA

PROVISIONAMENTO E PROCESSAMENTO DE CONSULTAS NO CASSANDRA EM UM AMBIENTE DE NUVEM CONSIDERANDO UM SLA

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Bacharelado e Sistemas de Informação da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: computação

Aprovado em: 04 / dezembro / 2014.

BANCA EXAMINADORA

_____

Profª. MSc. Ticiana Linhares Coelho da Silva
(Orientadora)
Universidade Federal do Ceará-UFC

_____

Prof. MSc. Régis Pires Magalhães
Universidade Federal do Ceará-UFC

_____

Prof. Dr. Flávio Rubens de Carvalho Sousa
Universidade Federal do Ceará-UFC

Aos meus pais.

AGRADECIMENTOS

Aos meus pais, amigos e a todos que participaram na minha trajetória acadêmica.

" A persistência é o caminho do êxito."
(Charles Chaplin)

RESUMO

A Computação em Nuvem é um paradigma promissor na oferta de serviços. A sua principal característica é a elasticidade, ou seja, a capacidade do sistema prover e remover recursos conforme a necessidade. Além disso, os ambientes de nuvem oferecem serviços baseados em um acordo de nível de serviço (Service Level Agreement - SLA). Esse trabalho explora a proposta de processamento de consultas em ambientes de nuvem utilizando como banco de dados o Cassandra. Cada consulta a ser processada está associada a um SLA e para que este seja atendido é proposta uma estratégia de provisionamento durante o processamento da consulta. Para a realização dos experimentos foram utilizadas duas máquinas virtuais da infraestrutura de nuvem da UFC Fortaleza. Foi utilizado um benchmark de dados YCSB, um cliente de testes da nuvem onde realiza diversas cargas de trabalho de gerar diversos tipos de volume de dados. Posteriormente a instalação do Cassandra em cada máquina virtual, para formar um cluster. Após foi preparado o ambiente com a criação do keyspace e família de colunas nas duas respectivas máquinas. O próximo passo foi a geração de cargas para preencher as tabelas do banco. A partir então foi possível realizar o processamento das consultas, calculando o SLA baseando-se no tempo de resposta uma consulta. Com um segundo cenário foi realizado o particionamento de cada consulta, de tal sorte que cada nova consulta possa ser realizada dentro do tempo estipulado pelo SLA.

Palavras chave: NoSQL. Cassandra. Computação em Nuvem

LISTA DE ILUSTRAÇÕES

LISTA DE TABELA

SUMÁRIO

1 INTRODUÇÃO

Grandes volumes de dados têm sido gerados por aplicações web, redes sociais, dispositivos como GPS, radares, entre outros. Essas grandes fontes de dados exigem tecnologia eficiente para manipulá-las. Os bancos de dados NoSQL (Not Only SQL) têm se tornado uma solução interessante neste contexto, pois permitem o manuseio dos mais diversos tipos de dados. Um exemplo de banco de dados NoSQL é o Cassandra, que diferencia-se de bancos relacionais por não possuir esquema de tabela fixa, como também não oferece suporte a junções SQL. O Cassandra é um sistema de banco de dados distribuído altamente escalável, além de possuir outras características como ser descentralizado, eficiente no processo de escrita e leitura dos dados e tolerante a falhas. O banco Cassandra pode ser executado sob uma infraestrutura distribuída, como um ambiente de computação em nuvem ou em um *cluster* de máquinas.

A computação em nuvem está em grande evidência por proporcionar uma alta escalabilidade e elasticidade, tornando-se uma boa alternativa quando se trata da necessidade de se trabalhar com uma grande base de dados, visando eficiência no processamento dos mesmos. Segundo Zhao, Hu e Meng (2010, p.18), "computação em nuvem nada mais é do que uma grande quantidade de computadores responsável pelo processamento e armazenamento de dados".A computação em nuvem provê recursos computacionais que podem ser alocados ou realocados conforme a necessidade de cada usuário (MELL; GRANCE, 2009). Essa característica é conhecida como elasticidade.

Para prover a elasticidade, é necessário monitorar continuamente se os sistemas estão precisando de recursos a fim de realizar um uso melhor da infraestrutura que a computação em nuvem provê, alocando recursos quando necessário ou diminuindo para evitar desperdícios de recursos.

A demanda dos usuários por recursos computacionais é observada na computação em nuvem mediante um Service-level Agreement (SLA). O SLA também é conhecido como nível da qualidade de serviço, em que provedor e usuário descrevem cada ponto do serviço oferecido, e como atendê-los. Segundo SOUSA (2012), um SLA é composto de vários parâmetros que serão apresentados na Seção 4.3.

Este trabalho irá explorar o uso da computação em nuvem como infraestrutura para processamento de consultas em banco de dados Cassandra. Cada consulta estará associada a um SLA. A métrica principal a ser utilizada no SLA é o tempo de resposta da consulta. Como as máquinas da infraestrutura de Nuvem podem apresentar desempenho heterogêneo com

relação ao tempo de CPU e I/O, é necessário que o processamento de consultas no cluster Cassandra seja monitorado de tal sorte que se prevista uma possível violação do SLA de uma consulta, o provedor da nuvem possa ser alertado e provisione mais máquinas para o processamento dessa consulta.

O público-alvo a ser beneficiado com este trabalho é a comunidade de banco de dados, mais especificamente pesquisadores NoSQL, oferecendo mais conhecimento a respeito do processamento de consultas em ambiente distribuído.

Nas próximas seções, serão apresentados mais detalhes sobre o trabalho. A seção 2 apresenta os trabalhos relacionados juntamente com a contribuição de cada trabalho. Na seção 3 é apresentada a fundamentação teórica com os principais conceitos relacionados, na seção 4 contém os procedimentos metodológicos mostrando cada passo da execução do trabalho. Já na seção 5 são mostrados os resultados alcançados. Na seção 6 um relato do que foi alcançado e na seção 7 as considerações finais.

2 TRABALHOS RELACIONADOS

Esta seção apresenta os principais trabalhos relacionados ao estudo que foram identificados.

