UNIVERSIDADE FEDERAL DO CEARÁ
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

TICIANA LINHARES COELHO DA SILVA

PROCESSAMENTO ELÁSTICO E NÃO-INTRUSIVO DE CONSULTAS EM AMBIENTES DE NUVEM CONSIDERANDO O SLA.

FORTALEZA, CEARÁ

2013

TICIANA LINHARES COELHO DA SILVA

PROCESSAMENTO ELÁSTICO E NÃO-INTRUSIVO DE CONSULTAS EM AMBIENTES DE NUVEM CONSIDERANDO O SLA.

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Área de concentração: Banco de Dados

Orientador: Prof. Dr. José Antônio Fernandes de Macêdo

FORTALEZA, CEARÁ

2013

A000z COELHO DA SILVA, T. L..

Processamento Elástico e Não-intrusivo de Consultas em Ambientes de Nuvem considerando o SLA. / Ticiana Linhares Coelho da Silva. 2013.

54p.;il. color. enc.

Orientador: Prof. Dr. José Antônio Fernandes de Macêdo

Dissertação(Ciência da Computação) - Universidade Federal do Ceará, Departamento de Computação, Fortaleza, 2013.

1. SLA 2. Processamento de Consultas 3. Elasticidade I. Prof. Dr. José Antônio Fernandes de Macêdo(Orient.) II. Universidade Federal do Ceará– Ciência da Computação(Mestrado) III. Mestre

CDD:000.0

TICIANA LINHARES COELHO DA SILVA

PROCESSAMENTO ELÁSTICO E NÃO-INTRUSIVO DE CONSULTAS EM AMBIENTES DE NUVEM CONSIDERANDO O SLA.

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação. Área de concentração: Banco de Dados

Aprovada em: __/__/____

BANCA EXAMINADORA

Prof. Dr. José Antônio Fernandes de Macêdo
Universidade Federal do Ceará - UFC
Orientador

Prof. Dr. Javam de Castro Machado
Universidade Federal do Ceará - UFC

Prof. Dr. José Maria da Silva Monteiro Filho
Universidade Federal do Ceará - UFC

Profa. Dra. Carmem Satie Hara
Universidade Federal do Paraná - UFPR

À Deus. Aos meus Pais.

AGRADECIMENTOS

Agradeço a DEUS.

"Mude, mas comece devagar, porque a direção é mais importante que a velocidade."

(Clarice Lispector)

RESUMO

Computação em Nuvem ou *Cloud Computing* é um paradigma promissor de computação orientada a serviços. O seu maior benefício é a elasticidade, isto é, a capacidade do sistema de adicionar e remover recursos automaticamente em tempo de execução. Para isso, é essencial projetar e implementar uma técnica efetiva e eficiente que tire proveito da flexibilidade do sistema. Dessa forma, prover elasticidade requer monitorar continuamente (ou prever) a demanda do sistema por recursos, com objetivo de decidir quando adicioná-los e removê-los.

Este trabalho apresenta um método de monitoramento não-intrusivo e contínuo de SGBDs relacionais em uma infraestrutura de nuvem, visando minimizar a quantidade de máquinas virtuais provisionadas para o processamento de consultas, e consequentemente maximizar o uso eficiente do ambiente do provedor. Além disso, ele visa satisfazer um "acordo de nível de serviço", em inglês service-level agrement (SLA), associado a cada consulta submetida ao sistema. Dessa forma, um objetivo desse trabalho também é minimizar a penalidade paga pelo provedor para os casos em que ocorre a violação do SLA. Além do método de monitoramento, este trabalho também apresenta um método de provisionamento de MVs para o processamento da consulta como contribuições.

Nossa estratégia de monitoramento é aplicada a consultas *select-range* e consultas com *agregação* sobre uma única tabela. Os experimentos foram realizados na infraestrutura de nuvem da Amazon, confirmando que nossa técnica é elástica, permitindo ajustar os recursos alocados no sistema de forma automática e dinâmica, com base no SLA acordado.

Palavras-chave: SLA. Processamento de Consultas. Elasticidade.

LISTA DE FIGURAS

Figura 3.1 Processo de particionamento de Q. 24

Figura 3.2 Variação no número de MVs alocadas pela nossa estratégia. 25

Figura 6.1 Arquitetura do Protótipo 40

Figura 6.2 Número de nós para diferentes valores de *SLO* em uma distribuição Poisson usando consultas *Select-range* cujo predicado é sobre atributo que é não chave primária. 43

Figura 6.3 Número de nós para diferentes valores de *SLO* em uma distribuição Uniforme usando consultas *Select-range* cujo predicado é sobre atributo que é não chave primária. 44

Figura 6.4 Número de nós para diferentes valores de *SLO* em uma distribuição Poisson usando consultas *Select-range* cujo predicado é sobre atributo que é chave primária. 45

Figura 6.5 Número de nós para diferentes valores de *SLO* em uma distribuição Uniforme usando consultas *Select-range* cujo predicado é sobre atributo que é chave primária. 46

Figura 6.6 Número de nós para diferentes valores de *SLO* em uma Distribuição Poisson usando consultas com *agregação*. 47

Figura 6.7 Número de nós para diferentes valores de *SLO* em uma Distribuição Uniforme usando consultas com *agregação* 48

LISTA DE TABELAS

Tabela 3.1 Notação 22

Tabela 4.1 Análise Comparativa entre os trabalhos relacionados. 31

Tabela 6.1 Comparação da execução da solução proposta com variação de valores de *SLO* para consultas *Select-range* cujo predicado é sobre atributo não chave usando Distribuição de Poisson e Uniforme. Em todos os casos, *o método não resultou nenhuma penalidade*. 43

Tabela 6.2 Comparação da execução do método proposto neste trabalho com variação de valores de *SLO* para consultas *Select-range* cujo predicado é sobre atributo que é chave primária usando Distribuição de Poisson e Uniforme. Em todos os casos, *o método não resultou nenhuma penalidade*. 44

Tabela 6.3 Comparação da execução do método proposto neste trabalho com variação de valores de *SLO* para consultas com *agregação* usando Distribuição de Poisson e Uniforme. Em todos os casos, *o método não resultou nenhuma penalidade*. 46

Tabela 6.4 Comparação da média do número de MVs do método proposto neste trabalho, e o número mínimo de MVs que pode ser usado para que não haja penalidades. 47

SUMÁRIO

1 INTRODUÇÃO 12

1.1 Motivação 13

1.2 Contribuições 14

1.3 Publicações 14

1.4 Estrutura da dissertação 15

2 FUNDAMENTAÇÃO TEÓRICA 16

2.1 Preliminares 16

2.2 Processamento de Consultas Distribuído 16

2.3 Computação em Nuvem 17

2.4 Conclusão 19

3 FORMULAÇÃO DO PROBLEMA 20

3.1 Modelo de Custo 20

3.2 Definição da Consulta 21

3.3 Estudo de Caso 22

3.3.1 Consultas *Select-range* 22

3.3.2 Consultas com *Agregação* 26

4 TRABALHOS RELACIONADOS 29

4.1 Introdução 29

4.2 Comparativo 29

4.3 Conclusão 31

5 PROPOSTA DE MONITORAMENTO E PROVISIONAMENTO PARA O PROCESSAMENTO DE CONSULTAS EM NUVEM 33

5.1 Preliminares 33

5.2 Provisionamento Inicial 33

5.3 Particionamento da Consulta 35

5.4 Monitoramento do Processamento da Consulta 36

5.5 Provisionamento Dinâmico 37

5.6 Conclusão 38

6 IMPLEMENTAÇÃO DO PROTÓTIPO E AVALIAÇÃO 39
6.1 Ambiente de Experimento 39
6.2 Metodologia 39
6.3 Protótipo 41
6.4 Consultas Utilizadas 41
6.5 Resultados Obtidos 42
6.5.1 Consultas *Select-range* 42
6.5.2 Consultas com *agregação* 44
6.6 Conclusão 48
7 CONCLUSÃO E TRABALHOS FUTUROS 50
7.1 Objetivos Alcançados 50
7.2 Trabalhos Futuros 51
REFERÊNCIAS BIBLIOGRÁFICAS 52

1 INTRODUÇÃO

Uma plataforma de Computação em Nuvem consiste tipicamente de uma grande quantidade de computadores responsáveis pelo processamento e armazenamento dos dados (ZHAO; HU; MENG, 2010). Tais recursos computacionais são combinados para servir vários consumidores usando um modelo multi-inquilino, com diferentes recursos físicos e virtuais que são alocados e realocados dinamicamente de acordo com a demanda dos usuários (MELL; GRANCE, 2009). Um ambiente de computação em nuvem é composto por um conjunto de máquinas virtuais (MVs) que podem ser alocadas em diferentes máquinas físicas (MFs); MVs diferentes podem apresentar desempenhos diferentes devido a muitos fatores, por exemplo a concorrência com outros processos nas MFs.

A elasticidade da computação em nuvem permite que o sistema adicione e remova recursos de acordo com as necessidades da aplicação em tempo de execução (ISLAM et al., 2012). Porém, elasticidade adequada sob-demanda não é uma tarefa trivial. Um ambiente de computação em nuvem está sujeito a vários fatores que podem influenciar seu desempenho, incluindo tipos diferentes de sistemas virtuais fornecidos pelo serviço, zonas de disponibilidade diferentes (regiões onde se encontram a infraestrutura de nuvem), e variação da demanda (SCHAD; DITTRICH; QUIANÉ-RUIZ, 2010).

Dessa forma, prover elasticidade requer monitorar continuamente (ou prever) a demanda do sistema por recursos, com objetivo de decidir quando adicioná-los e removê-los, ou seja, provisionar adequadamente. Esta é uma atividade crítica, dado que subestimar ou superestimar uma quantidade de recursos computacionais a serem provisionados pode afetar negativamente o provedor ou usuário da nuvem, comprometendo a qualidade de serviço acordada entre provedor e cliente (ao subestimar), ou o custo do serviço ser alto (ao superestimar). O provisionamento de MVs será visto mais adiante.

Na nuvem, o usuário do serviço tem algumas garantias, tais como desempenho e disponibilidade. Essas garantias de qualidade do serviço são definidas entre o provedor do serviço e o usuário e expressas por meio de um acordo de nível de serviço, ou em inglês *service-level agreement* (SLA). De forma geral, um SLA é composto pelos seguintes parâmetros (SOUSA et al., 2012):

- Receita: valor monetário pago pelo usuário ao provedor pelo tempo de computação;
- Custo operacional: valor monetário pago pelos recursos computacionais alocados para processar a carga de trabalho do usuário;
- Objetivo de nível de serviço, ou em inglês, *Service Level Objective* (SLO): que está associado a uma métrica definida pelo usuário que deve ser satisfeita pelo provedor, por exemplo tempo de resposta, vazão, disponibilidade, etc;
- Penalidade: valor monetário pago pelo provedor ao usuário por não satisfazer o SLO;

Neste contexto, é importante o provedor monitorar e adicionar/remover seus recursos computacionais (por exemplo, MVs) de acordo com a carga de trabalho, em tempo de

execução, a fim de minimizar o custo computacional e penalidades, satisfazendo o SLA de cada consulta.

Neste trabalho, visa-se monitorar continuamente o desempenho do SGBD e minimizar de forma automática as MVs utilizadas para o processamento das consultas, além de minimizar as possíveis violações de SLO. O tempo de processamento de uma consulta é utilizado como métrica neste trabalho.

Diferente das outras abordagens que são discutidas no capítulo 4, o método de provisionamento de recursos proposto neste trabalho não assume um número fixo de MVs (ROGERS; PAPAEMMANOUIL; CETINTEMEL, 2010), ou que as MVs possuem o mesmo desempenho (ALVES; BIZARRO; MARQUES, 2011), nem assume que pode-se prever a carga de trabalho previamente (SHARMA et al., 2011). Além disso, não apoia-se em regras definidas pelo usuário (Amazon Cloud Watch, 2012; Amazon Auto Scaling, 2012) para aumentar ou diminuir de forma previsível dinamicamente a quantidade de recursos computacionais alocados para uma carga de trabalho.

1.1 Motivação

Para motivar a solução proposta, considere que uma empresa precisa migrar suas aplicações OLAP para um ambiente com o objetivo de alocar os recursos computacionais sob-demanda (elasticidade rápida) e atender a uma qualidade de serviço, que é especificada sobre o tempo de resposta das consultas, como uma métrica de contrato (SLO). Dessa forma, não é favorável continuar a utilizar as soluções existentes de banco de dados distribuído. A empresa deseja que o processo de migração dos seus bancos de dados relacionais seja feita sem a necessidade de alteração de modelo, e consequentemente das aplicações existentes.

Utilizando a solução elástica deste trabalho, o processo de migração das aplicações e dos bancos de dados para a nuvem pode ser feita diretamente. O método usa o modelo de dados relacional e trabalha com replicação total de banco de dados. Dessa forma, cada MV implantada tem um SGBD com uma cópia completa do banco de dados. A solução não particiona os dados, mas a consulta dada de entrada. O particionamento virtual (LIMA; MATTOSO; VALDURIEZ, 2010) é aplicado para dividir a consulta em subconsultas a serem processadas nas máquinas virtuais alocadas. O uso de replicação total facilita o particionamento virtual, pois não é necessário tratar de problemas como reescrita da consulta ou transferência de dados entre as MVs durante o processamento da consulta, como ocorre com replicação parcial, por exemplo.

Para uma carga de trabalho (composta por consultas) dada de entrada, a estratégia deste trabalho aloca os recursos computacionais (máquinas virtuais) inicialmente capazes de atender o SLO da consulta. Isso é chamado de provisionamento inicial. Durante todo o processamento da consulta, a estratégia de monitoramento é executada e por meio dela é possível continuamente verificar se a qualidade de serviço especificada no SLO é satisfeita. Caso seja detectado que o SLO pode ser violado, é feito o provisionamento dinâmico das máquinas virtuais ou realocação dos recursos computacionais. A solução apresentada neste trabalho se baseia

na carga de trabalho, no SLO a ser obedecido e no desempenho das máquinas virtuais para aumentar ou diminuir a quantidade de MVs a serem dedicadas ao processamento das consultas.

1.2 Contribuições

As principais contribuições desse trabalho são as seguintes:

- Uma estratégia não-intrusiva, automática e adaptativa para o monitoramento do desempenho do SGBD durante o processamento das consultas em nuvem.
- Um método para minimizar o conjunto de MVs utilizadas de forma a satisfazer o SLO tanto para cada consulta individual como para as consultas de forma geral.
- Implementação de um protótipo e experimentação do algoritmo de monitoramento e provisionamento dinâmico em nuvem pública. O protótipo recebe como entrada as consultas a serem processadas e seu respectivo SLO.

