Universidade Federal do Paraná

Simone Dominico

Provisionamento de Recursos Computacionais baseado em Redes de Petri para Bancos de Dados Orientados a Leitura

Curitiba PR
2016

Simone Dominico

Provisionamento de Recursos Computacionais baseado em Redes de Petri para Bancos de Dados Orientados a Leitura

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Dr. Eduardo Cunha de Almeida.

Co-orientador: Dr. Jorge Augusto Meira.

Curitiba PR
2016

Dominico, Simone
Provisionamento de recursos computacionais baseado em Redes de Petri para bancos de dados orientados a leitura / Simone Dominico. – Curitiba, 2016
72 f. : il.; tabs.

Dissertação (mestrado) – Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.
Orientador: Eduardo Cunha de Almeida
Coorientador: Jorge Augusto Meira

1. Computação. 2. Redes de petri. 3. Banco de dados - Gerência I. Almeida, Eduardo Cunha de. II. Meira, Jorge Augusto. III. Título

CDD 005.74

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
Setor CIÊNCIAS EXATAS
Programa de Pós Graduação em INFORMÁTICA
Código CAPES: 40001016034P5

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **SIMONE DOMINICO**, intitulada: "**Provisionamento de Recursos Computacionais baseado em Redes de Petri para Bancos de Dados Orientados a Leitura**", após terem inquirido a aluna e realizado a avaliação do trabalho, são de parecer pela sua aprovação.

Curitiba, 19 de Abril de 2016.

Prof EDUARDO CUNHA DE ALMEIDA (UFPR)
(Presidente da Banca Examinadora)

Prof CARLOS ALBERTO MAZIERO (UFPR)

Prof JORGE AUGUSTO MEIRA (SNT)
(Coorientador)

Prof LUIS ALLAN KÜNZLE (UFPR)

Prof MARCO ANTONIO ZANATA ALVES (UFPR)

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

Dedico esta dissertação a meu amado esposo que sempre que eu pensava em desistir me dava forças para continuar e sempre esteve ao meu lado em todas as decisões. Obrigada por sempre estar ao meu lado me dando forças, eu te amo!

Agradecimentos

Agradeço a DEUS primeiramente, por me conceder esta oportunidade em minha vida, me dando forças e esperanças para a conclusão de mais esta conquista. A meu esposo Tiago e filhos Agatha e Murilo pela compreensão nesta etapa tão importante em minha vida, me apoiando e auxiliando em todos os momentos necessários, mesmo quando acreditava que seria impossível.

Agradeço aos meus orientadores Eduardo Cunha de Almeida e Jorge Augusto Meira, por todos os conselhos, orientações, conhecimentos compartilhados e auxilio durante o mestrado. Agradeço a todos os professores pelos conhecimentos adquiridos dentro e fora da sala de aula, em especial ao professor Luis Allan Kunzle.

Ao professor Tarcizio Alexandre Bini pelo incentivo no inicio do mestrado e os e-mails trocados. A minha amiga Carolina pelas conversas e angústias compartilhadas. Aos colegas orientandos do professor Eduardo, pelas dicas durante as oficinas e conversas.

Aos professores da banca examinadora Carlos Masiero, Luis Allan Kunzle e Marco Zanata, pelo tempo dedicado, atenção e contribuições a este trabalho.

Ao Departamento de Informática da Universidade Federal do Paraná pelo comprometimento e profissionalismo. Ao André Ziviane e ao professor Marcos Castilho pela disponibilização do ambiente computacional necessário para desenvolvimento dos experimentos.

Agradeço à Capes pela bolsa de estudo durante o mestrado e a Fundação da Universidade Federal do Paraná (Funpar) pela bolsa nos últimos meses do mestrado. Agradeço a todos pelo incentivo, apoio e auxílio durante o mestrado que direta ou indiretamente ajudaram na finalização de mais esta caminhada.