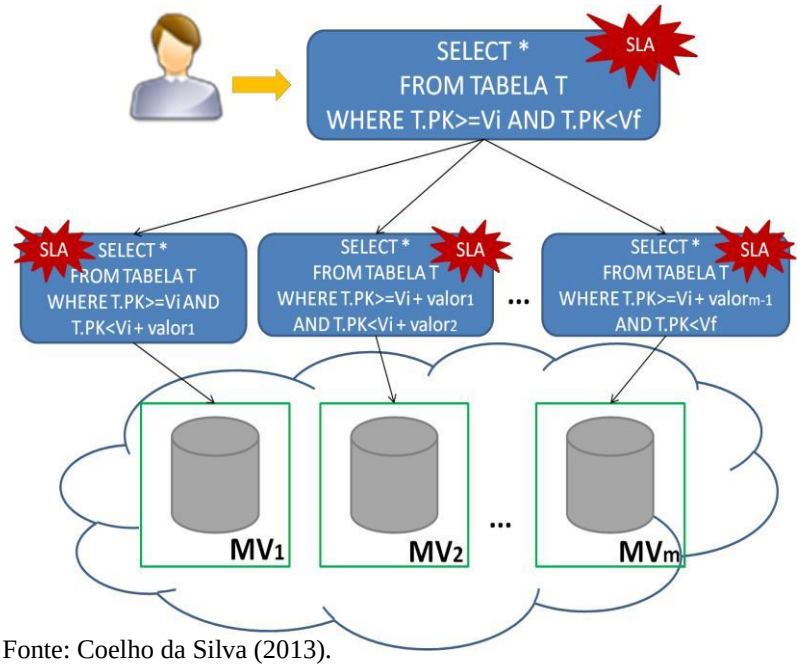
2.1 Processamento Elástico e Não-intrusivo de Consultas em Ambientes de Nuvem considerando o SLA

A monografia Processamento Elástico e Não-intrusivo de Consultas em Ambientes de Nuvem considerando o SLA, de Coelho da Silva (2013), mostra a implantação de um método de monitoramento não-intrusivo e contínuo, utilizando um SGBD relacional em cada máquina contida em uma infraestrutura de computação em nuvem. Os autores buscaram minimizar o custo com o provisionamento de máquinas, e assim tirar melhor proveito da estrutura do provedor de serviço de nuvem. Além disso, buscam atender o SLA de cada consulta.

Também faz parte do objetivo desse trabalho, minimizar a punição para o caso do SLA não ser atendido. Outra contribuição de Coelho da Silva (2013) é o método proposto para o provisionamento de máquinas virtuais durante o processamento de consultas, considerando o SLA e desempenho das máquinas (velocidade de recuperação de tuplas por segundo). Outra contribuição importante foi a estratégia de monitoramento do processamento das consultas a fim de que o SLA de cada consulta fosse atendido.

A figura 1 mostra uma consulta do tipo *select-range* que será usada como entrada pela solução de Coelho Da Silva (2013). A consulta é particionada pela chave primária da tabela e distribuída entre as máquinas virtuais. Cada máquina virtual processa partições diferentes da consulta, observando o SLA da consulta original. As bases de dados são réplicas totais.

Figura 1 – Exemplo Particionamento de Consulta

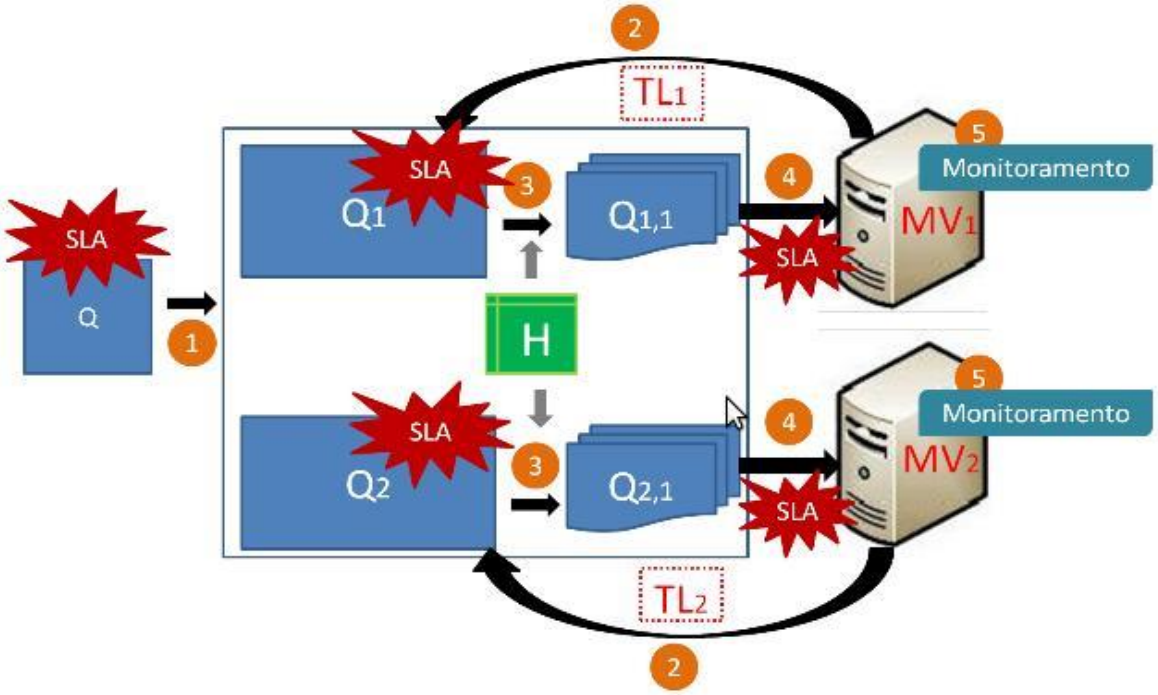


Fonte: Coelho da Silva (2013).

A figura 2 ilustra toda a solução proposta em Coelho da Silva (2013). Uma consulta Q dada como entrada é particionada em Q1 e Q2. Tanto no processamento de Q1 quanto no de Q2, o monitoramento é aplicado. Q1 é executada na máquina mv1 e a segunda Q2 é executada na máquina mv2, onde ocorre o monitoramento observando o SLA. O SLA descreve o tempo máximo para que todas as partições ou subconsultas sejam executadas. O monitoramento é realizado considerando a velocidade de recuperação das tuplas por segundo em cada máquina virtual. Dessa forma, cada máquina consegue prever se o SLA será atendido ou não, baseado na sua velocidade e nas partições ou subconsultas a serem ainda executadas.

As semelhanças desse trabalho em relação ao de Coelho Da Silva (2013) são: (i) a utilização de uma infraestrutura de computação em nuvem, (ii) consultas do tipo *select range*, (iii) SLA descrevendo métricas e critérios entre o provedor e cliente, e por fim (iv) tempo de resposta da consulta como a métrica do SLA a ser utilizado para o monitoramento das consultas, (v) Algoritmo para realizar o monitoramento do processamento da consulta.