Os resultados experimentais apresentados no capítulo 6, usando dados TPC-H (TPC-H, 2012) em uma infraestrutura da Amazon (Amazon EC2, 2013) mostram que a solução apresentada nesta dissertação é capaz de dinamicamente ajustar o número de MVs alocadas, de acordo com o SLO. Tal solução pode ser facilmente aplicado aos casos em que o usuário já tem suas aplicações usando bancos de dados relacionais e planeja migra-las automaticamente para uma infraestrutura em nuvem.

1.3 Publicações

Os seguintes artigos foram publicados durante o desenvolvimento deste trabalho. O primeiro artigo apresenta a ideia inicial do problema e uma proposta de monitoramento usando particionamento virtual. O segundo artigo destaca a solução de monitoramento aplicada para consultas *Select-range* (definidas mais adiante) com predicado usando atributo que é chave primária. Este também aborda uma estratégia de provisionamento dinâmico, a fim de minimizar as possíveis penalidades e o custo computacional. Por fim, o segundo artigo foi convidado para o Journal of Computer Science and Technology, sendo estendido para tratar de consultas com agregação e consultas *Select-range* cujo predicado pode ser inclusive sobre atributos não chave. Os trabalhos publicados são listados abaixo:

- COELHO DA SILVA, T. L., MACEDO, J. A. F., VIDAL, V. M. P. UMA ARQUITETURA PARA A EXECUÇÃO DE CONSULTAS CONSIDERANDO SLA EM AMBIENTES DE COMPUTAÇÃO EM NUVEM. Workshop de Teses e Dissertações no Simpósio Brasileiro de Banco de Dados, 2011.
- COELHO DA SILVA, T. L., NASCIMENTO, M. A., MACEDO, J. A. F., SOUSA, F. R. C., MACHADO, J. C. TOWARDS NON-INTRUSIVE ELASTIC QUERY PROCES-

SING IN THE CLOUD. The Fourth International Workshop on Cloud Data Management, 2012.

- COELHO DA SILVA, T. L., NASCIMENTO, M. A., MACEDO, J. A. F., SOUSA, F. R. C., MACHADO, J. C. Non-Intrusive Elastic Query Processing in the Cloud. Journal of Computer Science and Technology (JCST) special issue on cloud data management, 2012. (submetido)

1.4 Estrutura da dissertação

Esta dissertação está organizada da seguinte forma. No Capítulo 2, são apresentados conceitos básicos sobre Computação em Nuvem e Processamento de Consultas Distribuído. Em seguida, no Capítulo 3, o problema a ser atacado neste trabalho é caracterizado. No Capítulo 4, os trabalhos relacionados são destacados e discutidos. O Capítulo 5 apresenta a solução proposta para processamento de consultas em nuvem. Ela é composta por (i) uma estratégia de particionamento e monitoramento para o processamento de consultas e (ii) uma estratégia de provisionamento dinâmico e seus algoritmos. O Capítulo 6 apresenta a arquitetura e os resultados experimentais obtidos a partir da construção de um protótipo. Finalmente, o Capítulo 7 apresentada a conclusão deste trabalho, resumindo os resultados desta dissertação e apresentando os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Preliminares

Neste capítulo são apresentados os conceitos preliminares relacionados ao processamento de consultas distribuído na Seção 2.2 e sobre Computação em Nuvem na Seção 2.3.

2.2 Processamento de Consultas Distribuído

O processamento de consultas recebe até hoje atenção tanto nos casos em que o SGBD é centralizado quanto distribuído. No entanto, o processamento de consultas distribuído apresenta mais dificuldades, por ter um grande número de parâmetros que podem afetar o desempenho das consultas, como veremos a seguir.

Se nenhuma relação é fragmentada em um SGBD distribuído, a execução da consulta envolve apenas o custo de processamento local. Por outro lado, as relações envolvidas na consulta podem estar fragmentadas e/ou replicadas parcialmente, gerando alto custo de comunicação. O tempo de resposta da consulta também pode ser alto, caso tenha muitos sítios a acessar (ÖZSU; VALDURIEZ, 2011). No processamento de consultas distribuído, o objetivo é minimizar o custo de 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 de comunicação.

O processamento local de uma consulta distribuída é avaliado em termos do custo de E/S e CPU. O custo de comunicação diz respeito a troca de dados entre os sítios participantes. Este custo depende de vários fatores, tais como a quantidade de dados que está sendo transferida entre os sítios, a seleção do melhor sítio para executar a consulta, o número de mensagens de transferência entre os sítios participantes e a comunicação na rede. A localidade dos dados em sistemas distribuídos pode minimizar o volume e o custo de comunicação de dados na rede (ARNAUT; SCHROEDER; HARA, 2011).

Além de tentar otimizar o tempo de resposta da consulta distribuída, que compreende o tempo de inicialização e produção do resultado da consulta, outro objetivo importante do processamento de consultas distribuído é maximizar o paralelismo da execução de operações de uma dada consulta distribuída, a fim de que o tempo de resposta seja significativamente menor que o tempo total (Chhanda Ray, 2009).

Este trabalho não envolve fragmentação dos dados/replicação parcial. O escopo deste trabalho se restringe ao uso de replicação total, pois facilita o particionamento virtual da consulta. Não faz parte do escopo otimizar o tempo de resposta, reescrever a consulta nem encontrar o nível de máximo de paralelismo para o processamento da consulta. Também não é tratado do problema de transferência de dados entre as MVs durante o processamento da consulta, como ocorreria se outros tipos de fragmentação/replicação fossem usados. O objetivo deste trabalho é satisfazer o *SLO* acordado para cada consulta durante todo o seu processamento.

2.3 Computação em Nuvem

Existem diferentes definições e conceitos de computação em nuvem. Segundo (MELL; GRANCE, 2009), computação em nuvem pode ser definido como um modelo que permite acesso sob demanda a um agrupamento de recursos computacionais que podem ser configuráveis, como CPU, armazenamento, memória, entre outros. Eles podem ser rapidamente fornecidos e liberados com o mínimo esforço de gerenciamento ou assistência do provedor da nuvem.

Algumas propriedades fundamentais distinguem a computação em nuvem dos sistemas distribuídos tradicionais (e.g. sistemas em grade, *clusters*, P2P, etc) e estão relacionadas ao seu caráter atrativo: (i) Auto-Serviço sob demanda, (ii) Elasticidade rápida, (iii) Pagamento a medida que o serviço é utilizado (*Pay-as-you-go*) (iv) Nível de qualidade de serviço (SLA), (v) Agrupamento ou *Pooling* de Recursos. Consequentemente, qualquer serviço computacional que seja executado em um ambiente em nuvem precisará estar em conformidade com tais propriedades.

Auto-serviço sob demanda diz respeito a possibilidade do consumidor utilizar os recursos computacionais (tempo de servidor, por exemplo) quando necessário sem qualquer intervenção humana com o provedor do serviço. Outra característica é a elasticidade que permite adicionar mais recursos com o aumento da demanda e liberar recurso, na retração dessa demanda. Para o consumidor, os recursos disponíveis para provisionamento muitas vezes parecem ser ilimitados e podem ser alocados em qualquer quantidade e a qualquer momento. Os recursos em sistemas na nuvem são automaticamente controlados e o seu uso é medido em um nível de abstração apropriado para o tipo de serviço (como armazenamento e processamento, por exemplo). No modelo *Pay-as-you-go*, o consumidor só paga pelo que utiliza e pelo tempo de utilização. A utilização dos recursos pode ser monitorada, controlada e informada, gerando transparência tanto para o provedor como para o consumidor do serviço utilizado.

O SLA (*service-level agreement*) é a garantia mínima de qualidade de serviço (QoS). Os recursos de computação do provedor são agrupados para atender a múltiplos consumidores em modelo multi-inquilinos. Os diferentes recursos físicos e virtuais são dinamicamente alocados e desalocados conforme a demanda dos consumidores. Há uma certa independência de localização geográfica, uma vez que o consumidor em geral não controla ou conhece a localização exata dos recursos fornecidos (como armazenamento, processamento, memória e comunicação de rede), mas pode ser capaz de especificar a localização em um nível de abstração mais alto (como país, estado ou *datacenter*).

O provisionamento de serviços em nuvem deve estar em conformidade com um conjunto de propriedades não funcionais pré-definidos e especificadas como uma qualidade de serviço. Tais propriedades são negociadas por meio de contratos de nível de serviço ou SLAs. Os SLAs são utilizados para indicar os ganhos e as penalidades. O provedor lucra se disponibiliza o serviço em certo nível, porém é penalizado se a qualidade acordada com seu cliente via SLA não foi atingida (CHI et al., 2011). Os provedores de serviços em Computação em Nuvem devem tentar otimizar seus lucros enquanto atendem as requisições de vários clientes. Dessa

forma o SLA forma a base de acordo entre os provedores de nuvem e consumidores.

Como a maioria das aplicações em nuvem são orientadas a dados, os Sistemas Gerenciadores de Banco de Dados (SGBDs) são componentes críticos na pilha de software (ELMORE et al., 2011). O modelo Chave-Valor apresenta armazenamento distribuído e em grande escala, como por exemplo os sistemas Dynamo (DECANDIA et al., 2007), Bigtable (CHANG et al., 2006), PNUTS (COOPER et al., 2008) e Cassandra (LAKSHMAN; MALIK, 2009). Eles são projetados para atender grande número de solicitações simultâneas usando uma infraestrutura com milhares de servidores, de maneira elástica e tolerante a falhas. Embora o modelo seja extremamente bem-sucedido, ele desconsidera vários aspectos fundamentais presentes nos SGBDs relacionais. Isso pode ser considerado como um retrocesso, pois dificulta a presença de fatores importantes, tais como a independência de dados, transações confiáveis e muitas outras características exigidas por aplicações que são fundamentais para a indústria de banco de dados (ARNAUT; SCHROEDER; HARA, 2011). Além disso, não é vantajoso para uma aplicação cujo os requisitos de armazenamento são menores (dezenas de MBs para alguns GB) utilizar o modelo chave valor.

Este trabalho pode ser usado para os casos em que o usuário já tem suas aplicações usando bancos de dados relacionais e planeja implantá-las diretamente em uma infraestrutura de nuvem. Esse trabalho foca no gerenciamento de dados como um serviço, mais especificamente no processamento da consulta e no monitoramento do SLA durante o processamento da mesma. Atualmente, as infra-estruturas de monitoramento carecem de soluções apropriadas para o monitoramento do SLA. Um dos desafios é determinar intervalos apropriados de monitoramento no nível de aplicação, tal que seja possível detectar quando um SLA é violado (EMEAKAROHA, 2012).

Ao monitorar os recursos da infra-estrutura de nuvem, o provedor ganha informação sobre o uso dos recursos e a disponibilidade dos mesmos. É importante considerar a taxa com que essa informação é adquirida. Isso pode comprometer tanto o desempenho geral do sistema quanto o lucro do provedor. Alta sobrecarga pode eventualmente degradar o desempenho do sistema. Por outro lado, o monitoramento realizado a uma baixa taxa faz com que a falta de informações não detecte que o SLA será violado, resultando no pagamento de penalidades pelo provedor. Portanto, para resolver esse problema, são necessárias técnicas que determinem intervalos de medição ideais para monitorar de forma eficiente e que detecte violações no SLA. (EMEAKAROHA, 2012).

Este trabalho apresenta uma técnica de monitoramento não intrusiva que não se baseia em intervalos fixos de tempo. O monitoramento é feito de acordo com o particionamento da consulta, que é adaptativo, durante o processamento da mesma. A consulta é particionada de acordo com o valor do SLO. Durante o monitoramento, caso seja detectado que o SLA é violado, a decisão de aumentar o número de máquinas virtuais para auxiliar no processamento da consulta e evitar penalidades é tomada. Ou ainda, se durante o monitoramento for detectado que uma máquina virtual pode finalizar o processamento de uma consulta antes do acordado no SLA, a MV possui um "tempo de folga". Esse tempo de folga pode ser dedicado para as novas consultas que chegam ao sistema ou para "ajudar"as outras MVs no processamento de consultas que ainda estão sendo executadas no sistema. Todos esses casos serão contemplados

no Capítulo 5.

2.4 Conclusão

Neste capítulo, são apresentados os conceitos que são essenciais para o desenvolvimento de uma solução elástica para o processamento de consultas. A seguir, o problema abordado nesta dissertação é detalhado.

3 FORMULAÇÃO DO PROBLEMA

O foco deste trabalho é o problema de processar consultas em ambientes de nuvem de maneira elástica e que satisfaça ao SLA. Com o objetivo de melhorar a compreensão, a Seção 3.1 especifica o modelo de custo deste trabalho e descreve o problema. Na Seção 3.2 são apresentados os tipos de consultas considerados nesse trabalho. Um exemplo de motivação da solução é apresentado na Seção 3.3.

3.1 Modelo de Custo

Considere TT_Q como sendo uma estimativa de tempo total necessário para processar uma consulta Q. Seja SLO_Q o tempo acordado para processar uma dada consulta Q de acordo com o SLA.

Se p é o custo, por unidade de tempo, de não satisfazer ao SLO_Q, pode-se definir a *penalidade do provedor* (custo) como:

$$pp_Q = max\{(TT_Q - SLO_Q) * p, 0\} \tag{3.1}$$

Seja c_{vm} o custo, por unidade de tempo, de usar uma MV que contém uma réplica total de um banco de dados relacional. Seja n_Qt o número de MVs alocadas para executar Q no tempo t, onde o tempo é discretizado em unidades de cobrança (por exemplo, horas no caso da Amazon (Amazon EC2, 2013)). Assim, define-se o *custo computacional de executar Q* como:

$$cc_Q = \sum_{t=1}^{TT_Q} (n_Q(t) * c_{vm}). \tag{3.2}$$

Neste contexto, alocar uma grande quantidade de MVs minimiza as penalidades do provedor, mas aumenta o seu custo computacional. Por outro lado, alocar uma quantidade menor de MVs reduz os custos computacionais, mas pode implicar em aumento nos custos de penalidades.

> Dessa forma, dada uma consulta Q, é necessário obter, para cada instante de tempo t (discretizado), o número de MVs (n_Qt) que minimiza o custo de processar Q:
>
> $$(pp_Q + cc_Q) \tag{3.3}$$

Esta definição de problema captura a elasticidade do ambiente de nuvem, pois ao longo da execução das consultas, uma quantidade diferente de MVs pode ser suficiente para processar uma consulta respeitando o seu SLO. Como argumentado anteriormente, essa quantidade pode variar devido, por exemplo, ao aumento (ou diminuição) da carga de trabalho nas máquinas físicas em que se encontram as MVs ou ainda, devido à quantidade de consultas submetidas ao sistema.