Resumo

O provisionamento de recursos é uma técnica utilizada para alocar recursos computacionais em ambientes de alto desempenho. Tais ambientes estão sujeitos a processar diferentes padrões de carga de trabalho (e.g., e-commerce), incluindo picos de carga durante datas específicas, como por exemplo, *black friday*, natal e páscoa. Através do provisionamento é possível adicionar e remover recursos conforme a necessidade apresentada pelo sistema. Neste trabalho nos concentramos no provisionamento de núcleos de CPU para processamento de consultas em bancos de dados. Nós propomos um modelo de alto nível para sincronização dinâmica de múltiplos núcleos para processamento de consultas. Nosso modelo chamado de PrT-PRO é baseado em um provisionamento dinâmico reativo utilizando Redes de Petri Predicado/Transição, que atua por meio de regra-condição-ação no topo do monitoramento de desempenho. Através da PrT-PRO, busca-se obter um valor ótimo de múltiplos núcleos que atenda a demanda das consultas com objetivo de melhorar seu desempenho. Nosso modelo foi validado através de experimentos no popular sistema gerenciador de banco de dados (SGBD) PostgreSQL. Os resultados demonstram que ao encontrar o valor ótimo de múltiplos núcleos utilizando a PrT-PRO diminuímos substancialmente os *misses* de *cache* de CPU quando comparado com a execução utilizando todos os recursos disponíveis no hardware. A melhora de desempenho no processamento de consultas fica evidente, pois ao diminuir os *misses* de CPU diminui também o tempo de execução de uma determinada carga de trabalho. Assim, podemos afirmar que a PrT-PRO apresenta um melhor aproveitamento de CPU comparado com o atual modelo interno do SGBD PostgreSQL. Utilizando a PrT-PRO o SGBD PostgreSQL foi capaz de sincronizar o acesso aos múltiplos núcleos para acomodar leituras simultâneas com tipos mistos de acesso a CPU.

Palavras-chave: Provisionamento de Recursos Computacionais, SGBD, Redes de Petri.

Abstract

Resource provisioning is a technique to efficiently allocate computational resources on-demand in high-performance environments. The goal is to provision resources upon every running condition, even if the environment is challenged by different workload patterns (e.g., e-commerce), including peak loads during specific dates, for example, black friday, and christmas. In this dissertation, we focus on resource provisioning of multi-core CPUs for query processing. Our goal is to present a multi-core harnessing model to reduce response time for query processing in relational database systems. We present a high-level model for dynamic synchronization of multi-core in query processing. Our model called PrT-PRO is based on reactive dynamic provisioning using Petri Nets Predicate/Transition, which operates through rule-condition-action on top of the performance monitoring. The PrT-PRO seeks an optimal number of CPU cores to quickly respond to on-line needs of query processing. We validate the PrT-PRO on top of the popular open-source DBMS PostgreSQL. The results show that the optimal number of cores given by PrT-PRO substantially reduces the cache misses of CPU when compared with results using all available resources. Moreover, we show that decreasing the cache misses of CPU, it also decreases the response time to execute a particular workload. Thus, we can affirm that the PrT-PRO features a better harnessing of CPU compared with the current internal model of PostgreSQL.

Keywords: Computing Resouce provisioning, DBMS, Petri nets.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

ANS	Acordo de Nível de Serviço
ARIMA	*Autoregressive Integrated Moving Average*
CPU	*Central Processing Unit*
DSM	*Database State Machine*
EBS	*Elastic Block Storage*
ERP	*Enterprise Resource Planning*
GB	*Gigabyte*
KB	*Kilobyte*
L1	*Level* 1
L2	*Level* 2
L3	*Level* 3
MB	*Megabyte*
OLAP	*Online Analytical Processing*
PrT	Redes de Petri Predicado/Transição
PRT-PRO	Prt de Provisionamento de Recursos Computacionais
QdS	Qualidade de Serviço
QPNs	*Queueing Petri Nets*
RD	Recurso Disponível no *hardware*
RdP	Redes de Petri
SGBD	Sistema Gerenciador de Banco de Dados
SVR	*Support Vector Regression*
TPC	*Transaction Performance Council*
VM	*Virtual Machine*

Capítulo 1

Introdução

Com o recente crescimento de volume, velocidade e variedade na aquisição e tratamento de dados, torna-se primordial o uso eficiente de recursos computacionais disponíveis em *hardware* com o objetivo de processar dados em tempo hábil. Por tempo hábil, entendemos que o tempo de processamento de grandes volumes de dados deve ser compatível com os tempos de processamento reportados pelos principais *benchmarks* da literatura, como por exemplo: TPC-H e TPC-DS [1].