Figura 2 – Exemplo do Processo de Monitoramento



Fonte: Coelho da Silva (2013)

As diferenças mais importantes entre este trabalho em relação a Coelho da Silva (2013) são: (i) benckmark de dados utilizado será um apropriado ao paradigma NoSQL (ii) Não será oferecida como solução neste trabalho, uma estratégia de provisionamento de máquinas virtuais mas, caso o SLA seja violado deve ser exibido ao usuário que o SLA não foi atendido (iii) utilização da tecnologia NoSQL utilizando o banco de dados Cassandra. Não serão utilizadas réplicas totais das bases como em Coelho da Silva (2013). O Cassandra possuiu seus algoritmos de particionamento de dados, entre os principais estão o *Random Partitioner* e *Order preserving Partitioner*. Os dados serão particionados nas bases distribuídas seguindo o *Order Preserving Partioner* que será explicado adiante.

Um segundo trabalho, Qualidade de Serviço para Banco de Dados na Nuvem (SOUSA, 2012), mostra como se encaixa o SLA em uma infraestrutura básica de Computação em Nuvem, além de apresentar os diferentes tipos de SLA empregados na Nuvem e suas métricas utilizadas. Sousa (2010) destaca a abordagem utilizada para a qualidade de serviço dos bancos de dados na nuvem, e mostra as principais métricas utilizadas como, tempo de resposta, rendimento, disponibilidade e consistência. Podemos citar como principais contribuições Sousa (2012): (i) um SLA específico para banco de dados, (ii) definição de um conjunto de métricas para o monitoramento dos serviços envolvendo banco de dados.

Como semelhanças do trabalho com o de Sousa (2012) podemos citar o uso de um SLA para o funcionamento de banco de dados na nuvem com todos os detalhes do serviço que irá ser prestado, e acordado entre o provedor e o cliente, além da métrica, o tempo de resposta como item principal utilizado.

Entretanto, será utilizada apenas um tipo de métrica para o SLA, enquanto Sousa (2012) aborda vários tipos de métricas em seu detalhamento. Além disso, aborda o monitoramento no processamento de consultas em bases Cassandra, o que não é feito em Sousa (2012).

3 FUNDAMENTAÇÃO TEÓRICA

Nas subseções seguintes são descritos com mais detalhes os conceitos mais importantes relacionados a este trabalho. Na primeira subseção contém detalhes do Cassandra, como a sua estrutura em geral, arquitetura utilizada para o armazenamento dos dados.A segunda subseção mostra o monitoramento e processamento de consultas. Na terceira subseção são apresentados os princípios de um SLA, e dos detalhes para garantir que os itens acordados entre provedor e cliente sejam atendidos.

.

3.1 Cassandra

Cassandra pode ser definido como um projeto de banco de dados *open-source*, distribuído, descentralizado, elasticamente expansível, altamente disponível.É um banco de dados orientado a colunas, que possui tolerância a falhas, utiliza itens da arquitetura Dynamo, da Amazon e o seu modelo de dados é baseado no Bigtable da Google (CASSANDRA, 2010). Foi criado pelo Facebook sendo utilizado por diversos sites populares, empresas como Netflix, Twitter, Digg, Reddit dentre outros.

Sendo um banco distribuído, permite que grandes volumes de dados sejam armazenados e gerenciados através da sua estrutura composta por nós onde esses dados são distribuídos. À medida que o volume de dados aumenta, a escalabilidade é outro fator importante no Cassandra, que permite esse aumento gradativo de dados sem afetar o desempenho do banco. Possui também um alto poder de gravação e leitura eficientes.

Outra característica forte do Cassandra é a tolerância a falhas, aumentando a confiabilidade do banco, pois os dados são replicados entre os nós que formam o cluster. Caso haja uma falha em um dos nós, o seu funcionamento não é alterado, podendo o dado solicitado ser buscado nos outros nós.

Cassandra se baseia no princípio de família de colunas, que são estruturas para armazenar um conjunto ordenado de linhas, onde cada linha é composta por um conjunto ordenado de colunas fazendo uma comparação com o modelo relacional, uma família de coluna é similar a uma tabela. Linha é a menor unidade para o armazenamento da informação do Cassandra, esse termo podendo ser também conhecido como célula onde a informação será guardada.

A coluna, por sua vez, é composta por um nome, um valor, e um *timestamp*. O nome corresponde à chave que será atribuída ao dado, e o valor corresponde a informação real, associada a chave em questão. Já o *timestamp* corresponde a um conjunto de valores para diferenciar versões de um mesmo dado e evitar conflitos. Possui também como componentes da sua estrutura o *keyspace*, que está associado ao nome do banco.

Laskhman e Avinash (2010) mencionam o Cassandra como um sistema de armazenamento distribuído ideal para o gerenciamento de grandes volumes de dados, dispostos em uma infraestrutura contento diversos nós onde esses dados são espalhados, possuindo uma alta escalabilidade e confiabilidade.

Possui também, segundo Laskhman e Avinash (2010), as suas próprias técnicas de sistemas distribuídos usados que são: particionamento dos dados, replicação e redimensionamento. Todos esses itens funcionando juntamente com as solicitações de leitura e escrita.

Um item importante na composição do Cassandra é o *cluster*, ou também conhecido como *cluster* Cassandra, uma infraestrutura em formato de anel que agrupa um conjunto de nós, cada nó contendo réplicas do banco a fim de que caso haja uma falha em um dos nós, o desempenho não seja afetado podendo algum dado em específico ser buscado em outro nó. O *cluster* pode ser expandido com a inserção de mais nós, criando um ambiente distribuído. O maior cluster Cassandra é formado por 300 terabytes distribuídos entre 400 servidores.

O Cassandra possui a sua própria linguagem de consulta, a CQL(Cassandra Query Language), semelhante a linguagem SQL. Não há suporte a joins, como também não suporta chave estrangeira. Para resolver a esse problema, é possível a criação de chave composta, utilizando atributos que compõem a família de colunas. Também não suporta transações ACID, em vez disso utiliza o teorema “CAP” (Consistência-Alta Disponibilidade-Tolerância

a Particionamento) de Eric Brewer o qual menciona que os bancos NoSQL somente podem cumprir duas das três características, no qual o Cassandra prioriza a disponibilidade e a tolerância ao particionamento, perdendo em desempenho em relação a consistência dos dados, daí a importância da replicação dos dados entre os nós que compõem o cluster.

No que diz respeito a consistência em operações de leitura ou escrita, o Cassandra possui três níveis principais: one, quorum e all. No nível one, em relação a uma operação de escrita, esse tipo garante que o dado foi gravado na memória de pelo menos uma réplica, e na leitura o dado que se deseja buscar será retornado no primeiro nó que o contenha. Já em relação ao tipo quorum, a escrita o dado é feita em um número maior de réplicas, enquanto na leitura, o dado retornado é aquele que mais recentemente lido. E no nível all, tanto a operação de leitura como escrita utilizam todas as réplicas disponíveis.