Similar a (MIAN; MARTIN; VAZQUEZ-POLETTI, 2012), este trabalho utiliza um modelo de custo simplificado, pois considera apenas os custos associados às MVs, que, em geral, são os mais significativos. Este modelo não inclui os custos referentes à implantação, armazenamento dos dados persistentes e das imagens das MVS. Além disso, assume-se que todas as MVs possuem o mesmo custo, independente do desempenho distinto durante a sua execução. Como as operações sobre os dados consideradas nesse trabalho são somente de leitura, não há custos associados com atualizações nas réplicas de dados, e não é considerado o custo associado com o uso do SGBD.

3.2 Definição da Consulta

Esse trabalho foca em consultas referentes a uma única tabela ou chamadas de *single table*: *Select-range* e consultas com *agregação*. As consultas *Select-range* podem ser definidas da seguinte forma:

```
SELECT *
FROM tabela T
WHERE T.atr >= Vs and T.atr < Vf;
```

tal que `T.atr` é um atributo da tabela `T`. Além disso, `Vs` e `Vf` são valores inteiros.

Esse tipo de consulta é genérica o suficiente para acomodar consultas *single table* específicas discutidas no microbenchmark proposto em (POPESCU et al., 2010). Por exemplo, para consultas *scan*:

```
SELECT *
FROM tabela T
```

pode ser trivialmente reescrita como as duas consultas seguintes:

```
SELECT MIN(T.pk) into Vs, MAX(T.pk) into Vf
FROM tabela T;
```

```
SELECT *
FROM tabela T
WHERE T.pk >= Vs and T.pk < Vf + 1;
```

tal que `T.pk` é um atributo que é chave primária da tabela `T`.

Este trabalho lida também com consultas com *agregação* com ou sem predicado na cláusula WHERE:

```
SELECT OPER(T.atr)
FROM tabela T;
```

tal que `OPER` é um operador de agregação (SUM, por exemplo) e `T.atr` é um atributo da tabela `T`.

A seguir, é apresentado um exemplo do método proposto, que é detalhado nos pró-

ximos capítulos. A Tabela 3.1 apresenta a notação utilizada nesse trabalho.

Tabela 3.1: Notação

Notação	Significado
Q	Uma consulta do tipo *select-range* ou com *agregação*
Q_s	Seletividade de Q (número de tuplas a serem recuperadas)
SLO_Q	Máximo tempo (segs) permitido para processar Q
mv_i	MV i, $i \in \{1,2,...,m\}$
TL_i	taxa de leitura de mv_i (tuplas/seg)
NT_i^Q	$TL_i \times SLO_Q$, isto é, o número de tuplas que mv_i pode ler sem violar o SLO de Q
TF_i	Tempo de folga que mv_i pode alocar para outras consultas sem violar qualquer SLO
p	Custo, por unidade de tempo, por não satisfazer o SLO_Q
c_{mv}	Custo, por unidade de tempo, de usar uma MV
H	Tabela de dados contendo uma 4-tupla ⟨atributo de particionamento, seletividade da consulta, número de partições, tempo de processamento médio⟩
TM_i	Tempo gasto para monitorar o desempenho de mv_i
S	Um conjunto ordenado de pares do tipo $\langle mv_i, TF_i \rangle$ tal que TF_i>0 e existe apenas um elemento para cada mv_i
V	Um conjunto de mv_i alocado para processar uma consulta Q
$T_{restante}$	Tempo que resta para finalizar o processamento de Q sem violar SLO_Q

3.3 Estudo de Caso

3.3.1 Consultas *Select-range*

Assuma que a seguinte consulta *select-range* Q com SLO_Q é recebida por um provedor da nuvem.

```
SELECT * // <-- Q
FROM tabela T
WHERE T.pk >= 0 and T.pk < 3000;
```

tal que T.pk é a chave primária da tabela T.

Assumindo que os valores da chave primária de T sejam sequenciais, sem apresentar lacunas entre os valores, então $Q_s = 3000$ tuplas. Este trabalho considera que as tabelas estão clusterizadas pela chave primária. Assuma que SLO_Q é 100 segs e que o provisionamento inicial é uma única máquina virtual mv_1 tal que TL_1 = 20 tuplas/seg e consequentemente NT_1^Q = 2000. Claramente, usando somente mv_1 trará uma penalidade a ser paga pelo provedor, considerando o custo de ler todas as 3000 tuplas: $(TT_Q - SLO_Q) \times p = (150 - 100) \times p = 50 \times p$. Nesse caso, é necessário alocar uma outra MV (mv_2) para ajudar. Assuma que $TL_2 = 10$ e $NT_2^Q = 1000$.

Neste momento, é possível que as duas MVs sejam suficientes para processar Q e satisfazer o SLO. Reescreve-se Q em duas subconsultas seguintes: Q_1 e Q_2, a primeira é executada em mv_1 e a segunda em mv_2, respectivamente. Note que é usado o particionamento virtual (isto é, particiona-se o predicado que está sobre a chave primária) para dividir Q em Q_1 e Q_2.

```
SELECT * // <-- Q1
FROM tabela T
WHERE T.pk >= 0 and T.pk < 2000;

SELECT * // <-- Q2
FROM tabela T
WHERE T.pk >= 2000 and T.pk < 3000;
```

Utilizando somente as duas MVs durante todo o processamento não garante que a qualidade definida no SLO será satisfeita, pois o ambiente em nuvem é instável e o desempenho das MVs pode mudar durante a execução das consultas. Este fato pode ocorrer, por exemplo devido ao compartilhamento de recursos entre as máquinas físicas que hospedam as MVs ou pela adição de mais consultas a uma MV. Isso pode implicar em degradação do desempenho e, consequentemente, violação do SLO_Q. Assim sendo, é necessário utilizar uma abordagem proativa baseada em técnicas de monitoramento contínuo do desempenho.

A proposta para resolver este problema é particionar as consultas, de tal sorte que a taxa de leitura de cada MV possa ser monitorada em uma frequência que permita outras MVs serem adicionadas quando necessário, a fim de fazer cumprir o SLO_Q. Uma questão importante é a frequência do monitoramento. Se ele é muito frequente, as consultas originais teriam que ser particionadas em muitas subconsultas. Dessa forma, a sobrecarga adicionada pode prejudicar mais do que ajudar. Se o monitoramento é feito com pouca frequência, pode ser difícil fazer correções em tempo hábil e evitar possíveis penalidades.

O processo de particionamento usa um histórico de dados que contém informações sobre quanto tempo é necessário para se processar uma consulta *select-range* de determinada seletividade e com um certo número de partições. Assume-se a existência de uma tabela H para cada MV, tal que cada entrada é uma 4-tupla:$\langle$atributo de particionamento, seletividade da consulta, número de partições, tempo médio de processamento$\rangle$. Usando H, é possível encontrar o número máximo de partições para dividir a consulta, permitindo dessa forma, que o monitoramento seja tão frequente quanto possível, enquanto continua-se a satisfazer o SLO_Q.

Assuma que H tem informação suficiente para possuir as seguintes entradas quando a seletividade da consulta é 2000 e o atributo de particionamento é `pk`:
$H = \{\langle pk, 2000, 2, 72\rangle, \langle pk, 2000, 3, 80\rangle, \langle pk, 2000, 4, 95\rangle, \langle pk, 2000, 5, 120\rangle, \langle pk, 3000, 2, 81\rangle,$
$\langle pk, 3000, 3, 96\rangle, ...\}$. O aumento do tempo médio por ter mais partições refere-se, por exemplo ao tempo de mudança de contexto da execução de uma partição para outra, custo da reescrita da partição.

Nesse caso, o maior número de partições que pode-se usar é 4, maximizando a

chance de monitorar o desempenho do processamento de Q, enquanto o SLO_Q continua a ser satisfeito. Então, divide-se a consulta Q_1 em 4 consultas:

```
SELECT * // <-- Q1,1
FROM tabela T
WHERE T.pk >= 0 and T.pk < 500;

SELECT * // <-- Q1,2
FROM tabela T
WHERE T.pk >= 500 and T.pk < 1000;

SELECT * // <-- Q1,3
FROM tabela T
WHERE T.pk >= 1000 and T.pk < 1500;

SELECT * // <-- Q1,4
FROM tabela T
WHERE T.pk >= 1500 and T.pk < 2000;
```

O mesmo raciocínio pode ser aplicado no processamento de Q_2 em mv_2. Essa metodologia de particionamento usando a chave primária como atributo de particionamento é a mesma para qualquer consulta *select-range* e consultas com *agregação*, conforme apresentado na próxima seção. Assuma que o desempenho de vm_2 continue estável e que seja capaz de terminar a sua carga de trabalho dentro do planejado. A Figura 3.1 ilustra o processo de particionamento de Q.

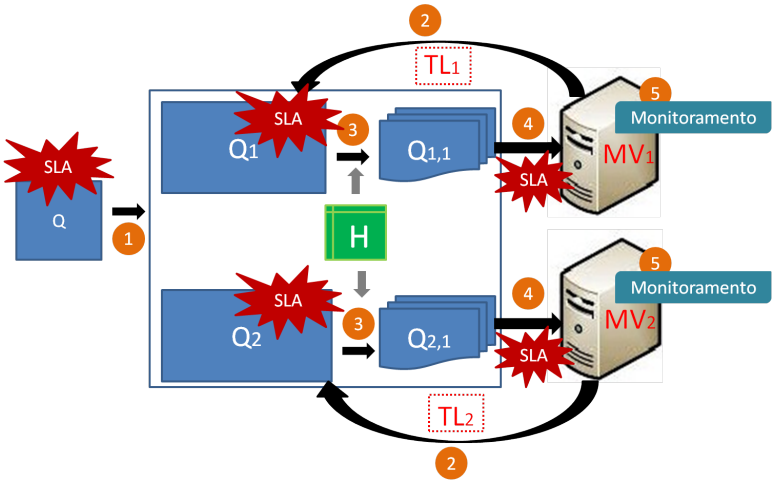


Figura 3.1: Processo de particionamento de Q.

Quando $Q_{1,1}$ termina, tem-se a primeira oportunidade para monitorar o desempenho da MV de maneira não intrusiva. Assuma que foram gastos 50 segundos para executar $Q_{1,1}$. Isto significa que o valor da taxa de leitura da MV é de $TL_1 = 10$ (500 tuplas em 50 segundos), o que leva a um tempo de conclusão do processamento das 3 subconsultas restantes de 150 segundos. Porém, esse valor está acima do valor do SLO_Q, e desencadeia uma revisão do provisionamento

inicial, a fim de que o SLA_Q possa ser satisfeito. Antes de revisar o provisionamento inicial, as 3 partições remanescentes ($Q_{1,2}$, $Q_{1,3}$ e $Q_{1,4}$) são reunidas em uma única consulta.

Dada a taxa de leitura corrente para mv_1, o melhor que se pode esperar é que a MV seja capaz de ler nos 50 segundos restantes antes que SLO_Q seja violado, apenas 500 tuplas das 1500 tuplas restantes. A única chance para satisfazer o SLO_Q é realocar parte do processamento de consulta para outra MV (recém-alocada). Suponha que a nova MV (mv_3) é tal que TL_3=50. Dessa forma, todas as 1000 tuplas restantes podem ser lidas por mv_3 em 20 segundos no melhor caso, isso não leva a uma violação do SLO_Q (relembre que nesse ponto, já foram gastos 50 segundos processando $Q_{1,1}$). Note que o particionamento é adaptativo, pois a consulta é novamente particionada caso o sistema aloque um novo número de máquinas.

Em síntese, a abordagem elástica deste trabalho é capaz de processar Q da seguinte forma: nos instantes de tempo 0 a 100 segundos, mv_1 é responsável por recuperar as tuplas que satisfazem o intervalo de chave primária [0, 1000]; mv_2 é responsável por recuperar as tuplas que satisfazem o intervalo [2000, 3000]. Esta MV é utilizada de 0 a 100 segundos; A mv_3 é utilizada de 50 a 70 segundos para recuperar as tuplas dentro do intervalo de chave primária igual a [1000, 2000]. O número de MVs usadas em função do tempo para executar Q são: 2 MVs durante [0-50], 3 MVs durante [50-70], e ainda 2 MVs durante [70-100] como apresentado na Figura 3.2. No instante de 100 segundos, as MVs são desalocadas.

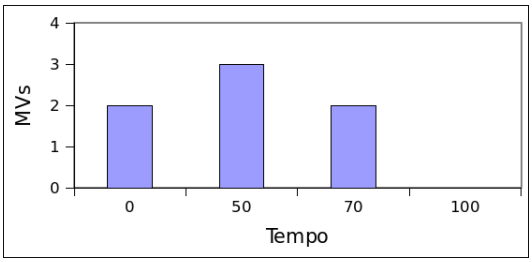


Figura 3.2: Variação no número de MVs alocadas pela nossa estratégia.

Considere um novo cenário, agora com o aumento do desempenho da MV. Por exemplo, suponha que uma consulta Q_2 foi particionada em, por exemplo 3 subconsultas. Imagine que depois que a primeira subconsulta termina e o monitoramento inicia, descobre-se que TL_2 melhorou, possivelmente por causa de alguns outros processos na MV que terminaram, e então mv_2 pode terminar o processamento de Q_2 antes do planejado. Dessa forma, esse tempo de folga pode ser usado no processamento de algumas consultas que a mv_1 não pode executar. Essas consultas podem ser alocadas para mv_2 visando satisfazer ao SLO_Q. Apesar de ser um exemplo simples e com algumas suposições, a solução dada por este trabalho trata dos cenários de diminuição e aumento de desempenho das MVs. Os próximos capítulos detalham e discutem os problemas destacados nesta seção.

No exemplo acima, o predicado da consulta *select-range* é sobre a chave primária da tabela. Claramente, nem todas as consultas utilizam chave primária no predicado. Isto é, é necessário tratar consultas tais como:

```
SELECT *
FROM tabela T
```

`WHERE T.atr >= Vs and T.atr < Vf;`
tal que `T.atr` denota um atributo da tabela `T`. Além disso, `Vs` e `Vf` são valores inteiros.