Neste trabalho, definimos o provisionamento de recursos computacionais como a alocação dos recursos computacionais disponíveis no hardware para processamento de dados. Os recursos são: CPUs, núcleos da CPU (*cores*), nodos computacionais ou espaço de armazenamento. Por exemplo, em sistemas de nuvens computacionais o provisionamento de recursos é peça essencial na otimização do uso da infraestrutura disponibilizada aos clientes. O provisionamento de recursos é utilizado para garantir a elasticidade sob demanda, ou seja, os recursos são provisionados baseado na necessidade apresentada pelo sistema sem desrespeitar o Acordo de Nível de Serviço (ANS) previamente estipulado. Dessa forma, por meio do provisionamento de recursos, é possível oferecer mais processamento e armazenamento sob demanda [Vergara et al., 2014].

Uma vez que o aumento do volume de dados é constante, a necessidade do uso do provisionamento de recursos se torna evidente. Mais especificamente, o aumento da capacidade de processamento é particularmente importante no processamento de consultas, tendo impacto direto no Sistema Gerenciador de Banco de Dados (SGBD). O SGBD é o software responsável por gerenciar qual é a melhor forma de acesso ao dados, ou seja, como o processamento de consultas irá proceder. Por exemplo, se existem múltiplas unidades de processamento disponíveis, sejam CPUs ou *cores*, então o SGBD analisa se uma determinada consulta pode ser paralelizada para aumentar a vazão do processamento e consequentemente promover a diminuição no tempo de resposta. Portanto, o aumento da quantidade de recursos computacionais tem grande influência no desempenho do processamento de consulta. O provisionamento de

[1] Transaction Processing Performance Council (TPC): http://www.tpc.org

recursos computacionais pode atender aos diferentes padrões de carga de trabalho aos quais o SGBD está sujeito, incluindo picos de carga durante datas específicas, como, por exemplo, *black friday*, natal, páscoa e etc. Tendo impacto direto na qualidade de serviço e desempenho do SGBD, provendo maior capacidade de processamento e evitando o desperdício e ociosidade de recursos.

Para evitar tal desperdício e garantir qualidade de serviço do SGBD, entretanto, o provisionamento de recursos apresenta o desafio de definir a quantidade ideal de recursos necessária para atender uma determinada demanda, a fim de otimizar a utilização dos recursos existentes e diminuir custos relacionados à aquisição de novos recursos. Considerando ainda a dinamicidade dos ambientes nos quais os SGBD atuais estão inseridos, técnicas que realizam o provisionamento dinâmico de recursos se mostram mais eficientes, uma vez que as mudanças constantes de carga de trabalho são detectadas em tempo de execução e as decisões podem ser tomadas sob demanda.

1.1 Problema e Motivação

Segundo [Das, 2011], os SGBD formam um componente crítico na pilha de programas. Para minimizar os custos operacionais, os SGBDs precisam se adaptar de forma dinâmica às diferentes cargas de trabalho, utilizando de maneira mais eficiente os recursos computacionais disponíveis no *hardware*. Para isto é necessário o monitoramento das mudanças na carga de trabalho que possam ocorrer, definindo o momento correto para alocar os recursos computacionais.