Como características de sua arquitetura, cada nó que compõe o cluster é similar aos outros nós, mostrando semelhanças à arquitetura P2P. Para a comunicação dos nós nessa estrutura, o Cassandra possui o protocolo gossip(fofoca), no qual cada nó mantém atualizada as informações dos outros nós como também na inserção de um novo nó,as informações em geral do cluster, os dados armazenados, são transmitidos por esse protocolo. Da mesma forma, quando um nó é removido as novas informações passam de nó por nó por todo o cluster.

3.2 Processamento e Monitoramento de Consultas

O processamento de consultas com o Cassandra em um ambiente distribuído pode haver a necessidade de boas estratégias de particionamento de dados, pois dependendo de como os dados estão distribuídos no cluster, pode afetar o desempenho das consultas.

Um dos principais quesitos envolvidos nesse processo é o custo. Segundo Özsu e Valduriez (2011), o tempo de resposta de uma consulta pode ser alto, caso tenham muitos sítios para acessar. Segundo Coelho da Silva (2013), o processamento de consultas em um ambiente distribuído, tem como objetivo minimizar o custo da execução da consulta que inclui (i) o custo de processamento total que é a soma dos custos do processamento da consulta local nos sítios participantes e (ii) o custo da comunicação.

Durante o processamento da consulta, o custo de comunicação entre os nós envolvidos no processo também é visto com devida importância,pois fatores como a forma como os dados estão dispostos entre os nós, mensagens de transferência envolvidas e a comunicação disponível de rede.Para ARNAUT; SCHROEDER;HARA(2011) a forma como estão localizados os dados em ambientes distribuídos pode minimizar o custo de comunicação

entre eles.

Já (Chhana Ray, 2009),um outro objetivo importante envolvendo um ambiente de processamento de consultas distribuído, é o aumento do paralelismo de execução envolvendo uma determinada consulta, a fim de chegar um tempo de resposta ideal e que não ultrapasse o tempo real.

3.3 Service-Level Agreement(SLA)

A computação em nuvem está em grande evidência na área da Tecnologia da Informação (TI), pois provê elasticidade e disponibilidade de recursos necessários para diferentes aplicações, como também para os bancos de dados na nuvem. Porém alinhar a computação em nuvem, a qualidade de serviço, ou SLA (Service-Level Agreement) tornou essencial para garantir que esses recursos da computação em nuvem sejam atendidos.

Um SLA descreve características acordadas entre provedor de serviço e o cliente buscando garantir requisitos como desempenho, disponibilidade de recursos. É descrita também, no SLA, as métricas a serem utilizadas, e pode-se citar como exemplo na utilização nesse trabalho, o tempo de resposta de uma consulta.

Alguns outros conceitos principais sobre SLA podem ser descritos. Segundo Coelho da Silva (2013), SLA (Service-level Agreement) é a garantia da qualidade de serviço, acordado em um contrato entre o provedor de serviço e seu cliente pelos recursos oferecidos ao mesmo, sejam esses físicos ou virtuais.

Os itens principais que compõem o SLA, segundo SOUSA (2012) são:

- Receita: valor monetário pago pelo usuário ao provedor de serviço por todo o tempo de computação;
- Custo Computacional: valor monetário pago pelos recursos computacionais alocados para processar a carga de trabalho do usuário;
- Objetivo de nível de serviço (Service Level Objective - SLO): é associado a um padrão de aceitação definido pelo cliente, que deve ser satisfeito pelo provedor, como por exemplo, o tempo de resposta;
- Penalidade: valor monetário pago pelo provedor ao cliente por não atender ao SLO

Esses itens que compõem o SLA serão importantes, pois serão utilizados durante

todo momento a fim de verificar o bom andamento das disponibilidades dos recursos acordados.

4 PROCEDIMENTOS

Nesta seção são descritos os procedimentos a serem realizados,divididos em subseções para melhor facilitar o entendimento.

4.1 Criação do Cluster Cassandra

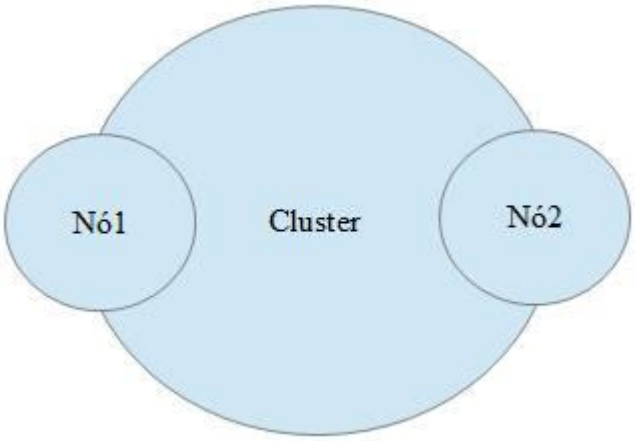
O procedimento de criação e inicialização de um cluster Cassandra é a primeira parte que faremos para configurar corretamente o ambiente necessário para a realização do trabalho. Um cluster Cassandra pode ser definido como uma estrutura organizada em formato de anel, composta por um conjunto de nós interligados, cada nó sendo representado por uma máquina virtual que possui intervalos de valores aos quais serão armazenados dados.

Cada nó, ao ser criado, deve ser configurado antes de ter a sua primeira inicialização. Após esta configuração, será configurado o hardware necessário para cada nó, a fim de tirar melhor proveito na parte de leitura e escrita dos dados. Ao criar o *Keyspace* é necessário definir a estratégia para a alocação da quantidade de réplicas a serem utilizadas pelos nós.

Assim, distribuem-se réplicas da base de dados por esses nós. A infraestrutura de Nuvem da Universidade Federal do Ceará Fortaleza será utilizada para o *cluster*.O Cassandra será instalado em duas máquinas.

A figura 3 ilustra a criação de 2 nós que serão utilizados para o experimento, cada nó ficará responsável por um determinado intervalo de valores que simbolicamente podemos associar como um intervalo de 0 à 50. O primeiro nó estará com o intervalo de 0 a 25, o segundo com o intervalo entre 25 e 50, sendo os dados distribuídos por essas porções de intervalos.

Figura 3 – Cluster Criado



Fonte: Elaborado pelo autor

4.2 Escolher o tipo de Consulta

Ao realizar pesquisas sobre os tipos de consultas possíveis no ambiente Cassandra, juntamente com os detalhes fornecidos pelo *benchmark* do tipo de dados a ser utilizado, essa etapa visa buscar o melhor tipo de consulta baseado no particionamento escolhido para os dados no *cluster* Cassandra.