Uma opção inicial é tentar o mesmo tipo de particionamento como se fez para a chave primária. Porém, ao contrário do caso anterior, em que um índice pode ser trivialmente usado na execução de cada partição, o plano de consulta de cada partição acusa uma varredura linear sobre a tabela. A criação de um índice para cada possível atributo presente no predicado da consulta não é viável. Assim, a solução proposta é reescrever a consulta, a fim de usar a chave primária para o particionamento, como apresentado a seguir:

```
SELECT MIN(T.pk) into V_min, MAX(T.pk) into V_max
FROM tabela T;
SELECT *
FROM tabela T
WHERE T.atr >= Vs and T.atr < Vf and T.pk >= V_min and T.pk < V_max + 1;
```

Nesse caso, para o calculo de Q_s usa-se o atributo de chave primária. Reescrevendo a consulta, pode-se ter uma super estimativa, porque a seletividade do predicado sobre `T.pk` é maior ou igual a seletividade do predicado sobre `T.atr`. Com isso, pode alocar MVs além do necessário. No entanto, essa estratégia é mais segura, já que pode-se realizar o particionamento e evitar penalidades. Mesmo que o predicado sobre `T.atr` seja menor que o predicado sobre `T.pk`, é importante particionar pela chave primária `T.pk`, porque cada partição não executará uma varredura completa na tabela `T` como ocorre se o particionamento for pelo `T.atr` (considerando que a partição recupera menos de 20% das tuplas da tabela, e consequentemente pode usar o índice da chave primária para recuperar os dados). Depois que a consulta é reescrita, usa-se a metodologia de particionamento apresentada anteriormente. O monitoramento e provisionamento de máquinas virtuais para processar a consulta reescrita também é feito da mesma forma como apresentado anteriormente.

É importante notar que todo o monitoramento e ajuste são feitos de *maneira não-intrusiva*, isto é, as MVs e seus respectivos SGBDs são fornecidos pelo provedor da nuvem e não requerem qualquer alteração para serem utilizados.

3.3.2 Consultas com *Agregação*

A seguir é discutido como tratar consultas com *agregação* usando a estratégia de particionamento discutida acima. Assuma que a seguinte consulta Q com *agregação*, possui SLO_Q igual a 100 segundos, e é recebida pelo provedor da nuvem:

```
SELECT OPER(T.atr) // <-- Q
FROM tabela T;
```
A diferença entre esse exemplo e o anterior é como o particionamento deve ser feito para consultas com *agregação*. Para particionar Q, adiciona-se um predicado sobre a chave primária de T que é usada como um atributo de particionamento. O intervalo de valores nesse caso está entre o valor mínimo e máximo do atributo que é chave primária. Para simplificar, considere que o maior valor para `T.pk` é 2999 e o mínimo é zero.

```
SELECT OPER(T.atr) // <-- Q
FROM tabela T
WHERE T.pk >= 0 and T.pk < 3000;
```

Assuma que a seletividade da consulta nesse caso é a seletividade do predicado que está sobre *T.pk* (Q_s=3000 tuplas). Se o operador de agregação é uma função distributiva (PAPADIAS et al., 2001) como COUNT, por exemplo, o resultado de Q é o somatório dos resultados coletados para cada partição. Porém, se o operador de agregação é uma função algébrica, como por exemplo AVG, o resultado da consulta original não pode ser facilmente obtido por meio das partições. Com isso, transforma-se a função algébrica em funções distributivas, no caso do AVG, é possível transformar em SUM e COUNT, a fim de que o resultado da consulta original possa ser facilmente obtido por meio da soma dos resultados das partições.

Considere o mesmo cenário apresentado no exemplo anterior, a mv_1 possui taxa de leitura igual $TL_1 = 20$, e consequentemente $NT_1^Q = 2000$. Usar somente mv_1 trará uma penalidade ($(TR_Q - SLO_Q) \times p = (150 - 100) \times p = 50 \times p$) a ser paga pelo provedor. É necessário alocar uma nova máquina virtual (mv_2) para ajudar. Assuma que $TL_2 = 10$ e $NT_2^Q = 1000$. Reescreva Q nas duas seguintes subconsultas, Q_1 e Q_2, execute a primeira na mv_1 e a segunda na mv_2, respectivamente.

```
// Q1:
SELECT OPER(T.attr)
FROM tabela T
WHERE T.pk >= 0 and T.pk < 2000;
```

```
// Q2:
SELECT OPER(T.attr)
FROM tabela T
WHERE T.pk >= 2000 and T.pk < 3000;
```

O mesmo histórico de dados H pode ser utilizado nesse exemplo, já que o atributo de particionamento `T.pk` é o mesmo. O maior número de partições nesse caso é 4, pois maximiza a chance de monitorar o processamento de Q, enquanto continua-se a satisfazer ao SLO_Q. Então, a consulta Q_1 é dividida em 4 consultas:

```
SELECT OPER(T.attr) // <-- Q1,1
FROM tabela T
WHERE T.pk >= 0 and T.pk < 500;
```

```
SELECT OPER(T.attr) // <-- Q1,2
FROM tabela T
WHERE T.pk >= 500 and T.pk < 1000;
```

```
SELECT OPER(T.attr) // <-- Q1,3
FROM tabela T
WHERE T.pk >= 1000 and T.pk < 1500;
```

```
SELECT OPER(T.attr) // <-- Q1,4
FROM tabela T
WHERE T.pk >= 1500 and T.pk < 2000;
```

Um raciocínio similar poderia ser aplicado com respeito a Q_2 a ser executada em mv_2. Porém, assuma que o desempenho de vm_2 é estável, e dessa forma é capaz de terminar a carga de trabalho de acordo com o que foi planejado.

Depois que $Q_{1,1}$ terminou, tem-se a primeira oportunidade de monitorar o desempenho da MV de maneira não intrusiva. Para simplificar o entendimento, suponha que ocorre a mesma situação do exemplo anterior, tal que o tempo gasto para executar $Q_{1,1}$ foi de 50 segundos. Dessa forma, a taxa de leitura da MV diminui para $RR_1 = 10$ e o tempo esperado para terminar todas as 3 partições restantes ($Q_{1,2}$, $Q_{1,3}$ e $Q_{1,4}$) é de 150 segundos.

Isso ultrapassa o valor do SLO_Q. A fim de que o SL_Q ainda seja satisfeito, é possível seguir os mesmos passos apresentados no exemplo anterior para revisar o provisionamento inicial, e dessa forma realizar o provisionamento dinâmico necessário. Dada a taxa de leitura da mv_1, o melhor que se pode esperar é que mv_1 seja capaz de recuperar somente 500 tuplas das 1500 restantes em 50 segundos, antes que o SLO_Q seja violado.

Considere que uma nova MV foi alocada (mv_3) com $TL_3 = 50$ para satisfazer o SLO_Q e redistribuir o processamento de algumas consultas. Dessa forma, no melhor caso todas as 1000 tuplas restantes que não podem ser recuperadas por mv_1, podem ser lidas por mv_3 em 20 segundos. Isso não acarretaria violação do SLO_Q (relembre que os 50 segundos já foram gastos com $Q_{1,1}$). Observe o comportamento elástico semelhante ao do exemplo anterior (2 MVs durante os instantes [0, 50], 3 MVs durante o tempo [50-70] e novamente 2 MVs durante o tempo [70-100]) apresentado na Figura 3.2.

4 TRABALHOS RELACIONADOS

4.1 Introdução

O gerenciamento de dados em ambiente de nuvem têm sido bastante estudado pela comunidade de banco de dados. Os trabalhos a seguir apresentam uma estratégia de provisionamento de recursos computacionais para o processamento de cargas de trabalho (ou consultas) sobre dados que estão armazenados e devem ser processados em uma infraestrutura de nuvem. Alguns desses trabalhos consideram uma qualidade de serviço ou SLA a ser atendido e propõem ainda uma estratégia de monitoramento, como é o caso deste trabalho.

Na Seção 4.2, os principais trabalhos relacionados são discutidos e um estudo comparativo é apresentado descrevendo o que difere tais trabalhos deste. Finalmente, a Seção 4.3 concluí esse capítulo.

4.2 Comparativo

Alves et al. (ALVES; BIZARRO; MARQUES, 2011) propõem FloodDQ, um sistema baseado em MapReduce que adaptativamente aumenta e diminui o poder computacional em tempo real, a fim de finalizar a execução dentro de um prazo especificado. Esse trabalho restringe o seu escopo para consultas *single pipeline*. O cálculo do número de nós a serem adicionados ou removidos é baseado na taxa de processamento dos dados e o trabalho assume que todos os nós tem a mesma capacidade de processamento. Essa estratégia é similar a apresentada neste trabalho, porém esse trabalho utiliza intervalos regulares de monitoramento, ao passo que nossa estratégia de monitoramento é adaptativa e considera que as MVs podem ter desempenho diferente, além de apresentar um algoritmo de provisionamento inicial.

Um método adaptativo de provisionamento de MVs para um Sistema de Processamento de Stream Distribuido (SPSD) em nuvem é apresentado em (CERVINO et al., 2012). O algoritmo proposto de provisionamento usa o método de caixa-preta, isto é, ele é independente de consultas específicas a serem executadas no SPSD. Este algoritmo calcula o número de MVs a serem alocadas, baseado unicamente na taxa de streams dadas como entrada. O algoritmo é chamado periodicamente e calcula o número de MVs que são necessárias para processar a demanda; contudo, o artigo não especifica em que frequência esse algoritmo é invocado. Diferentemente, nosso trabalho não foca apenas em provisionar recursos, mas em prover um algoritmo de monitoramento adaptativo, e o re-provisionamento (se necessário) durante o processamento da consulta.

Os autores de (SHARMA et al., 2011) propõem Kingfisher, um sistema que tem como objetivo minimizar o custo do cliente, isto é, o custo de alocar servidores para satisfazer o SLA das aplicações. Kingfisher utiliza programação linear inteira para o cálculo do custo da infra-estrutura e do custo de transição, resultantes das decisões de elasticidade tomadas devido a mudanças na carga de trabalho. Kingfisher usa um método proativo para saber quando provisionar e assume um preditor de carga de trabalho ideal que usa estatísticas recuperadas de um

sistema de monitoramento. Esse preditor é capaz de obter estimativas de cargas de trabalho futuras. Kingfisher foca no custo do cliente e não especifica qual a frequência de monitoramento. A abordagem deste trabalho tenta minimizar o custo do provedor, o monitoramento é adaptativo e não assume um preditor de carga.

Em (ROGERS; PAPAEMMANOUIL; CETINTEMEL, 2010), os autores apresentam um framework para provisionamento que identifica um conjunto de recursos de custo mínimo da infraestrutura, a fim de garantir que a qualidade de serviço seja satisfeita. O artigo descreve duas soluções para o problema de provisionamento de recursos: solução de caixa preta e de caixa branca. O provisionamento de caixa-preta considera perfis do desempenho e do custo de diferentes tipos de MVs sob a variação de taxas de consulta que são dadas de entrada (utiliza amostras de execução). O objetivo é capturar a taxa de entrada que cada MV suporta sem violar a qualidade de serviço que está associada às consultas executadas pelo sistema. O provisionamento por caixa-branca estima quanto de recurso computacional é necessário para a execução de carga de trabalho, usando estatísticas do otimizador de banco de dados. Este artigo mapeia os componentes da nuvem e suas as restrições em um problema da mochila multi-compartimentada, e assim resolvido por meio de heurísticas. Como as MVs de uma nuvem pública podem ter desempenho variável e as soluções que ambos os métodos propõem podem não ser ótima, o monitoramento durante a execução da carga é necessário e possivelmente, a tomada de decisões para que a qualidade de serviço seja garantida. Diferente deste trabalho, o artigo (ROGERS; PAPAEMMANOUIL; CETINTEMEL, 2010) não apresenta uma estratégia de monitoramento. Além disso, a solução desse artigo é intrusiva.

O problema de provisionar recursos para executar cargas de trabalhos analíticas em nuvem pública é examinado em (MIAN; MARTIN; VAZQUEZ-POLETTI, 2012). O algoritmo apresentado explora o espaço de possíveis configurações (um conjunto de diferentes tipos de MVs e o mapeamento das classes de consulta para MVs) para a carga de trabalho dada de entrada, baseado em predição dos custos das configurações. O algoritmo tenta encontrar uma configuração, minimizando o custo dos recursos e satisfazendo o SLA associado a carga de trabalho. O modelo de custo apresentado em (MIAN; MARTIN; VAZQUEZ-POLETTI, 2012) é similar ao deste trabalho. Porém, considerando que o desempenho de uma configuração em particular pode piorar ao longo do tempo e o SLA ser violado, é necessário modificar a alocação dos recursos automaticamente para a aplicação. O artigo (MIAN; MARTIN; VAZQUEZ-POLETTI, 2012) não apresenta uma estratégia de provisionamento dinâmico.

O sistema Kairos é apresentado em (CURINO et al., 2011). Este sistema prove um esquema de consolidação para servidores de banco de dados que estão sendo provisionados além do necessário. Kairos usa técnicas de monitoramento e modelos de recursos para medir os requisitos de hardware de cada carga de trabalho do banco de dados, e ainda prevê a utilização combinada dos recursos para essas cargas de trabalho. O trabalho visa minimizar o número de servidores e aumenta o balanceamento da carga, enquanto satisfaz o SLA. Similar ao objetivo deste trabalho, Kairos tenta reduzir o custo operacional, porém difere deste trabalho, pois não realiza provisionamento dinâmico que permite o aumento e a diminuição dos recursos alocados ao longo do tempo.

Finalmente, o artigo (VIGFUSSON et al., 2009) apresenta um método adaptativo

para otimizar o tempo de resposta das consultas (range queries) em banco de dados distribuídos. O algoritmo particiona e adaptativamente identifica o melhor nível de paralelismo para cada consulta. O maior nível de paralelismo escolhido não é necessariamente a melhor estratégia para otimizar o desempenho de execução da consulta. Se uma consulta é enviada para muitos MVs de armazenamento, os resultados podem ser produzidos rapidamente, porém consumidos lentamente, ou seja, o cliente fica saturado. Semelhante a este trabalho, (VIGFUSSON et al., 2009) propõe um algoritmo de provisionamento adaptativo para consultas *select-range* (ou *range queries*) e consideram variações no desempenho das MVs. Por outro lado, este trabalho não observa um SLA e não especifica qual a frequência do algoritmo de monitoramento.

A Tabela 4.1 mostra um resumo comparativo entre os trabalhos relacionados.

Tabela 4.1: Análise Comparativa entre os trabalhos relacionados.

Trabalho	**SLA**	**Elástico**	**Não intrusivo**	**Desempenho heterogêneo das MVs**	**Monitoramento adaptativo**	**Preditor**
(VIGFUSSON et al., 2009)	Não	Sim	Sim	Sim	Não	Não
(ROGERS; PAPAEMMANOUIL; CETINTEMEL, 2010)	Sim	Não	Não	Sim	N. A.	Não
(SHARMA et al., 2011)	Sim	Sim	Sim	Não	Intervalo não especificado	Sim
(CURINO et al., 2011)	Sim	Não	Sim	Sim	Sim	Sim
(ALVES; BIZARRO; MARQUES, 2011)	Sim	Sim	Sim	Não	Não	Não
(CERVINO et al., 2012)	Sim	Sim	Sim	Não	Intervalo não especificado	Não
(MIAN; MARTIN; VAZQUEZ-POLETTI, 2012)	Sim	Não	Sim	Sim	N. A.	Sim

4.3 Conclusão

Este capítulo apresentou os principais trabalhos relacionados com o tema desta dissertação. Apesar das várias estratégias para gerenciamento de dados em ambiente de nuvem que propõem o provisionamento de recursos computacionais e visam satisfazer ao SLO, tais trabalhos apresentam diversas limitações, conforme discutidas neste capítulo. Este trabalho apresenta uma solução mais completa para o processamento de consultas em nuvem, pois pro-

põe um algoritmo de monitoramento não intrusivo e o particionamento da consulta é feito de maneira adaptativa.