A adaptação de forma dinâmica de um SGBD para atender uma determinada carga de trabalho, no entanto, pode elevar os custos com infraestrutura que podem ser causados pelo excesso ou falta de recursos computacionais. Ao realizar o provisionamento dinâmico de recursos, a abordagem adotada deve ser capaz de adicionar e remover os recursos, quando existir a necessidade real de provisionamento. Para entender a relação entre o provisionamento de recursos computacionais e o desempenho do SGBD, um modelo de abstração contribui para que o provisionamento atenda a demanda de acordo com a necessidade do SGBD, mantendo seu desempenho e garantindo o gerenciamento adequado dos recursos computacionais. Podemos afirmar que uma das grandes deficiências das abordagens atuais é a ausência de um modelo abstrato [Farias et al., 2014, Cecchet et al., 2011, Minhas et al., 2012, Ghanbari et al., 2007, Cong and Martin, 2015], que mostre a relação entre desempenho e recursos disponíveis, como suporte para o provisionamento. A modelagem de tal relação tem papel decisivo na eficiência do provisionamento [Shivam et al., 2007]. Além de melhorar o entendimento e facilitar o desenvolvimento de uma ferramenta de provisionamento, o uso de um modelo abstrato de dados torna possível simulações que podem atestar o comportamento e eficiência da técnica de provisionamento adotada antes da construção de um protótipo.

As limitações apresentadas acima, analisadas mais profundamente na Seção 2, motivam este trabalho, que apresenta um modelo de provisionamento de recursos computacionais para SGBD, baseado em Redes de Petri (RdP) Predicado/Transição (PrT).

1.2 Hipótese de Pesquisa

A memória *cache*[2] de um processador armazena os dados que são buscados mais frequentemente, por um período de tempo. O dado é inicialmente buscado na memória *cache* do processador, quando encontrado ocorre um *hit* de *cache*, caso contrário um *miss* [Patterson and Hennessy, 2007]. Em virtude do aumento dos *hits* de *cache*, é possível aproveitar os dados salvos na *cache* do processador, tornando a busca mais rápida. Como a memória *cache* possui diferentes níveis, os níveis inferiores são mais lentos, no caso de um *hit*, o tempo de busca pelo dado é menor comparado ao de um *miss*, em que o dado é buscado em níveis inferiores.

A hipótese desta dissertação é que utilizando o modelo abstrato de provisionamento de recursos computacionais para SGBD baseado em Redes de Petri, é possível encontrar um valor ótimo (ver Definição na Seção 4.1) de recursos computacionais, de forma que os *hits* de *cache* do processador aumentem e os *misses* de *cache* diminuam, melhorando significativamente o desempenho do SGBD.

1.3 Objetivos e Contribuições

O objetivo deste trabalho consiste em apresentar um modelo abstrato de provisionamento dinâmico de recursos computacionais para SGBD, baseado em Redes de Petri. O modelo abstrato foi elaborado, utilizando as Rede de Petri de alto nível Predicado/Transição, a qual apresenta condições específicas que modelam sistemas dinâmicos com um conjunto de indivíduos estruturados por um conjunto de funções, relações, propriedades dos indivíduos e modificações [Fernandez, 2002]. O modelo abstrato avalia o SGBD como uma "caixa preta", podendo ser utilizado em qualquer SGBD sem modificação de seu código fonte. O modelo abstrato analisa o uso de recursos pelo sistema operacional.

Com a finalidade de encontrar a quantidade ideal de recursos computacionais que atenda a demanda do SGBD, é apresentada a definição de valor ótimo de recursos computacionais. Para a PrT-PRO gerenciar os recursos, de forma que seja encontrado o valor ótimo, foi elaborado um algoritmo que direciona o provisionamento de recursos através da quantidade disponível e o uso do recurso pelo SGBD.

Construímos um protótipo com o objetivo de demonstrar a eficiência da PrT-PRO e os impactos gerados pela adição e remoção de recursos computacionais. Para analisar e atestar o comportamento do modelo de provisionamento apresentado, foram utilizados os *benchmarks*

[2]Dispositivo de acesso rápido interno em um sistema, em processadores a memória *L1* é mais rápida, seguida pela *L2* e *L3*. A busca pela informação acontece inicialmente na *L1* e depois *L2* e *L3*.