O particionamento que será utilizado para este trabalho é o OPP (Order-Preserving Partitioner), um tipo de particionamento fornecido pelo Cassandra que preserva a ordem dos dados ao serem distribuídos pelos nós do cluster, o que proporciona benefícios em relação a consultas que envolvam intervalos, como exemplo, s*elect * from tabela where atributo1 >=0 and atributo2<=100* . Ao ser realizado o particionamento, os dados estarão dispostos entre os nós do cluster de tal forma que porções do banco de forma ordenada estarão espalhados entre os nós, facilitando a recuperação das informações de uma determinada consulta.

Diante das observações quanto à maneira em que os dados estarão dispostos na estrutura dos nós do cluster, o tipo de consulta ideal escolhida foi do tipo *select-range*. Podemos definir uma consulta *select-range*, como do tipo em que após a condição WHERE envolva um determinado intervalo entre valores para a recuperação dos dados. A figura 4 ilustra um exemplo.

Figura 4 – Consulta Select-Range

```
SELECT *  FROM tabela T
WHERE T.idade >=18 and T.idade < 50;
```

Fonte: Elaborada pelo autor

4.3 Particionamento dos dados

Essa etapa visa buscar uma melhor maneira de particionamento dos dados, com o menor comprometimento possível de processamento da consulta, como também não violar o SLA.

Partindo de uma base de dados definida, busca-se uma maneira de distribuir os dados entre os nós do cluster. Para isso, o Cassandra provê duas formas principais de particionamento: o Particionamento Randômico (RP) e o Particionador Preservador da Ordem (OPP). Ao definir o tipo de consultas, definimos o particionamento OPP como sendo o tipo ideal OPP.

O OPP distribui os dados de maneira uniforme preservando a ordem, ideal para o tipo de consulta *select-range*, pois ao buscar um determinado intervalo de uma condição da consulta, retornada de maneira simples. Porém essa forma de distribuição dos dados, gera um desbalanceamento do cluster, ocasionando que um determinado nó tenha muito mais dados, do que outro nó, causando mais custos no processamento da consulta e consequentemente podendo ocasionar a violação do SLA. O cenário da Figura 5 mostra o comportamento de tais particionamentos.

Figura 5 – Réplicas do Banco

Fonte: elaborada pelo autor

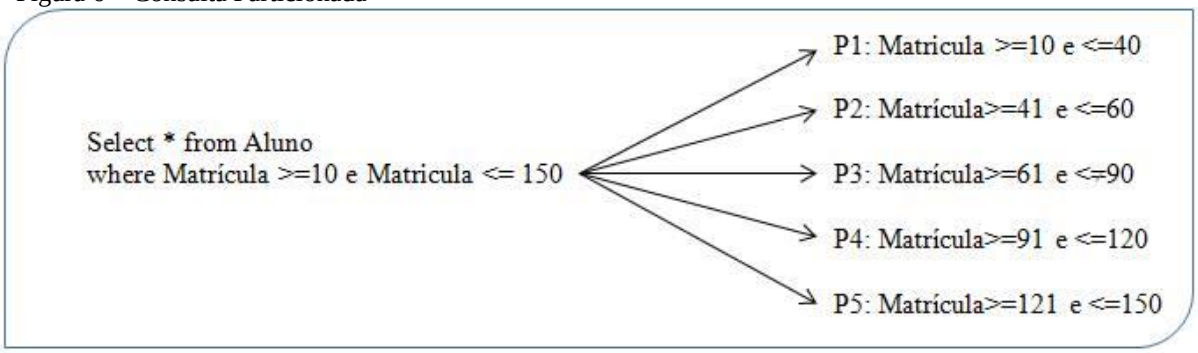
A figura 5 ilustra as réplicas do banco que serão distribuídas entres os nós. Para cada nó, ficará com uma base ilustrada pelo container na figura.

Nas versões mais recentes do Cassandra, o particionamento padrão é o Murmur3 que possui uma função hash mais eficiente do que o particionamento randômico, a função MurmurHash que proporciona a criação de valores de hash de 64 bits, para a distribuição dos dados entre os nós e para uma maior variedade de chaves de linha. A partir desse particionamento, permite a utilização da função TOKEN() para as consultas na interface CQL.

4.4 Elaborar uma estratégia de monitoramento

Durante o processamento da consulta, existe a necessidade de verificação constante do tempo de resposta proposto no SLA com o tempo corrente do processamento a fim de evitar que seja violado. Para isso será proposto a elaboração de uma estratégia de monitoramento a fim de evitar que o tempo de resposta da consulta seja violado. Esse monitoramento é feito verificando o tempo corrente do processamento da consulta e comparando com o tempo de resposta proposto no SLA. Caso preveja a violação desse tempo, a consulta é particionada e redistribuída entre as máquinas.

Figura 6 – Consulta Particionada



fonte : Elaborada pelo autor

A figura 6 representa a consulta sendo particionada após ser realizado o monitoramento e visando a necessidade de obter novas subconsultas a fim de não violar o tempo de resposta da consulta. As 5 partições ficaram com novos intervalos de valores e cada partição ficará distribuída, sendo as partições p1, p2 e p3 direcionadas para o nó 1, a partição p4 e p5 para o nó 2, através do monitoramento realizado para buscar nossas alternativas de consultas e assim cumprir o tempo proposto para a consulta. Para o SLA foi atribuído um tempo de resposta de 100s.

4.5 Definir Benckmark NoSQL para experimentos

Para um melhor desempenho utilizando ao máximo os recursos oferecidos pelo Cassandra, realizou-se a análise dos *benchmarks* de dados disponíveis a fim de encontrar o melhor para as necessidades desse trabalho. Após essa análise, o benchmark ideal a ser utilizado nos experimentos foi o *benchmark* Yahoo Cloud Serving Benchmark (YCSB),porque foi projetado para ser utilizado em bancos NoSQL, além permite a utilização de uma esquema flexível para a criação de uma família de coluna, como também as mais diversas formas de dados, adaptando a necessidade de cada cenário.O benchmark TPCH também poderia ser uma solução, porém é mais utilizado em bancos relacionais.YCSB é um

cliente de teste de serviço na nuvem o qual realiza a leitura, escrita e atualização dos dados baseando-se no tipo de carga de trabalho especificada.