5 PROPOSTA DE MONITORAMENTO E PROVISIONAMENTO PARA O PROCESSAMENTO DE CONSULTAS EM NUVEM

5.1 Preliminares

A seguir, são apresentados os algoritmos de particionamento e monitoramento que fazem parte da solução. Eles são executados durante o processamento das consultas de maneira não intrusiva. O algoritmo responsável por particionar a consulta é executado em uma única MV, sendo esta MV responsável por repassar as partições para as MVs que estão alocadas para o processamento da consulta. Já o algoritmo de monitoramento é realizado em cada máquina virtual que participa do processamento das partições. A Seção 5.3 apresenta o algoritmo de particionamento e a Seção 5.4 apresenta o algoritmo de monitoramento.

O algoritmo de provisionamento dinâmico das MVs, que também é apresentado neste capítulo, tem com objetivo de satisfazer o SLO das consultas e minimizar o custo computacional, considerando a carga de trabalho e a variação do desempenho das MVs. São propostos dois algoritmos para alcançar este objetivo. O primeiro algoritmo é descrito na Seção 5.2 e calcula a quantidade de MVs necessárias antes do início do processamento da consulta. Durante o processamento, o segundo algoritmo, apresentado na Seção 5.5, é executado quando uma MV detecta que o SLO_Q pode ser violado. Vale ressaltar que o algoritmo de monitoramento trata da tomada de decisão para garantir o processamento da consulta sem penalidade.

5.2 Provisionamento Inicial

O Algoritmo 1 implementa o método de provisionamento inicial. O propósito deste algoritmo é computar o menor conjunto V de máquinas virtuais (V=$\{mv_0, ..., mv_n\}$) que deve ser inicialmente alocado para processar Q satisfazendo o SLO_Q. Para cada mv_i alocada, é necessário definir a quantidade de tuplas de Q que mv_i é capaz de processar sem violar o SLO_Q, isto é, NT_i^Q. NT_i^Q é calculado de acordo com a capacidade de leitura (TL_i) dentro do tempo especificado no SLO_Q. Quando uma máquina virtual mv_i está sendo utilizada pela primeira vez, o valor de TL_i é obtido a partir da ferramenta *iostat*. Durante o processamento da consulta, TL_i é calculada pela divisão do número de tuplas recuperadas pelo tempo gasto para recuperá-las. O valor NT_i^Q para cada mv_i corresponde a uma parte da seletividade da consulta. O conjunto de máquinas virtuais presentes em V deve ser capaz de recuperar toda a seletividade de Q. A seletividade da consulta pode ser obtida por meio das estatísticas do plano de consulta presentes no SGBD. Por exemplo, nos SGBDs PostgreSQL e MySQL, o comando: `EXPLAIN` $\langle query \rangle$ pode ser usado para obter tais informações.

Algoritmo 1: Provisionamento Inicial

Entrada: Q, SLO_Q

1 $V \longleftarrow \emptyset$;
2 $Q_s \longleftarrow seletividade(Q)$;
3 **enquanto** $Q_s > 0 \wedge S \neq \emptyset$ **faça**
4 $\quad \langle mv_i, TF_i \rangle \longleftarrow S.remova()$;
5 $\quad$ **se** $TF_i > SLO_Q$ **então**
6 $\quad\quad NT_i^Q \longleftarrow SLO_Q \times TL_i$;
7 $\quad$ **senão**
8 $\quad\quad NT_i^Q \longleftarrow TF_i \times TL_i$;
9 $\quad$ **fim se**
10 $\quad Q_s \longleftarrow Q_s - NT_i^Q$;
11 $\quad V \longleftarrow V \cup \{mv_i\}$;
12 **fim enqto**
13 **enquanto** $Q_s > 0$ **faça**
14 $\quad mv_i \longleftarrow new()$;
15 $\quad NT_i^Q \longleftarrow SLO_Q \times TL_i$;
16 $\quad Q_s \longleftarrow Q_s - NT_i^Q$;
17 $\quad V \longleftarrow V \cup \{mv_i\}$;
18 **fim enqto**
19 $particionaQ(H, Q, SLO_Q, V)$;

Um ponto fundamental é aproveitar o tempo de folga das MVs. O Algoritmo 1 encontra em S o maior conjunto de mv_i cujo TF_i pode ser dedicado para processar Q. A escolha do maior conjunto permite que o provedor sirva uma quantidade maior de usuários utilizando menos MVs e satisfazendo todos os SLAs envolvidos. A escolha é feita usando uma estratégia gulosa, seguindo a ordem de S.

Seja S um conjunto ordenado de pares $\langle mv_i, TF_i \rangle$, que contém o tempo de folga TF_i para cada mv_i, tal que existe somente um elemento para cada mv_i e TF_i é maior que zero para todos os elementos. O conjunto S é ordenado por ordem decrescente do valor de TF_i.

O Algoritmo 1 calcula a seletividade de Q (linha 2). Se $S \neq \emptyset$ (linha 3), este algoritmo remove a primeira mv_i de S (linha 4) e calcula o número de tuplas que mv_i pode recuperar. Se $TF_i > SLO_Q$, então o valor de NT_i^Q é calculado baseado em SLO_Q e TL_i (linha 6). Por outro lado, o valor de NT_i^Q é calculado baseado em TF_i e TL_i (linha 8).

O segundo laço do Algoritmo 1 (linha 13) é responsável por distribuir as tuplas de Q que ainda faltam ser recuperadas (linha 10) para as novas MVs (linha 11). Quando uma nova máquina virtual mv_i é instanciada (linha 14), o algoritmo calcula o seu NT_i (linha 15). NT_i é calculado usando a taxa de leitura da MV (TL_i) e o valor do SLO_Q. O Algoritmo 1 termina quando não existem mais tuplas em Q a serem distribuídas. Em seguida, a estratégia particiona Q (Algoritmo 2 discutido na próxima seção), a fim de monitorar o desempenho de cada MV provisionada durante a execução de Q (linha 19).

5.3 Particionamento da Consulta

O algoritmo de particionamento distribui as devidas partições para cada MV disponível baseado no seu desempenho. Isso se torna uma tarefa complexa em ambientes de nuvem dinâmicos, onde o desempenho de cada MV pode variar com o tempo. Dessa forma, esse algoritmo usa uma tabela H para escolher o número adequado de partições para dividir uma consulta Q. É importante observar que a tabela H contém informação sobre o número máximo de partições que se pode dividir uma consulta Q satisfazendo o SLO_Q. Esse trabalho assume que para cada banco de dados e consulta submetida ao nosso sistema, é possível obter um número suficiente de amostras para a tabela H de maneira *offline*. Essa suposição é viável, pois o foco deste trabalho é o processamento de consultas OLAP.

O tempo de processamento médio presente em cada entrada da tabela H pode ser computado usando o comando do PostgreSQL: `EXPLAIN ANALYZE` $\langle query \rangle$. O tempo estimado de execução retornado por esse comando é baseado no plano da consulta, e este é normalmente menor do que o tempo de resposta real da consulta. Dessa forma, é possível que cada MV receba mais partições para monitorar do que o necessário para satisfazer o SLO_Q. Contudo, um número maior de partições permite um monitoramento mais frequente, melhorando a possibilidade de adaptação de acordo com as variações do ambiente.

O Algoritmo 2 detalha como as partições são criadas para uma consulta Q levando em consideração o SLO_Q, o desempenho de cada máquina virtual $mv_i \in V$ e a tabela de dados H. Este algoritmo funciona da seguinte forma: para cada mv_i alocada para processar Q, a consulta Q é reescrita em uma (sub)consulta Q_i (linha 2) que tem seletividade igual ao número de tuplas que mv_i pode recuperar sem violar o SLO_Q, isto é, NT_i^Q. Para que o desempenho da máquina virtual mv_i que irá processar Q_i seja monitorado, Q_i também é particionada. Para isso, este trabalho utiliza a técnica de particionamento virtual (LIMA; MATTOSO; VALDURIEZ, 2010). Assim, Q_i é em seguida particionada usando H por construir o maior conjunto de partições P_i a serem processadas por mv_i de tal sorte que o tempo médio de processamento de P_i não exceda o SLO_Q (linhas 3 e 4). Novamente, baseada no particionamento virtual. Por fim, após a construção de P_i, o Algoritmo 3 é aplicado dentro de cada mv_i (linha 5). Como obter o valor de NT_i^Q para cada mv_i é já foi discutido na Seção 5.2 e é relembrado na Seção 5.5.

Algoritmo 2: ParticionaQ

Entrada: $H, Q, SLO_Q, V = \{mv_0, .., mv_m\}$

1 **para cada** $mv_i \in V$ **faça**
2 $\quad Q_i \leftarrow$ subconsulta de Q com seletividade igual a NT_i^Q
3 $\quad n_i^Q \leftarrow$ consulta H, o maior número de partições para Q_i satisfazendo SLO_Q;
4 $\quad P_i \longleftarrow$ divide Q_i em n_i^Q partições;
5 $\quad monitora(P_i, SLO_Q)$;
6 **fim para cada**

5.4 Monitoramento do Processamento da Consulta

Para cada mv_i, o Algoritmo 3 monitora continuamente a taxa de leitura da MV e estima quanto tempo levará para terminar todas as partições em P_i. Inicialmente, a seletividade de cada partição é calculada de acordo com a seletividade do predicado associado ao atributo de particionamento (linha 1). É importante ressaltar que a chave primária é utilizada como atributo de particionamento em todas as consultas deste trabalho.

Para cada partição em P_i (linha 3), o Algoritmo 3 calcula o tempo gasto (linha 7) para processar a partição (linha 5). Com base na informação do tempo gasto, é possível saber quantas tuplas de P_i restam ser recuperadas (linha 9) e também fazer uma estimativa do tempo para recuperá-las (linha 10). O tempo estimado é baseado na taxa de leitura da MV, isto é, TL_i.

Algoritmo 3: Monitora

Entrada: P_i,SLO_Q.

```
1  P_s ⟵ seletividade(P_i);
2  enquanto P_i ≠ ∅ faça
3      q ⟵ P_i.remova();
4      T_inicio ⟵ tempo();
5      TP ⟵ execute(q);
6      T_fim ⟵ tempo();
7      T_q ⟵ T_fim − T_inicio
8      T_gasto ⟵ T_gasto + T_q;
9      P_s ⟵ P_s − |TP|;
10     T_estimado ⟵ P_s ÷ TL_i
11     TT_Q ⟵ T_gasto + T_estimado + TM_i;
12     se TT_Q − SLO_Q > 0 então
13         T_restante ⟵ SLO_Q − (T_gasto + TM_i);
14         retorna tomeDecisao(mv_i, P_i, T_restante);
15     fim se
16     senão se TT_Q − SLO_Q < 0 então
17         TF_i ⟵ SLO_Q − TT_Q;
18         possuiTempoFolga(TF_i);
19     fim se
20 fim enqto
21 removaMV(mv_i);
```

Se o tempo total estimado para processar Q (linha 11) é maior que SLO_Q (linha 12), isso indica que o SLO_Q pode ser violado. Note que também é considerado o tempo TM_i gasto com o monitoramento do desempenho de mv_i. Claramente, continuar processando P_i somente com mv_i pode levar a uma penalidade. A fim de usar o tempo restante (linha 13) para processar as partições remanescentes que estão em P_i sem violar o SLO_Q, alguma decisão deve ser tomada (linha 14). Nesse ponto, P_i é re-computado para mv_i, usando o Algoritmo 4 explicado ainda nesse capítulo na próxima seção.

O Algoritmo 3 também contempla a situação em que SLO_Q pode ser satisfeito mais rápido do que o esperado (linha 15). Nesse caso, este algoritmo usa o tempo de folga (linha 18) tanto para processar as novas consultas que chegam ao sistema quanto para diminuir o número de MVs alocadas. TF_i representa o valor do tempo de folga da mv_i que pode ser alocado sem violar o SLO_Q. As MVs com tempo folga são usadas pelos algoritmos de provisionamento que serão explicados na próxima seção, a fim de reutilizar os recursos das máquinas.

5.5 Provisionamento Dinâmico

Algoritmo 4: TomeDecisao

Entrada: $mv_i, P_i, T_{restante}$

1 $V \longleftarrow \emptyset$;
2 $Q' \longleftarrow juneParticoes(P_i)$;
3 $Q'_s \longleftarrow seletividade(Q')$;
4 $NT_i^{Q'} \longleftarrow T_{restante} \times TL_i$;
5 $Q'_s \longleftarrow Q'_s - NT_i^{Q'}$;
6 $V \longleftarrow V \cup \{mv_i\}$;
7 **enquanto** $Q'_s > 0 \wedge S \neq \emptyset$ **faça**
8 $\quad \langle mv_j, TF_j \rangle \longleftarrow S.remova()$;
9 $\quad$ **se** $TF_j > T_{restante}$ **então**
10 $\quad\quad NT_j^{Q'} \longleftarrow T_{restante} \times TL_j$;
11 $\quad$ **senão**
12 $\quad\quad NT_j^{Q'} \leftarrow TF_j \times TL_j$;
13 $\quad$ **fim se**
14 $\quad Q'_s \longleftarrow Q'_s - NT_j^{Q'}$;
15 $\quad V \longleftarrow V \cup \{mv_j\}$;
16 **fim enqto**
17 **enquanto** $Q'_s > 0$ **faça**
18 $\quad mv_j \longleftarrow new()$;
19 $\quad NT_j^{Q'} \longleftarrow T_{restante} \times TL_j$;
20 $\quad Q'_s \longleftarrow Q'_s - NT_i^{Q'}$;
21 $\quad V \longleftarrow V' \cup \{mv_j\}$;
22 **fim enqto**
23 $particionaQ(H, Q', T_{restante}, V)$;

Na fase de monitoramento, o Algoritmo 4 é chamado para cada mv_i que tem a possibilidade de violar SLO_Q. Suponha que mv_i tem um conjunto de partições P_i que ainda faltam ser processadas, e que devem ser executadas em $T_{restante}$ unidades de tempo, de tal sorte que o SLO_Q não seja violado. O Algoritmo 4 recalcula o número de MVs para colaborar com mv_i a concluir o processamento de P_i dentro de $T_{restante}$, com o objetivo de satisfazer ao SLO_Q. Primeiro, Algoritmo 4 recalcula quantas tuplas mv_i ainda pode recuperar sem violar o SLO_Q. Em seguida, o Algoritmo 4 provisiona um novo conjunto de MVs e aloca para cada uma delas

uma quantidade de tuplas a serem recuperadas.