TPC-H [Council, 2015b] e TPC-DS [Council, 2015a], que simulam um ambiente de suporte à decisão.

Com a PrT-PRO, mostramos que o popular SGBD de código aberto PostgreSQL apresenta redução significativa no tempo de execução das consultas, uma vez que é estabelecida a quantidade ideal de recursos para atender a carga de trabalho que está sendo executada pelo SGBD. Além disso, evidencia a diferença de tempo de execução da carga de trabalho, quando há a falta de recursos ou quando são utilizados todos os recursos disponíveis no *hardware* que poderiam ser designados para outras tarefas. Analisamos também o impacto no desempenho do processamento de consulta, mudando as diferentes configurações possíveis da PrT-PRO e discutindo quais obtiveram melhor desempenho na execução da carga de trabalho, essas diferentes configurações da PrT-PRO são utilizadas como base para o monitoramento e provisionamento dos recursos utilizados durante os experimentos.

1.4 Estrutura da Dissertação

Esta dissertação está organizada da seguinte forma: O Capítulo 2 introduz os principais conceitos relacionados ao provisionamento de recursos computacionais, apresentando os trabalhos que tratam de provisionamento de recursos computacionais em nuvem e para SGBD. O Capítulo 3 apresenta conceitos relacionados à Redes de Petri e Redes de Petri de alto nível Predicado/Transição utilizada para desenvolvimento do nosso modelo abstrato de provisionamento de recursos para SGBD. O Capítulo 4 apresenta a nosso modelo de provisionamento. O Capítulo 5 descreve o ambiente experimental, as cargas de trabalho utilizadas, os experimentos realizados e os resultados alcançados. Por fim, o Capítulo 6 apresenta as considerações finais e trabalhos futuros.

Capítulo 2

Estado da Arte

Neste capítulo, introduzimos conceitos gerais do provisionamento de recursos computacionais para contextualizar este trabalho. Em seguida, abordamos de maneira mais específica uma visão do provisionamento de recursos computacionais em nuvem. Apresentamos as abordagens relacionadas ao provisionamento de recursos para SGBD e trabalhos que utilizam modelos abstratos para diferentes finalidades. Por fim, fazemos uma discussão dos assuntos abordados.

2.1 Provisionamento de Recursos Computacionais

O provisionamento de recursos computacionais busca garantir a alocação eficiente de recursos computacionais como, por exemplo, CPU (*Central Processing Unit*), armazenamento (memória RAM e disco rígido) e largura de banda de rede. Podemos considerar como alocação eficiente dos recursos, se não existem desperdícios de recursos que se encontram ociosos, ou quando não há falta de recursos para garantir um certo nível de qualidade de serviço.

O provisionamento é subdividido em duas técnicas [Reis, 2006]:

- Estático: os recursos são provisionados pelo administrador/programador de forma estática e alterada em longos períodos de tempo.
- Dinâmico: Os recursos são alocados, conforme a demanda em tempo de execução.

O provisionamento dinâmico pode ser utilizado sob duas abordagens:

- Reativo: uma abordagem orientada a recursos, com um conjunto de regras pré-definidas, baseada em regra-condição-ação. São adotados *thresholds* superiores e inferiores que indicam quando adicionar e remover recursos [Rodrigo, 2013].
- Proativo ou Preditivo: para o provisionamento de recursos, é realizado um reconhecimento nos padrões da carga de trabalho, antecipando situações negativas para o sistema, para estimar a quantidade de recursos necessária [Cecchet et al., 2011].

A escolha da métrica a ser observada é fator importante para um provisionamento eficiente. Através desta, é possível analisar o comportamento do sistema. Segundo [Rodrigo, 2013], uma métrica define qual é o objetivo do estudo realizado. Essas métricas podem avaliar, por exemplo, a carga de uso de instâncias de VM (*Virtual Machine*), a carga de uso de CPU, gasto de energia, memória entre outros.