A estrutura consiste de um cliente que gera uma determinada carga de trabalho, e um pacote padrão de carga de trabalho que atuam as partes interessadas no campo de atuação.O YCSB provê uma ótima extensibilidade, pois o gerador de carga de trabalho permite que seja fácil definir novos tipos de carga de trabalho, como também adaptar o cliente para novos sistemas de dados do *benchmark*. Esse *framework* está disponível juntamente com a sua carga de trabalho em código aberto.

4.6 Realizar experimentos para verificar se o sistema está verificando corretamente o SLA da consulta

Os experimentos serão realizados com um conjunto de consultas sendo processadas disponibilizadas por uma carga de trabalho e utilizando a proposta de monitoramento. A partir desse momento, realizando a verificação dessas consultas de tal forma que exista um percentual aceitável delas sejam processadas no tempo proposto. Por exemplo, um percentual de 90% das consultas monitoradas sejam processadas dentro do tempo acordado no SLO.

5 DESENVOLVIMENTO/RESULTADOS

5.1 Instalação das máquinas e criação do cluster Cassandra na nuvem UFC-Fortaleza

Para o início dos experimentos, se fez necessário a instalação do Cassandra em duas máquinas virtuais no ambiente de nuvem da UFC. Cada máquina possui 2Gb de memória. O sistema operacional ubuntu também faz parte das características das máquinas. Por conta dos poucos recursos disponíveis para a criação de mais máquinas na infraestrutura da nuvem, só foi possível utilizar duas.

Para a criação do cluster Cassandra, se fez necessário a configuração do arquivo cassandra.yaml no diretório conf. O nome do cluster, ip da máquina,dentre outros atributos são necessários que sejam inseridos em cada máquina virtual.

5.2 Instalação do Benchmark

O benchmark YCSB foi instalado somente em uma máquina, sendo a carga gerada para as duas máquinas a partir dela. Posteriormente, é realizada a carga de trabalho para gerar os dados. Primeiramente a configuração de cada banco para testar o benchmark, juntamente com a criação do keyspace e família de colunas para o armazenamento dos dados. Cada tipo de criação varia com a carga de trabalho escolhida.

A próxima etapa é selecionar a camada de interface do banco: classes java que quando executadas realizam operações de leitura, escrita, atualização e exclusão. Essas chamadas são geradas pelo benchmark junto a API do banco de dados.

Em seguida, é feita a escolha e execução da carga de trabalho. Os tipos de carga são definidos por classes java workload. Cada carga de trabalho possui duas fases: a fase de definição do tipo de dados a ser inserido e a fase de execução de operações para inserção dos dados. Para os experimentos foi utilizada a carga workload do tipo a. Cada registro inserido possui tamanho de 1KB.

5.3 Conexão ao Banco via JDBC

Para a realização das consultas, foi utilizada a API JDBC para a conexão com o Cassandra. Uma outra alternativa a ser utilizada é o DataStax Java Driver, da própria

comunidade Cassandra. Foi criado um seguinte projeto utilizando a IDE eclipse, contendo as seguintes classes:ConnectionFactory,ConsultaDAO,ConsultaJDBCDAO,ExecutaConsulta:

Figura 7 – Classe ConnectionFactory

```
package br.ufc.si.util;

import java.sql.Connection;

public class ConnectionFactory {

    private static final String host = "localhost";
    private static final String url = "jdbc:cassandra://"+ host + ":9160/usertable";

    public static Connection getConnection() throws SQLException, ClassNotFoundException {
        Class.forName("org.apache.cassandra.cql.jdbc.CassandraDriver");
        return DriverManager.getConnection(url);
    }

}
```

Fonte: elaborada pelo autor

A figura 7 representa a classe ConnectionFactory, apresentando as constantes host e url que representa o ip da máquina remota e a url que conecta ao processo do Cassandra nessa máquina.Logo em seguida um método que retorna um objeto Connection que estabelece uma conexão junto ao Cassandra.

Figura 8 – Interface ConsultaDAO

```
package br.ufc.si.dao;

import java.sql.ResultSet;

public interface ConsultaDAO {

    public ResultSet getResultSerConsulta(String query);

}
```

Fonte: elaborada pelo autor

A figura 8 representa uma interface ConsultaDAO com um único método que retorna o ResultSet da consulta tendo como parâmetros de entrada a string que representa a consulta ao qual queremos processar.

Figura 9 - ConsultaJDBCDAO

```
package br.ufc.si.dao.impl;

import java.sql.Connection;

public class ConsultaJDBCDAO implements ConsultaDAO{

    @Override
    public ResultSet getResultSerConsulta(String query) {
        Connection conexao = null;
        try {
            conexao = ConnectionFactory.getConnection();
            Statement st = conexao.createStatement();
            ResultSet resultSet = st.executeQuery(query);
            st.close();
            resultSet.close();
            return resultSet;
        } catch (ClassNotFoundException e) {
            System.err.println(e.getMessage());
        } catch (SQLException e) {
            System.err.println(e.getMessage());
        }finally{
            try {
                conexao.close();
            } catch (SQLException e) {
                System.err.println(e.getMessage());
            }
        }
        return null;
    }

}
```

Fonte: elaborada pelo autor

A classe ConsultaJDBCDAO da figura 9 representa a implementação da interface ConsultaDAO e seu respectivo método getResultSet() que executa a consulta junto ao banco Cassandra. Inicialmente obtém uma conexão com o banco, através da fábrica de conexões (Connection Factory). Em seguida, o método createStatement() do objeto conexão retorna um objeto Statement que representa uma simples query. Posteriormente o método executeQuery() do objeto st do tipo Statement retorna um ResultSet com o resultado da query. Todo o código está dentro de um bloco try-catch-finally para caso ocorra algum lançamento de exceção a devida mensagem é impressa no console informando a sua causa, e o finally tem a função de fechar a conexão estabelecida independentemente da consulta ter sido executada com sucesso ou alguma exceção ter sido lançada.

Figura 10 – Classe ExecutaConsulta

```java
package br.ufc.si.teste;

import java.sql.ResultSet;

public class ExecutaConsulta {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        ConsultaDAO dao = new ConsultaJDBCDAO();


        String query = sc.nextLine();
        long inicio = System.currentTimeMillis();
        ResultSet result = dao.getResultSerConsulta(query);
        long fim = System.currentTimeMillis();
        if(result != null){
            long tempo = fim - inicio;
            System.out.println(tempo);
        }else{
            System.out.println("Falha ao executar a consulta!");
        }

    }

}
```

Fonte: Elaborada pelo autor

Na figura 10 a classe ExecutaConsulta possui o método main para de fato realizar o processamento da consulta. Inicialmente a consulta desejada é capturada através do teclado por um objeto do tipo Scanner. Em seguida é instanciado um objeto do tipo ConsultaDAO responsável pela manipulação das consultas. Após o usuário digitar a string da query o tempo de consulta é iniciado(long inicio), e então o método getResultSetConsulta(query) é invocado e o ResultSet com o conteúdo da busca será retornado e o tempo da consulta será encerrado(long fim). Se o resultado do ResultSet não for nulo, o tempo da consulta(fim-inicio) será calculado e impresso no console senão, o ResultSet não pode ser instanciado devido a alguma exceção que tenha sido lançada na implementação do método getResultSetConsulta(query). O projeto pode ser baixado através do link ao final desse trabalho.