De acordo com o Algoritmo 4, na linha 2, as partições restantes de P_i são escritas em uma única consulta Q'. Logo após, na linha 3, a seletividade de Q' é calculada e na linha 4, a quantidade de tuplas de Q' que mv_i pode processar, isto é, $NT_i^{Q'}$. Depois, a mesma ideia do Algoritmo 1 é aplicada e consiste em encontrar em S o maior conjunto de MVs cujo tempo de folga pode ser dedicado para processar Q' (linhas 7-13) e distribui as tuplas de Q' para cada uma das MVs.

Se ainda existem tuplas de Q' para serem processadas e $S = \emptyset$ (linha 17), novas MVs são alocadas (linha 18). Quando uma nova máquina virtual mv_j é instanciada, o algoritmo calcula o $NT_j^{Q'}$ (linha 20) por meio da taxa de leitura da MV (TL_j) e do valor de $T_{restante}$. O Algoritmo 4 termina quando não existem mais tuplas de Q' a serem distribuídas. Depois disso, nossa estratégia particiona Q' (o Algoritmo 2 é chamado) para monitorar o desempenho de cada MV provisionada durante a execução de Q', de forma a garantir que a execução de Q' dure apenas $T_{restante}$ unidades de tempo (linha 23).

5.6 Conclusão

Neste capítulo, foram apresentados os algoritmos de monitoramento e particionamento da consulta. Estes algoritmos tem como objetivo satisfazer o SLA, minimizar a penalidade e custo computacional, assim como tratar as variações de desempenho das máquinas. Essa etapa de monitoramento é muito importante. Tal proposta difere este trabalho de muitos outros trabalhos de processamento de consultas em nuvem. Além disso, foram apresentados os algoritmos de provisionamento inicial e dinâmico das máquinas virtuais que participam do processamento de consultas em um ambiente de nuvem. O objetivo desses algoritmos é provisionar uma quantidade adequada de MVs de modo que o custo computacional seja minimizado, assim como as penalidades, garantindo que o SLA seja satisfeito.

6 IMPLEMENTAÇÃO DO PROTÓTIPO E AVALIAÇÃO

Este capítulo apresenta um protótipo que implementa as estratégias apresentadas para o processamento de consultas *Select-range* e com *agregação* em ambientes de nuvem observando o SLA. A arquitetura do protótipo é apresentada na Seção 6.3. As consultas utilizadas durante os experimentos são descritas na Seção 6.4. A Seção 6.5 apresenta alguns resultados considerando a realização de consultas usando a base de dados TPC-H. Obtido os resultados dos experimentos, foi analisada a quantidade de máquinas utilizadas para o processamento das consultas e se o SLA das consultas foi obedecido.

6.1 Ambiente de Experimento

O protótipo proposto foi implementado na linguagem Java e executado na infraestrutura de nuvem da Amazon EC2 com máquinas do tipo *small*, isto é, o ambiente é homogêneo, mas relembre que o desempenho de cada MV, em particular, pode variar com o passar do tempo. Cada instância é uma máquina virtual com um processador Xeon de 2.4 GHz, 1.7 GB de memória e 160 GB de capacidade de disco. Foi criada uma imagem da MV na Amazon (ou em inglês, *Amazon Machine Image (AMI)*) contendo um SGBD e a base de dados. A imagem possibilita iniciar uma nova MV rapidamente. O *Amazon Elastic Block Store (EBS)* foi usado para armazenar a AMI.

O sistema operacional Ubuntu 11.10 e o SGBD PostgreSQL 9.1 foram utilizados nos experimentos. Em virtude de limitações da Amazon, foram utilizadas no máximo 16 MVs em qualquer instante de tempo. Nestes experimentos, foi considerada a existência de um número fixo de MVs disponíveis, pois o trabalho foca em aplicações OLAP que apresenta grande quantidade de dados e o tempo necessário para instanciar uma MV com esses dados poderia ser muito grande. Dessa forma, a ideia é otimizar a utilização dessas MVs. O *benchmark* TPC-H (TPC-H, 2012) foi usado somente para gerar a base de dados e este foi configurado com a escala 8, que permite gerar uma.base de dados de aproximadamente 13 GB. Esta base de dados é armazenada em SGBD e totalmente replicada em cada MV.

6.2 Metodologia

Os experimentos visam demonstrar que a abordagem deste trabalho: (i) é elástica em relação ao número de MV alocadas durante o processamento da consultas, e ainda (ii) é capaz de evitar penalidades associadas a violação do *SLO*. Os detalhes de cada um dos experimentos são apresentados a seguir.

Os experimentos foram realizados com consultas que possuem a mesma seletividade de 2 milhões de tuplas e o mesmo valor de *SLO*. A ideia do experimento não é medir o grau de interferência entre as consultas em relação ao tempo de processamento, mas medir a alocação de recursos do sistema. O sistema foi estressado com sete cargas de consultas. Cada

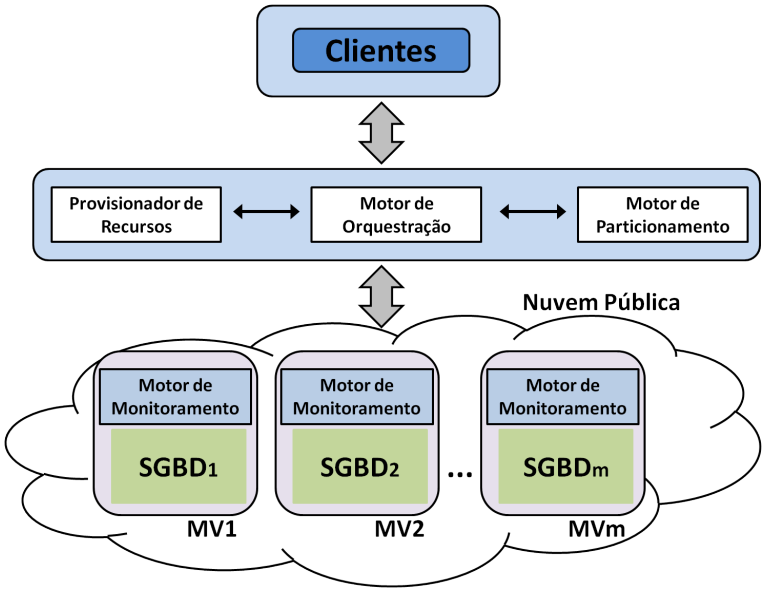


Figura 6.1: Arquitetura do Protótipo

carga possui de 8 a 13 consultas. A primeira carga possui 13 consultas, a segunda 11 consultas, a terceira e quarta 10 consultas, quinta 8 consultas, sexta 9 consultas e por fim, a sétima possui 11 consultas. O tempo de chegada das cargas seguiu uma distribuição Poisson (WILLIG, 1999) ($\lambda = 60$) e uma distribuição Uniforme com cada carga chegando em intervalos de 30 segundos. Foram usados diferentes valores de *SLOs*, do mais restrito ao mais relaxado. Os valores do tempo de chegada seguindo a distribuição de Poisson foram obtidos por meio do comando *poissrnd* pela ferramenta *Octave* (Gnu Octave, 2013).

Note que quando o *SLO* é mais restrito é razoável esperar que as consultas de uma carga ainda estarão executando quando as consultas da próxima carga chegarem. Isso permite investigar precisamente se a a solução é elástica, ou seja, adiciona/remove recursos de acordo com a quantidade de consultas.

Para representar um ambiente mais real, definiu-se a noção de teste, que é composto por cada sequência de 7 cargas submetidas ao sistema, sendo que cada teste é repetido 5 vezes. Considerando as 5 repetições de um mesmo teste, são apresentados o máximo, o mínimo e a média do número de MVs necessárias em função do tempo. A média do número de MVs foi arredondada e reportada como um valor inteiro. Finalmente, para eliminar todas as possíveis interferências entre os testes subsequentes, em particular os efeitos das outras consultas já executadas, nós limpamos a cache do SO e reiniciamos o SGBD antes de executar cada teste. O custo computacional foi contabilizado somando o número de MVs utilizadas a cada 60 segundos para os experimentos usando distribuição de Poisson e 30 segundos para os experimentos usando distribuição Uniforme.

6.3 Protótipo

A arquitetura do protótipo é apresentada na Figura 6.1, sendo composta por 4 módulos: *Motor de Particionamento*, *Motor de Monitoramento*, *Provisionador de Recursos* e *Motor de Orquestração*. O módulo *Motor de Particionamento* usa a tabela H e é responsável por particionar a consulta de tal sorte que o SLO não seja violado. O *Motor de Monitoramento* é executado dentro de cada MV mv_i alocada para processar uma consulta Q e visa garantir que cada MV processe as consultas dentro do SLO acordado. Durante o processamento da consulta, MVs podem ser podem ser requisitadas para impedir que o SLO da consulta seja violado. Ambos os módulos *Motor de Particionamento* e *Motor de Monitoramento* estão no centro da solução e implementam os algoritmos que foram discutidos no Capítulo 5 nas Seções 5.3 e 5.4.

O módulo *Provisionador de Recursos* (descrito nas Seções 5.2 e 5.5) inicialmente provisiona um número de MVs para processar a consulta Q dentro do acordado SLO_Q, minimizando o custo computacional e a penalidade. Esse módulo também toma algumas decisões quando o *Motor de Monitoramento* avisa que o SLO_Q está prestes a ser violado. Este foi um dos assuntos do Capítulo 5.

Finalmente, o módulo *Motor de Orquestração* se comunica com o *Provisionador de Recursos* para obter um provisionamento, e com o *Motor de Particionamento* para obter as partições da consulta, e em seguida envia as partições ao *Motor de Monitoramento*.

6.4 Consultas Utilizadas

Nos experimentos apresentados na próxima seção, as consultas são do tipo *Select-range* e com *agregação*. Utilizou-se a tabela *orders* do benchmark TPC-H e sua chave primária ($o\_orderkey$) como atributo de particionamento.

Para as consultas *Select-range* cujo predicado é sobre o atributo que não é chave primária, foi utilizado o atributo $o\_totalprice$ da tabela *orders* no predicado. O intervalo de valores para $o\_totalprice$ de cada consulta foi gerado randomicamente dentre os valores armazenados no banco de dados e com seletividade de aproximadamente 2 milhões de tuplas. Os valores do *SLO* adotados foram do mais restrito (80 segundos) a um valor mais relaxado (120 segundos).

Para as consultas *Select-range* cujo predicado é sobre o atributo que é chave primária $o\_orderkey$, o intervalo de valores no predicado de cada consulta também foi gerado randomicamente dentre os valores armazenados no banco de dados e com seletividade de aproximadamente 2 milhões de tuplas. Nesses experimentos, foi possível ser mais restrito quanto ao valor do *SLO*, pois o tempo de resposta dessas consultas é menor do que o tempo de resposta das consultas usadas nos experimentos com *Select-range*, cujo predicado é sobre o atributo não chave. Utilizou-se os valores mais restritos de SLO (60 segundos) a valores mais relaxados (100 segundos).

Os experimentos que envolvem apenas consultas com *agregação* que possuem um

predicado sobre o atributo *o_orderkey* são explicados na Seção 6.5.2. No caso da agregação, as mesmas consultas dos experimentos com *Select-range* cujo predicado é sobre um atributo que é chave primária foram utilizadas, apenas com uma modificação na cláusula SELECT (substituindo o "*"por AVG(*o_totalprice*)).

Nos experimentos usando apenas consultas com *agregação*, o objetivo foi utilizar a mesma seletividade e os mesmos valores do *SLO* que das consultas utilizadas nos experimentos com *Select-range* apresentados na Seção 6.5.1. Devido a limitações no esquema do TPC-H, as cargas das consultas não poderiam ser constituídas por consultas com *agregação* sem predicado na cláusula WHERE. O intervalo do predicado limita a seletividade da consulta e possibilita a criação de sete cargas com 8 a 13 consultas cada uma. Dessa forma, a variação do *SLO* é de 60 segundos a 100 segundos como nos experimentos da Seção 6.5.1.

6.5 Resultados Obtidos

A seguir, os resultados dos experimentos realizados com consultas *Select-range* são apresentados na Seção 6.5.1 e com consultas com *agregação* são apresentados na Seção 6.5.2. Os resultados apresentados mostram que em todos dos experimentos não houve penalidade.

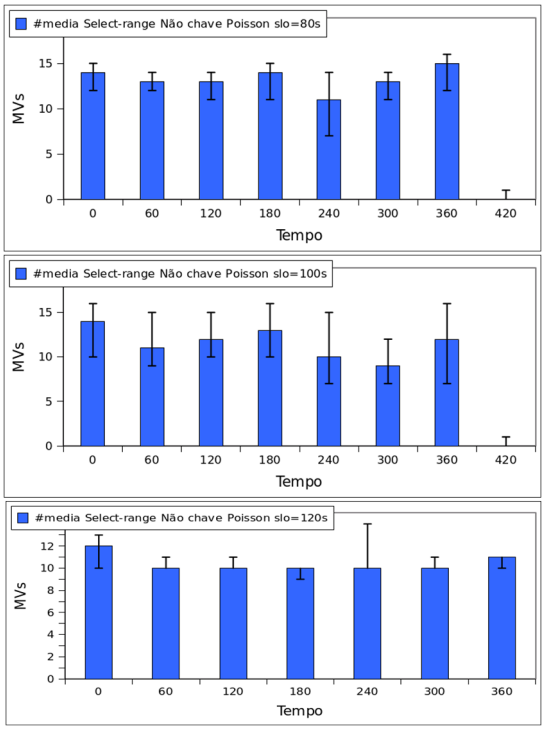
6.5.1 Consultas *Select-range*

Os gráficos apresentados a seguir mostram o número de MVs pelo tempo em segundos. As Figuras 6.2 e 6.3 apresentam os experimentos com consultas *Select-range* cujo predicado é sobre atributo não chave. Note que mesmo a carga das consultas sendo a mesma em todos os testes, vários fatores que estão fora do controle, por exemplo, outros processos executando na máquina física que hospeda a MV podem contribuir para uma variação de desempenho (SCHAD; DITTRICH; QUIANÉ-RUIZ, 2010). Assim, a variação acima e abaixo dos valores médios são reportados nas figuras.