2.1.1 Provisionamento de Recursos Computacionais em Nuvem

A computação em nuvem é um paradigma da computação que propõe a integração de diversos recursos computacionais, fornece processamento, infraestrutura e armazenamento de dados através da internet [Sousa et al., 2010]. Em computação em nuvem, o provisionamento de recursos computacionais pode ser caracterizado pelas operações realizadas para garantir a Qualidade de Serviço (QdS), baseado em Acordos de Níveis de Serviço (ANS), que fornecem informações de níveis de disponibilidade, funcionalidade, desempenho entre outros [Fernandez, 2002]. O provisionamento é usado para alocar de forma eficiente os recursos existentes, considerando que o superprovisionamento (i.e., excesso) pode desperdiçar recursos e o subprovisionamento (i.e., falta) não é capaz de oferecer o desempenho esperado.

O provisionamento de recursos em computação em nuvem pode ser feito através de adição e remoção de instâncias de VM, sendo definido por provisionamento horizontal. Ele também pode aumentar ou diminuir recursos como memória, CPU, rede e disco, classificado como provisionamento vertical.

Em [Schubert et al., 2011], é apresentado um modelo de provisionamento proativo de recursos para computação em nuvem que utiliza mecanismos de predição, baseado em redes neurais artificiais, para adicionar e remover recursos, buscando garantir os ANS. Na Figura 2.1, é apresentado o modelo baseado em redes neurais, em que são definidos requisitos de entrada e saída, bem como a arquitetura complementar utilizada.

O modelo apresentado pelos autores busca, por meio das entradas (taxas de requisições, tempo necessário para requisição ser processada, consumo de memória, uso de processamento e consumo de disco), definir a quantidade de recursos necessários para atender o ANS e as necessidades do sistema (quantidade de núcleos (*cores*)) de processamento, quantidade de memória, e disco). A saída é utilizada para a criação de uma nova VM, com as quantidades de recursos necessárias para atender a demanda do sistema avaliado.

Em [Rajiv, 2013], utiliza-se a abordagem reativa e preditiva para apresentar um modelo de provisionamento recursos para aplicações *multi-tier* com escrita intensiva. A abordagem reativa é utilizada para adicionar recursos. Para isso é realizada a leitura dos *logs* de *proxy*, os quais são categorizados em requisições estáticas (acesso às páginas que possuem textos e imagens) e dinâmicas (páginas geradas em tempo de processamento da requisição). Então, calcula-se o tempo de resposta de ambos, se o tempo de resposta dinâmico estiver acima do limite definido, provisiona-se uma nova VM para a camada web. Se o tempo de resposta estático

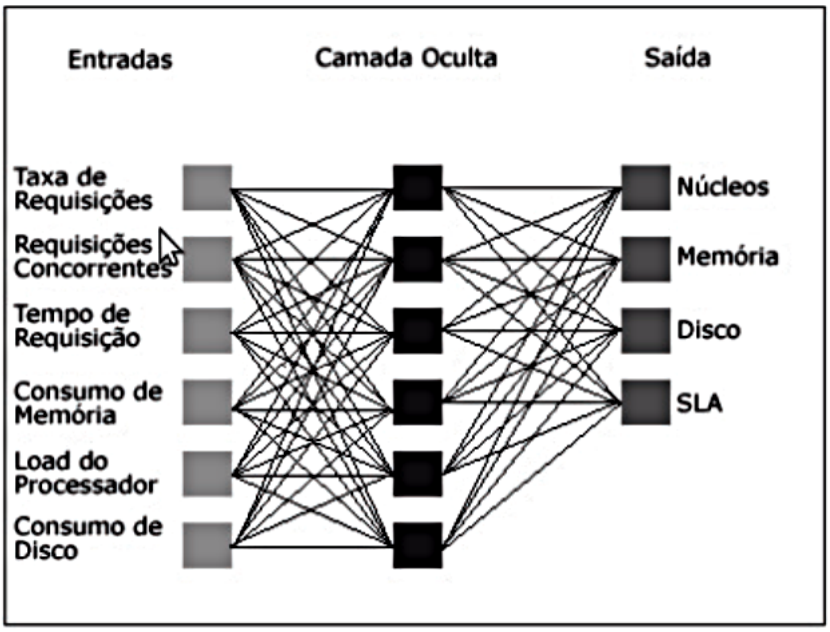


Figura 2.1: Rede neural artificial [Schubert et al., 2011]