5.4 Escolha de sete consultas com ranges diferentes

As consultas foram determinadas baseando-se nas chaves de linha geradas pela carga de trabalho. O benchmark também possui o seu conjunto de consultas, porém para os

experimentos foram utilizadas somente as cargas de trabalho para a geração dos dados.A tabela abaixo segue uma descrição de cada uma das consultas.

Consulta1	SELECT key FROM data WHERE TOKEN(key)>=1010099187537061287 and TOKEN(key)=<2032700585171816769
Consulta2	SELECT key FROM data WHERE TOKEN(key)>=1010099187537061287 and TOKEN(key)<=2332700585171816769
Consulta3	SELECT key FROM data WHERE TOKEN(key)>=1010099187537061287 and TOKEN(key)<=3532700585171816769
Consulta4	SELECT key FROM data WHERE TOKEN(key)>=1010099187537061287 and TOKEN(key)<=433270051009918753706128 7
Consulta5	SELECT key FROM data WHERE TOKEN(key)>=1010099187537061287 and TOKEN(key)=<5032700585171816769
Consulta6	SELECT key FROM data WHERE TOKEN(key)>=1010099187537061287 and TOKEN(key)<=6032700585171816769
Consulta7	SELECT key FROM data WHERE TOKEN(key)>=1010099187537061287 and TOKEN(key)=<7592700585171816769

Tabela 1: consultas selecionadas para os experimentos

Cada consulta possui o atributo *key* da família de colunas *data* criada para a geração das cargas de trabalho.Para a utilização da cláusula WHERE em consultas select range, é possível somente em atributos que são chave de partição (partition key) da família de coluna. Para a execução de consultas propostas nos experimentos, foi utilizada a função TOKEN usada após a cláusula WHERE para facilitar a consulta utilizando a chave de partição da família de colunas.

5.5 Execução de cada consulta em uma máquina

Para realizar o cálculo do SLA de cada consulta, foi feito o processamento de cada consulta descrita na tabela 1 junto à uma máquina. A tabela 2 abaixo ilustra cada consulta com seu respectivo SLA.

Consulta	**SLA(milisegundos)**
Consulta1	7090
Consulta2	9138
Consulta3	18886

Consulta4	37990
Consulta5	47185

Tabela 2: valores do SLA para cada Consulta

As consultas 6 e 7 não foram possíveis por conta da falta de recursos da infraestrutura. As demais consultas foram executadas.Os tempos estão descritos em milissegundos. Com o SLA de cada consulta será possível fazer novas consultas particionadas a fim de se recuperar esse novo conjunto dentro do tempo proposto no SLA. O particionamento de cada consulta foi realizado de forma manual.

5.6 Execução das consultas particionadas

Em um segundo cenário, cada consulta foi particionada em duas partes de forma a executar cada nova partição em uma máquina diferente para coletar o novo tempo. A tabela 3 a seguir ilustra as novas consultas com seus respectivos tempos.

Consulta	**Consultas Particionadas**
Consulta1	SELECT key FROM data WHERE TOKEN(key)>=1010099187537061287 and TOKEN(key)<=1521399886354439000
	SELECT key FROM data WHERE TOKEN(key)> 1521399886354439000 and TOKEN(key)<2032700585171816769
Consulta 2	SELECT key FROM data WHERE TOKEN(key)>=1010099187537061287 and TOKEN(key)<=1671399886354439200
	SELECT key FROM data WHERE TOKEN(key)>=1671399886354439201 and TOKEN(key)<=2332700585171816769
Consulta 3	SELECT key FROM data WHERE TOKEN(key)>=1010099187537061287 and TOKEN(key)<=2522601397634755600
	SELECT key FROM data WHERE TOKEN(key)>=2522601397634755600 and TOKEN(key)<=3532700585171816769
Consulta 4	SELECT key FROM data WHERE TOKEN(key)>=1010099187537061287 (key)<=26713998863544390
	SELECT key FROM data WHERE TOKEN(key)>=26713998863544391 (key)<=433270051009918753706128 7

Consulta 5	SELECT key FROM data WHERE TOKEN(key)>1010099187537061287 and TOKEN(key)< 3271399886354438700
	SELECT key FROM data WHERE TOKEN(key)> 3271399886354438700 and TOKEN(key)<5032700585171816769

Tabela 3: Consultas Particionadas

Na tabela 4 ilustra os tempos coletados de cada subconsulta, para verificar se cada uma ocorreu dentro do tempo estabelecido pelo SLA.

Consulta	**Tempo de Execução das Partições(Milisegundos)**	
Consulta1 Particionada	Partição1: 4316	Partição2: 2204
Consulta2 Particionada	Partição1: 3727	Partição2: 4364
Consulta3 Particionada	Partição1: 8283	Partição2: 7446
Consulta4 Particionada	Partição1: 9559	Partição2: 11773
Consulta5 Particionada	Partição1: 12208	Partição2: 9172

Tabela 4: tempos de execução de cada subconsulta

5.7 Simular estratégia de provisionamento

Nessa etapa buscou-se simular uma estratégia de provisionamento em relação a quantidade de máquinas suficientes para retornar uma determinada quantidade de tuplas. Por exemplo, suponhamos que uma máquina possa realizar 0,50 kb/seg. O tamanho de cada tupla da carga de trabalho tem um tamanho de 1kb.

Então dependendo do SLA da consulta, uma máquina poderia ser suficiente para recuperar as tuplas dentro do tempo previsto, caso não seja é então calculada a quantidade de

máquinas suficientes para o processamento da consulta dentro do prazo estipulado pelo SLA. Essa é a estratégia de provisionamento proposta para as consultas utilizadas no trabalho.

6 DISCUSSÃO

Com base nos experimentos realizados, apesar de dificuldades de recursos da infraestrutura, capacidades das máquinas de processamento, foi possível executar um determinado conjunto de consultas através dos dados gerados pelo benchmark. Para uma única consulta foi encontrado um determinado tempo, mas com a mesma consulta particionada foi possível verificar um processamento mais eficiente.

A criação do cluster Cassandra somente foi possível com a disponibilidade da nuvem da UFC em Fortaleza.