As Figuras 6.4 e 6.5 apresentam os experimentos com *Select-range* cujo predicado é sobre atributo que é chave primária. O método dinamicamente aumenta e diminui a quantidade de MVs usadas durante o processamento de cargas de consultas *Select-range*, refletindo a adaptatividade desejada quanto ao aumento e a diminuição da carga, garantindo satisfazer o *SLO* das consultas. A Tabela 6.1 e Tabela 6.2 nos mostra a média do custo computacional de todos os testes realizados em função do *SLO* acordado.

Como para consultas *Select-range* cujo predicado é sobre atributo que não é chave primária, um novo predicado deve ser adicionado sobre o atributo de particionamento que no caso é chave primária da tabela. O intervalo de valores para o atributo de particionamento compreende do menor ao maior valor de chave primária armazenado no banco de dados. Dessa forma, todas as tuplas da tabela são checadas, verificando se satisfazem ao predicado sobre atributo não chave. Assim, para esses experimentos, as consultas *Select-range* com predicado sobre atributo não chave requerem mais tempo de processamento do que as *Select-range* que possuem predicado sobre atributo chave. Como o tempo de resposta é maior, as MVs são

Figura 6.2: Número de nós para diferentes valores de *SLO* em uma distribuição Poisson usando consultas *Select-range* cujo predicado é sobre atributo que é não chave primária.



alocadas por mais tempo, e consequentemente com um custo computacional maior é requerido, como é possível ver na Tabela 6.1.

Na Tabela 6.1, como esperado, para os valores de *SLO* mais restritos, o custo computacional foi maior ou igual ao custo computacional para valores de *SLO* mais relaxados, a fim de evitar possíveis penalidades. O mesmo padrão pode ser observado na Tabela 6.2. A diferença entre o custo computacional ser maior em uma distribuição Poisson do que na Uniforme pode estar relacionado ao "comportamento"em que as cargas de consultas chegam ao sistema. Devido ao comportamento não constante e inesperado da chegada das consultas em distribuição Poisson, mais MVs são alocadas e o custo computacional aumenta.

Tabela 6.1: Comparação da execução da solução proposta com variação de valores de *SLO* para consultas *Select-range* cujo predicado é sobre atributo não chave usando Distribuição de Poisson e Uniforme. Em todos os casos, *o método não resultou nenhuma penalidade*.

Valor do *SLO* [seg]	Custo Computacional [c_{MV}]
Uniforme	
80	83
100	74
120	72
Poisson	
80	93
100	81
120	73

Figura 6.3: Número de nós para diferentes valores de *SLO* em uma distribuição Uniforme usando consultas *Select-range* cujo predicado é sobre atributo que é não chave primária.

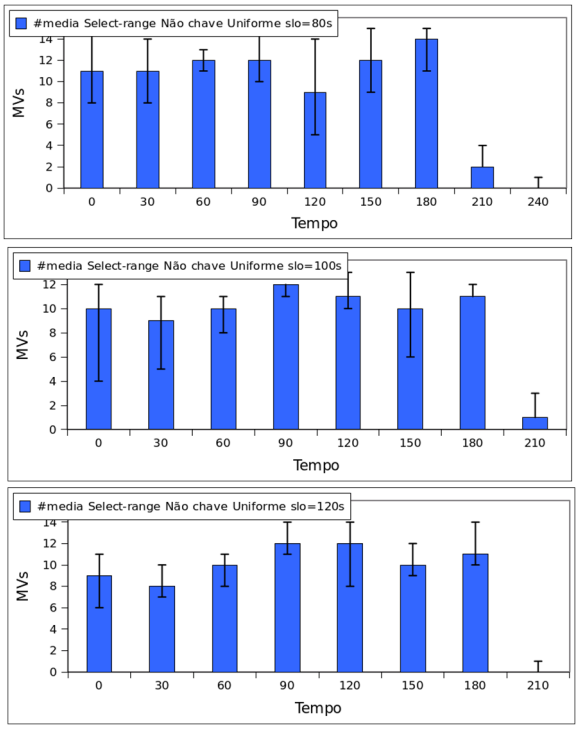


Tabela 6.2: Comparação da execução do método proposto neste trabalho com variação de valores de *SLO* para consultas *Select-range* cujo predicado é sobre atributo que é chave primária usando Distribuição de Poisson e Uniforme. Em todos os casos, *o método não resultou nenhuma penalidade*.

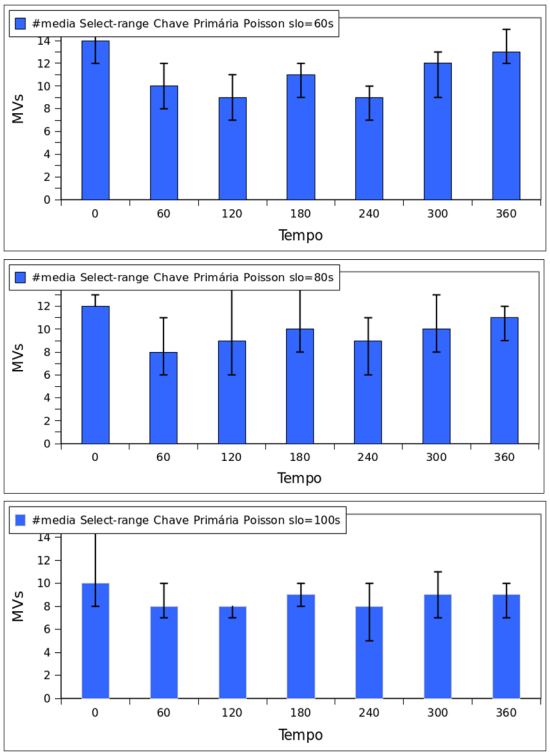
Valor do *SLO* [seg]	Custo Computacional [c_{MV}]
Uniforme	
60	62
80	61
100	61
Poisson	
60	78
80	69
100	61

6.5.2 Consultas com *agregação*

Os resultados dos experimentos utilizando apenas consultas com *agregação* são apresentados nas Figuras 6.6 e 6.7. Note que o valor do *SLO* também foi variado, do mais restrito (60 segundos) para o mais relaxado (100 segundos). Pode-se observar um comportamento similar ao ocorrido com as consulta *Select-range*. Diante do aumento e da diminuição da carga, é possível perceber a elasticidade no número de máquinas virtuais alocadas pelo método proposto neste trabalho para processar as consultas.

A Tabela 6.3 mostra o custo computacional dos experimentos usando consultas com *agregação*. Nenhuma penalidade ocorreu nesses experimentos e os resultados confirmam que

Figura 6.4: Número de nós para diferentes valores de *SLO* em uma distribuição Poisson usando consultas *Select-range* cujo predicado é sobre atributo que é chave primária.

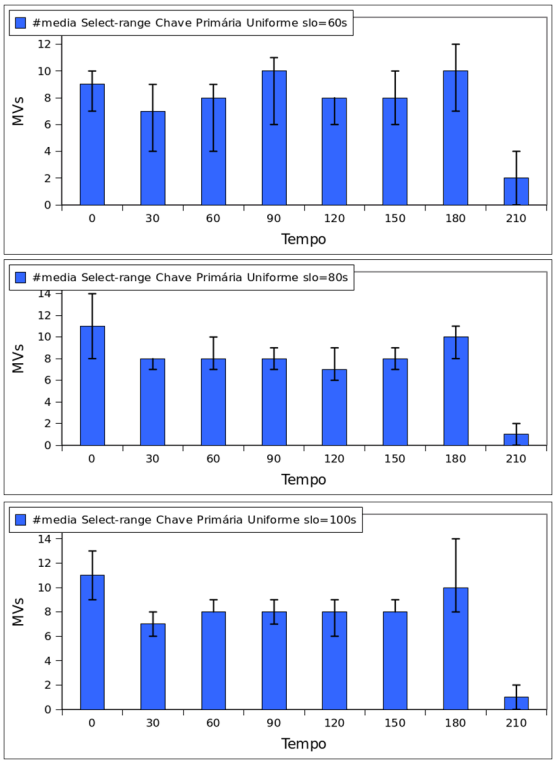


para os valores mais restritos do *SLO*, o custo computacional foi maior ou igual ao custo computacional para valores mais relaxados, evitando possíveis penalidades. Estes experimentos poderiam utilizar *SLO* com valores mais restritos do que nos experimentos com consultas *Select-range*. As consultas com *agregação* apresentaram tempo de resposta menor do que as consultas utilizadas nos experimentos de *Select-range*.

Dessa forma, as consultas *Select-range* cujo predicado é sobre atributo que é chave primária consomem uma quantidade de tempo maior para serem processadas, já que é necessário recuperar todos os valores dos atributos (uma grande quantidade de bytes) presente na cláusula SELECT, ou ainda mais operações de I/O são realizadas. Acreditamos que existe uma sobrecarga para trazer esses dados para a memória, ao passo que as consultas com *agregação* precisam recuperar somente o valor agregado do atributo $o\_totalprice$.

Também foram executados testes em que o número de MVs alocadas era fixo, a fim de comparar os resultados utilizando o método elástico proposto neste trabalho. Utilizando um número fixo, o usuário não tem garantias de que o valor escolhido será suficiente para atender ao SLO ou usará VMs em excesso. Corre o risco de subestimar (economizar no custo computacional com MVs, porém pagar penalidade por violar o *SLO* de algumas consultas) ou superestimar (pagar por mais e mais MVs do que o necessário). A Tabela 6.4 mostra a média do número de máquinas escolhidas pela estratégia proposta neste trabalho durante os experimentos das consultas *Select-range* e das consultas com *agregação*, tal que o *SLO* de todas as consultas é o mais restrito (ou seja, 80 segundos para *Select-range* cujo predicado é usando um atributo que não é chave primária, e 60 segundos para *Select-range* cujo predicado é sobre um atributo

Figura 6.5: Número de nós para diferentes valores de *SLO* em uma distribuição Uniforme usando consultas *Select-range* cujo predicado é sobre atributo que é chave primária.



chave primária e para consultas com *agregação*).

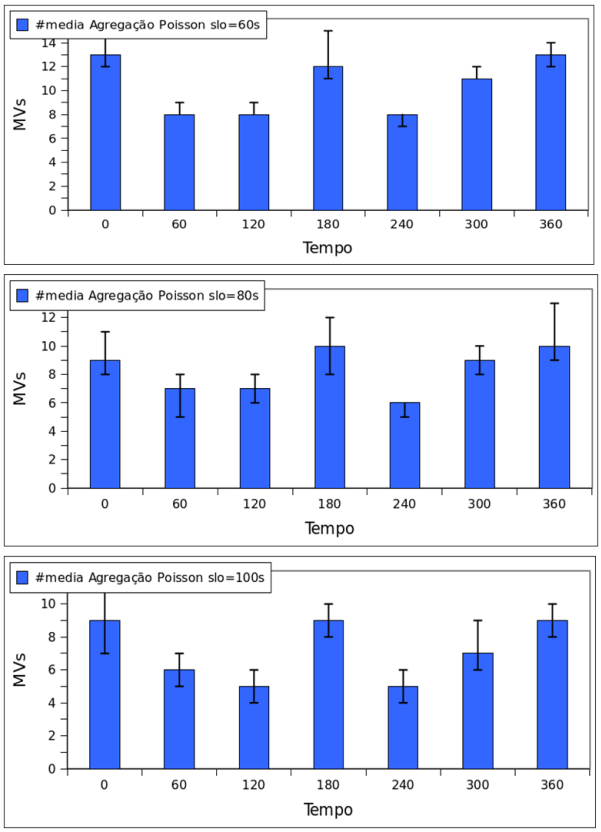
Tabela 6.3: Comparação da execução do método proposto neste trabalho com variação de valores de *SLO* para consultas com *agregação* usando Distribuição de Poisson e Uniforme. Em todos os casos, *o método não resultou nenhuma penalidade*.

Valor do *SLO* [sec]	Custo Computacional [c_{MV}]
Uniforme	
60	58
80	55
100	45
Poisson	
60	73
80	58
100	50

Fazer a escolha correta do número de MVs é uma tarefa complexa. Para sabermos a quantidade mínima de máquinas requeridas, testes prévios foram executados utilizando um número fixo de máquinas durante todo o processamento das consultas da seguinte forma: inicialmente, escolhe-se apenas 1(uma) máquina virtual para ser alocada ao processamento da consulta e calcula a penalidade associada. Se houve penalidade, realiza o teste com duas máquinas, e logo após calcula a penalidade. E assim sucessivamente, até encontrar um valor mínimo do número de máquinas fixo em que não haja penalidade. Cada teste foi executado 5 vezes e a média da penalidade foi calculada, devido as variações de desempenho do ambiente.

Note que vários testes prévios precisam ser executados para definir a menor quan-

Figura 6.6: Número de nós para diferentes valores de *SLO* em uma Distribuição Poisson usando consultas com *agregação*.



tidade de MVs e suas respectivas penalidades no processamento das consultas. Note que o número mínimo de máquinas necessário foi encontrado para uma configuração da carga de trabalho (composta por consultas) dada de entrada. Se outras consultas chegassem ao sistema, seria necessário executar testes exaustivos novamente.

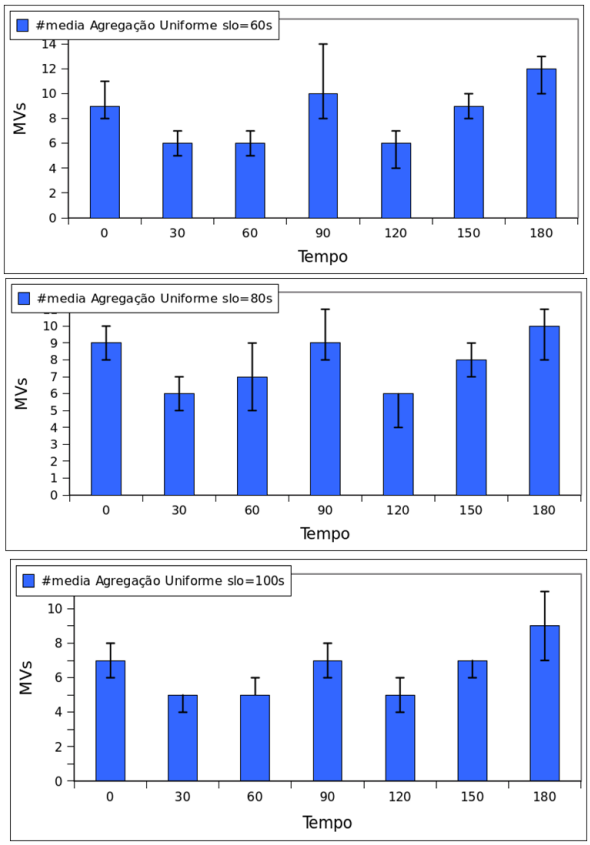
Tabela 6.4: Comparação da média do número de MVs do método proposto neste trabalho, e o número mínimo de MVs que pode ser usado para que não haja penalidades.