estiver acima do limite, verifica-se o uso de CPU, quando alto uso de CPU pela camada web for identificado, provisiona-se uma nova VM para a mesma, caso contrário provisiona-se uma nova VM para a camada de banco de dados. A remoção dos recursos utiliza a abordagem preditiva, por meio de um modelo de regressão que prevê quantas instâncias *web* e de banco de dados são necessárias para atender a carga de trabalho. Assim, é verificada se a quantidade provisionada é maior que o necessário, caso a comparação da resposta seja positiva, os recursos em excesso são removidos.

No contexto de computação em nuvem, existem diferentes pesquisas que buscam realizar de forma eficiente o provisionamento de recursos computacionais para garantir a QdS atendendo os ANS. Em [Meng et al., 2010], os autores apresentam uma técnica de multiplexação de VM para tornar o provisionamento de recursos computacionais em nuvem eficiente, considerando a capacidade que uma VM necessita para atender o ANS, utilizam um algoritmo que combina as VM's com cargas de trabalho similares, sendo provisionadas em conjuntos para alcançar a utilização capacidade máxima de cada VM.

Em [Garg et al., 2011] os autores buscam melhorar o provisionamento de recursos computacionais para um *datacenter* que possui diferentes cargas de trabalho, o provisionamento de recursos é monitorado e realizado em intervalos regulares, buscando controlar quando aceitar uma nova aplicação (transacional ou lotes), baseado nos recursos ainda disponíveis, a demanda por recursos é monitorada para garantir a QdS contratada no ANS.

Em [Zhang et al., 2010] apresentam uma abordagem para provisionar recursos de forma rápida para *datacenter* virtualizados, utilizam *gosth* VM que são mantidas dentro das plataformas de computação separadas da internet, sua existência é invisível e não atendem pedidos realizados por clientes, quando necessárias, são ativadas e passam a receber tais requisições.

2.1.2 Provisionamento de Recursos Computacionais para SGBD

Segundo [Sousa et al., 2010], os SGBD são potenciais candidatos para implementação em nuvem, reduzindo o custo de armazenamento e melhorando o acesso ao mesmo. O desafio é o provisionamento de recursos para o SGBD, pois é necessário adicionar e remover recursos não perdendo o acesso aos dados, mantendo sua consistência e estando pronto a atender cargas de trabalho elevadas. Nesse contexto, há alguns trabalhos buscam criar um SGBD que possa ser elástico [Minhas et al., 2012] e apresentam soluções para provisionamento de recursos utilizando abordagens preditivas [Farias et al., 2014], trabalhos onde consideram tempo da replicação do banco de dados para realizar o provisionamento [Cecchet et al., 2011] e trabalhos que utilizam a abordagem preditiva de avaliação de desempenho do SGBD para adicionar e remover recursos [Cong and Martin, 2015].

Em [Farias et al., 2014], os autores apresentam uma proposta que utiliza uma estratégia para provisionamento de recursos para SGBD, os recursos são novas VMs. A estratégia é baseada em uma técnica de análise de séries temporais para realizar um processo de predição da carga de trabalho. Assim dividem a estratégia em dois módulos, sendo um módulo de predição, e outro de provisionamento conforme demonstrado na Figura 2.2.

O módulo de predição utiliza como métrica a quantidade de transações por unidade de tempo (hora, minuto ou segundo). Para realizar a predição, os autores utilizam o modelo de regressão $\epsilon-$SVR (*Support Vector Regression*) [Smola and olkopf, 2004] ou ARIMA (*Autoregressive Integrated Moving Average*) [Kongcharoen and Kruangpradit, 2013], cuja escolha depende do erro apresentado. As informações geradas pelo módulo de predição e as métricas serão utilizadas no segundo módulo, no qual será feito o cálculo da quantidade de recursos que serão adicionados ou removidos. Assim a estratégia apresentada pelos autores pretende encontrar a relação entre qualidade e quantidade de recursos que serão utilizados.