Em um primeiro instante foi possível realizar consultas básicas na interface Cassandra CLI, porém para os tipos de consulta select range, somente a interface CQL com mais recursos disponíveis com o auxílio da função TOKEN.

Posteriormente ao processamento das consultas, foi possível analisar como age o SLA em uma consulta simples com o seu determinado tempo de resposta e em um segundo cenário com a mesma consulta sendo agora particionada em duas e distribuída entre as máquinas.

Da mesma forma ao realizar diferentes cargas de trabalho sobre o cluster, foi possível ver a diferença nos tempos de resposta das consultas para as variações das cargas, mas com o número reduzido de máquinas virtuais disponíveis, não foi possível resultados mais expressivos.

7 CONSIDERAÇÕES FINAIS

Esse trabalho teve por finalidade o processamento de consultas utilizando o banco de dados Cassandra com um grande volume de dados, em um ambiente de nuvem utilizando o SLA tendo como item o tempo de resposta da consulta.

Em um primeiro instante foi feita a tentativa de utilização da nuvem do Campus de Quixadá, mas por problemas recorrentes na rede não foi possível começar a mostrar o cenário para os experimentos. As máquinas passaram um período em manutenção, mas novamente não foi possível utiliza-las primeiramente por acesso restrito dentro do campus e posteriormente elas já não estavam disponíveis.

Em uma segunda tentativa, foi feita utilizando máquinas virtuais no virtual box. Mas as máquinas físicas disponíveis não possuíam recursos computacionais suficientes para a execução dos experimentos. Porém não era possível ter uma real noção da execução das consultas visto que outros itens estavam em execução nas máquinas.

Posteriormente, conseguiu-se duas máquinas virtuais da nuvem de fortaleza e partir disso o início da configuração do ambiente para os experimentos.Em um primeiro momento foi feita a configuração para a utilização do particionamento preservador da ordem(OPP), mas foi constatado o seu desuso nas versões mais recentes do Cassandra e por conta de prover um cluster desbalanceado, o que dificultaria na exatidão dos experimentos. Porém em determinados momentos a nuvem de fortaleza chegou a ficar fora do ar por alguns dias impedindo o decorrer dos experimentos.

Uma alternativa foi uma configuração padrão do Cassandra mantendo o particionamento Murmur3, porém dessa vez com os dados replicados nas duas máquinas. Após isso, um estudo de como age no benchmark nas famílias de colunas e a inserção dos dados. Na realização das cargas, foi visto quais tipos de dados foram gerados nas bases de dados. Na busca de iniciar os primeiros testes com consultas básicas analisou-se duas alternativas: Interface CLI ou CQL. O CLI está em desuso pois possui poucos recursos para as consultas, a opção mais adequada foi realizar através da interface CQL. Nela foi possível processar consultas de maneira mais eficiente, utilizando a função TOKEN a qual foi possível realizar utilizando a chave primária da tabela.

Porém, no CQL não era possível obter com mais detalhes o tempo de processamento da consulta, item indispensável para o andamento dos experimentos. Surgiu então a alternativa de utilizar a API JDBC para o Cassandra, criando um pequeno projeto para o processamento das consultas e posteriormente reportar o tempo, sendo uma opção atrativa em vista que é uma alternativa extra de conhecimento para a realização de consultas.

Foi interessante observar as mais diversas opções de consulta que o Cassandra fornece, como também os protocolos utilizados para comunicação entre nós de forma a manter o cluster sincronizado.

Devido ao atraso em ter uma infraestrutura disponível para realizar os experimentos, um dos objetivos de criar um algoritmo de monitoramento ficou como trabalho futuro tendo em vista o pouco tempo disponível para realizar os experimentos mais básicos.

REFERÊNCIAS

COELHO DA SILVA, Ticiana L, Mario A. Nascimento , José Antônio F.de Macêdo, Flávio R.C. Sousa, Javam C. Machado , **Non-Intrusive Elastic Query Processing in the cloud,** *J*ournal of Computer Science and Technology vol. 28, pp. 932-947 , *November 2013* Disponível em: <http: //mdcc.ufc.br/teses/doc_download/199-174-ticiana-linhares-coelho-da-silva> Acesso em: 01 mar. 2014.

SOUSA, Flávio R. C; MOREIRA, Leonardo O.; SANTOS, Gustavo A. C ; MACHADO, Javam C. **Quality of Service for DataBase in The Cloud.** In: Internacional Coference on Cloud Computing and Services Science(Closer),2012,v.1,pp.595-601 Disponível em: < http: //qosdbc.sourceforge.net/qosdbc.pdf > Acesso em: 15 mar 2014 **.**

HEWITT, Eben. **Cassandra: the definitive guide**. O'Reilly Media, 2010

COOPER, Brian F. et al. **Benchmarking cloud serving systems with YCSB.** In: Proceedings of the 1st ACM symposium on Cloud computing. ACM, 2010. p. 143-154 Disponível em: <http:// ipij.aei.polsl.pl/django-media/lecture_file/ycsb.pdf> Acesso em: 20 jun 2014

DATASTAX, **Documentacion Cassandra 2.0.** ,2014 http://www.datastax.com/documentation/cassandra/2.0/cassandra/gettingStartedCassandraIntro.html?q=/documentation/cassandra/2.0/webhelp/upgrade/cassandra/cassandra/install/installJnaTar.html Acesso em : 10 mai. 2014

LAKSHMAN, Avinash; MALIK, Prashant. **Cassandra: a decentralized structured storage system**. ACM SIGOPS Operating Systems Review, v. 44, n. 2, p. 35-40, 2010 Disponível em : < http://dl.acm.org/citation.cfm?id=1773922 > Acesso em : 20 mar 2014.

BRIANFRANKCOOPER, **Running a Workload** ,

SOUSA, Flávio RC; MOREIRA, Leonardo O.; MACHADO, Javam C. **Computação em nuvem: Conceitos, tecnologias, aplicações e desafios**. Ceará: Universidade Federal do Ceará, 2009

Strauch, Christof, Ultra-Large Scale Sites, and Walter Kriha. **NoSQL databases**. Lecture Notes, Stuttgart Media University (2011).

FEATHERSTON, Dietrich. **Cassandra: Principles and application**. University of Illinois, v. 7, p. 28, 2010

Cassandra Query Language (CQL) v3.1.7

Wu, Linlin, and Rajkumar Buyya. **Service level agreement (SLA) in utility computing systems**. arXiv preprint arXiv:1010.2881 (2010).

Projeto Cassandra JDBC, Disponível em: https://github.com/saviolima/CassandraJDBC