Tipo da Consulta	Média Alocada	Mínimo Sem Penalidade
Poisson		
Select-range sobre chave primária	11	7
Select-range sobre chave não primária	13	9
Agregação	10	7
Uniforme		
Select-range sobre chave primária	8	7
Select-range sobre chave não primária	10	9
Agregação	8	7

Usuários podem confiar em técnicas automatizadas de predição do tempo de execução das consultas pelas MVs, mas estas técnicas apresentam uma margem de erro considerável ou requerem, em tempo de execução, informações aprofundadas que nem sempre estão disponíveis (ALVES; BIZARRO; MARQUES, 2011).

Se o provedor da nuvem tentar definir a quantidade de MVs necessárias para o pro-

Figura 6.7: Número de nós para diferentes valores de *SLO* em uma Distribuição Uniforme usando consultas com *agregação*



cessamento das consultas usando uma estratégia que subestima o valor, ele irá pagar menos pelas MVs alocadas, porém isso pode ocasionar em penalidade a ser paga pelas consultas cujo *SLO* não foi satisfeito. Por outro lado, para estratégias que usam estimativas pessimistas ou superestimam a quantidade de MVs alocadas, não se tem penalidades, mas o custo computacional pode ser bem maior.

Os resultados obtidos pelo método proposto neste trabalho mostram que a solução elástica reage à variação do ambiente e a diferentes tamanhos de cargas de consulta. Tal solução assegura que o *SLO* é satisfeito de maneira não intrusiva e automática. Além disso, os resultados confirmam que o método não ocasiona penalidades, o que é importante, visto que, em geral, estas penalidades possuem um custo maior do que o custo computacional.

6.6 Conclusão

Neste capítulo foi apresentada uma implementação da solução proposta no Capítulo 5. Inicialmente, o ambiente de experimentação foi descrito na Seção 6.1 e a definição dos experimentos realizados foi apresentada na Seção 6.2. Posteriormente, informações sobre a implementação do protótipo, assim como os tipos de consultas utilizadas nos experimentos foram apresentadas nas Seções 6.3 e 6.4, respectivamente. Finalmente, os resultados dos experimentos foram apresentados na Seção 6.5, mostrando que a abordagem possui um comportamento

elástico em relação ao número de máquinas alocadas para processamento das consultas e que o *SLO* foi satisfeito em todos os experimentos, sem incorrer em penalidades.

7 CONCLUSÃO E TRABALHOS FUTUROS

Parte desse trabalho já foi validada pela comunidade em (SILVA et al., 2012). Esta dissertação apresentou o problema de processar consultas em ambientes de nuvem, considerando que as máquinas virtuais podem ter desempenho variável ao longo do tempo. Neste problema, cada consulta está associada a uma qualidade de serviço descrita pelo SLA e que deve ser satisfeita. A dificuldade do problema está em satisfazer o SLA com o custo computacional mínimo durante o processamento da consulta. Ou ainda, oferecer uma solução elástica, capaz de decidir quando uma máquina virtual deve ser adicionada ou removida de tal sorte que o SLA continue a ser satisfeito.

Esta dissertação apresentou conceitos básicos para entender melhor o problema sobre Computação em Nuvem e Processamento de Consultas Distribuído no Capítulo 2. O Capítulo 3 apresentou o problema de **processar consultas em ambientes de nuvem de maneira elástica e satisfazendo o SLA**. Um estudo de caso foi descrito nesse capítulo para motivar o problema e introduzir a solução proposta. No Capítulo 4, os trabalhos relacionados foram destacados e discutidos. O Capítulo 5 se concentra em apresentar a solução composta por algoritmos de monitoramento e provisionamento dinâmico para o processamento de consultas em nuvem de forma elástica e que o SLA seja satisfeito. No Capítulo 6, são abordados aspectos sobre a implementação do protótipo para que se possa avaliar a proposta e analisar seus resultados.

7.1 Objetivos Alcançados

No Capítulo 1 desta dissertação, os objetivos a serem alcançados foram listados, visando resolver o problema de processar consultas em ambientes de nuvem de maneira elástica e de acordo com o SLA. Durante todo o trabalho realizado, procurou-se alcançar todos os objetivos específicos, objetivando solucionar o problema. Dessa forma, foi proposta uma técnica não intrusiva, automática e adaptativa de monitoramento do desempenho de um SGBD em ambientes de nuvem para consultas *Select-range* e com *agregação*. Bem como, outro método que dinamicamente minimiza o número de MVs necessárias para satisfazer o SLO das consultas.

Nos experimentos também foram destacadas as dificuldades de provisionar uma quantidade ideal de máquinas virtuais para o processamento das consultas, tal que o SLA seja satisfeito e o custo computacional seja mínimo. Há técnicas que podem subestimar ou superestimar a quantidade de máquinas necessárias para o processamento das consultas. Ao utilizar uma estratégia que subestima o valor, o provedor irá pagar menos pelas MVs alocadas, porém as chances são maiores de haver uma penalidade a ser paga, pelas consultas cujo *SLO* não foi satisfeito. Por outro lado, ao tentar usar estratégias que superestimam, não há penalidades, mas o custo computacional pode ser bem maior.

Finalmente, os resultados dos experimentos confirmam que a abordagem desse trabalho possui um comportamento elástico em relação ao número de máquinas alocadas para processamento das consultas e que o *SLO* foi satisfeito em todos os experimentos, sem incorrer em penalidades. A solução proposta assegura que o *SLO* é satisfeito por meio de uma abordagem

não-intrusiva e automática. Além disso, os resultados confirmam que método proposto neste trabalho não ocasiona penalidades, o que é importante, visto que, em geral, estas penalidades possuem um custo maior do que o custo computacional.

7.2 Trabalhos Futuros

Inicialmente, o objetivo é como resolver o mesmo problema desta dissertação para consultas com junção. Em um segundo momento, pretende-se realizar mais experimentos e modificar o modelo de custo apresentado no Capítulo 3 para envolver custo de E/S e armazenamento. Outro trabalho futuro é investigar a influência de outros parâmetros, não só a taxa de leitura de cada máquina virtual, que podem influenciar no provisionamento das máquinas, como CPU e memória disponível em cada MV.

REFERÊNCIAS BIBLIOGRÁFICAS

ALVES, D.; BIZARRO, P.; MARQUES, P. Deadline queries: Leveraging the cloud to produce on-time results. In: *IEEE International Conference on Cloud Computing (CLOUD)*. [S.l.: s.n.], 2011. p. 171 –178. ISSN 2159-6182.

Amazon Auto Scaling. 2012. Disponível em: <http://aws.amazon.com/pt/autoscaling/>.

Amazon Cloud Watch. 2012. Disponível em: <http://aws.amazon.com/pt/cloudwatch/>.

Amazon EC2. *Amazon Elastic Compute Cloud*. 2013. Disponível em: <http://aws.amazon.com/pt/ec2/>.

ARNAUT, D.; SCHROEDER, R.; HARA, C. S. Phoenix: A relational storage component for the cloud. In: *IEEE International Conference on Cloud Computing, CLOUD 2011, Washington, DC, USA*. [S.l.: s.n.], 2011. p. 684–691.

CERVINO, J.; KALYVIANAKI, E.; SALVACHUA, J.; PIETZUCH, P. Adaptive provisioning of stream processing systems in the cloud. In: *IEEE 28th International Conference on Data Engineering Workshops (ICDEW)*. [S.l.: s.n.], 2012. p. 295 –301.

CHANG, F.; DEAN, J.; GHEMAWAT, S.; HSIEH, W. C.; WALLACH, D. A.; BURROWS, M.; CHANDRA, T.; FIKES, A.; GRUBER, R. E. Bigtable: a distributed storage system for structured data. In: *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2006. (OSDI '06), p. 15–15.

Chhanda Ray. *Distributed Database Systems*. Delhi, India: Pearson/Dorling Kindersley, 2009.

CHI, Y.; MOON, H. J.; HACIGÜMÜS, H.; TATEMURA, J. Sla-tree: a framework for efficiently supporting sla-based decisions in cloud computing. In: *Proceedings of the 14th International Conference on Extending Database Technology*. New York, NY, USA: ACM, 2011. (EDBT/ICDT '11), p. 129–140.

COOPER, B. F.; RAMAKRISHNAN, R.; SRIVASTAVA, U.; SILBERSTEIN, A.; BOHANNON, P.; JACOBSEN, H.-A.; PUZ, N.; WEAVER, D.; YERNENI, R. Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, VLDB Endowment, v. 1, n. 2, p. 1277–1288, ago. 2008. ISSN 2150-8097. Disponível em: <http://dl.acm.org/citation.cfm?id=1454159.1454167>.

CURINO, C.; JONES, E. P.; MADDEN, S.; BALAKRISHNAN, H. Workload-aware database monitoring and consolidation. In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. New York, NY, USA: ACM, 2011. (SIGMOD '11), p. 313–324. ISBN 978-1-4503-0661-4. Disponível em: <http://doi.acm.org/10.1145/1989323.1989357>.

DECANDIA, G.; HASTORUN, D.; JAMPANI, M.; KAKULAPATI, G.; LAKSHMAN, A.; PILCHIN, A.; SIVASUBRAMANIAN, S.; VOSSHALL, P.; VOGELS, W. Dynamo: amazon's highly available key-value store. In: *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*. New York, NY, USA: ACM, 2007. (SOSP '07), p. 205–220.

ELMORE, A. J.; DAS, S.; AGRAWAL, D.; ABBADI, A. E. Zephyr: live migration in shared nothing databases for elastic cloud platforms. In: *SIGMOD '11*. [S.l.: s.n.], 2011. p. 301–312. ISBN 978-1-4503-0661-4.

EMEAKAROHA, V. C. *Managing Cloud Service Provisioning and SLA Enforcement via Holistic Monitoring Techniques*. Tese (PhD) — Vienna University of Technology, 2012.

Gnu Octave. 2013. Disponível em: <http://www.gnu.org/software/octave/>.

ISLAM, S.; LEE, K.; FEKETE, A.; LIU, A. How a consumer can measure elasticity for cloud platforms. In: *Proceedings of the 3rd joint WOSP/SIPEW International conference on Performance Engineering*. New York, NY, USA: ACM, 2012. (ICPE '12), p. 85–96. ISBN 978-1-4503-1202-8. Disponível em: <http://doi.acm.org/10.1145/2188286.2188301>.

LAKSHMAN, A.; MALIK, P. Cassandra: structured storage system on a p2p network. In: *Proceedings of the 28th ACM symposium on Principles of distributed computing*. New York, NY, USA: ACM, 2009. (PODC '09), p. 5–5. ISBN 978-1-60558-396-9. Disponível em: <http://doi.acm.org/10.1145/1582716.1582722>.

LIMA, A. A. B.; MATTOSO, M.; VALDURIEZ, P. Adaptive virtual partitioning for olap query processing in a database cluster. *Journal of Information and Data Management*, v. 1, p. 75–88, 2010.

MELL, P.; GRANCE, T. The nist definition of cloud computing. *NIST*, p. 50, July 2009.

MIAN, R.; MARTIN, P.; VAZQUEZ-POLETTI, J. L. Provisioning data analytic workloads in a cloud. *Future Generation Computer Systems*, n. 0, 2012.

ÖZSU, M. T.; VALDURIEZ, P. *Principles of Distributed Database Systems*. Third. [S.l.]: Springer, 2011.

PAPADIAS, D.; KALNIS, P.; ZHANG, J.; TAO, Y. Efficient olap operations in spatial data warehouses. In: *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*. London, UK, UK: Springer-Verlag, 2001. (SSTD '01), p. 443–459. ISBN 3-540-42301-X. Disponível em: <http://dl.acm.org/citation.cfm?id=647227.719095>.

POPESCU, A. D.; DASH, D.; KANTERE, V.; AILAMAKI, A. Adaptive query execution for data management in the cloud. In: *Proceedings of the 2nd International Workshop on Cloud data management*. New York, NY, USA: ACM, 2010. (CloudDB '10), p. 17–24. ISBN 978-1-4503-0380-4. Disponível em: <http://doi.acm.org/10.1145/1871929.1871933>.

ROGERS, J.; PAPAEMMANOUIL, O.; CETINTEMEL, U. A generic auto-provisioning framework for cloud databases. In: *IEEE 26th International Conference on Data Engineering Workshops (ICDEW)*. [S.l.: s.n.], 2010. p. 63 –68.

SCHAD, J.; DITTRICH, J.; QUIANÉ-RUIZ, J.-A. Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proceedings of VLDB Endowment*, VLDB Endowment, v. 3, n. 1-2, p. 460–471, set. 2010. ISSN 2150-8097. Disponível em: <http://dl.acm.org/citation.cfm?id=1920841.1920902>.

SHARMA, U.; SHENOY, P.; SAHU, S.; SHAIKH, A. A cost-aware elasticity provisioning system for the cloud. In: *31st International Conference on Distributed Computing Systems (ICDCS)*. [S.l.: s.n.], 2011. p. 559 –570. ISSN 1063-6927.

SILVA, T. L. Coelho da; NASCIMENTO, M. A.; MACêDO, J. A. F.; SOUSA, F. R. C.; MACHADO, J. C. Towards non-intrusive elastic query processing in the cloud. In: *Proceedings of the 4th International Workshop on Cloud data management*. [S.l.: s.n.], 2012. (CloudDB '12), p. 9–16.

SOUSA, F. R. C.; MOREIRA, L. O.; SANTOS, G. A. C.; MACHADO, J. C. Quality of service for database in the cloud. In: *Proceedings of the 2nd International Conference on Cloud Computing and Services Science - CLOSER '12*. [S.l.: s.n.], 2012. p. 595–601.

TPC-H. *TPC-H Benchmark*. 2012. Disponível em: <http://www.tpc.org/tpch/>.

VIGFUSSON, Y.; SILBERSTEIN, A.; COOPER, B. F.; FONSECA, R. Adaptively parallelizing distributed range queries. *Proceedings of VLDB Endowment*, VLDB Endowment, v. 2, n. 1, p. 682–693, ago. 2009. ISSN 2150-8097. Disponível em: <http://dl.acm.org/citation.cfm?id=1687627.1687705>.

WILLIG, A. *A Short Introduction to Queueing Theory*. [S.l.]: Technical University Berlin, Telecommunication Networks Group, 1999.

ZHAO, J.; HU, X.; MENG, X. Esqp: an efficient sql query processing for cloud data management. In: *Proceedings of the 2nd International Workshop on Cloud data management*. New York, NY, USA: ACM, 2010. (CloudDB '10), p. 1–8. ISBN 978-1-4503-0380-4. Disponível em: <http://doi.acm.org/10.1145/1871929.1871931>.