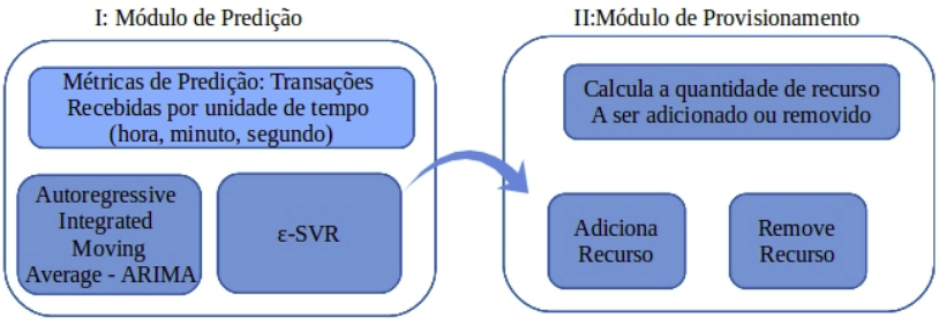


Figura 2.2: Módulos de provisionamento de recursos [Farias et al., 2014]

No trabalho apresentado em [Farias, 2016], o autor descreve um mecanismo de elasticidade que utiliza modelos de desempenho para adicionar e remover nós de um *cluster* NoSQL[1],

[1] Classe de banco de dados não-relacionais.

utilizando um provisionamento preditivo. Os modelos de desempenho são obtidos através da modelagem das características da carga de trabalho e do *cluster*, estimando o desempenho do serviço através de um conjunto de parâmetros referentes à carga de trabalho. Os parâmetros são relacionados à quantidade de requisições por segundo realizadas, a porcentagem de requisições para cada tipo de operação (leitura, atualização, inserção) e as violações de ANS. Nos resultados gerados pelos parâmetros, é realizada uma filtragem, para cada parâmetro é definido um conjunto de dados e um modelo preditivo.

Os modelos de predição que apresentam uma melhor capacidade preditiva são utilizados. Através das requisições que chegam ao sistema a cada um segundo, é verificada a necessidade de provisionamento, se detectada é calculada a nova quantidade de recursos necessária.

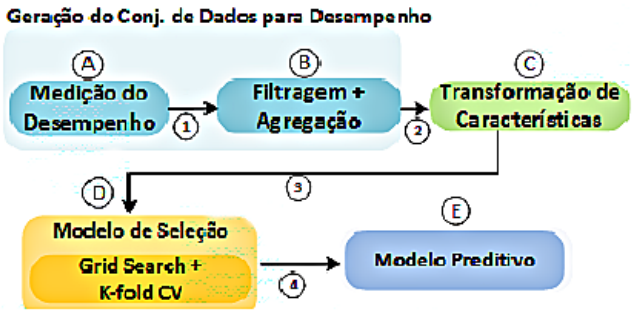


Figura 2.3: Etapas para modelagem de desempenho e provisionamento [Farias, 2016].

Na Figura 2.3, são apresentadas pelo autor as etapas de modelagem de desempenho e provisionamento de recursos. Em uma primeira etapa, o autor gera um conjunto de dados de desempenho, utilizando aprendizado de máquina. A abordagem de aprendizado de máquina utilizada necessita de um conjunto de n dígitos (conjunto de treinamento). Cada dígito possui uma categoria, o resultado é o modelo preditivo. A primeira etapa é subdividida em etapas de medição do desempenho, em que foram realizados testes para verificar os parâmetros que contribuem para a variação de desempenho e relacionar com o tamanho do *cluster*.

Na etapa de agregação e filtragem, os *logs* gerados nos testes da etapa de medição de desempenho são agregados de forma a criar um conjunto menor de dados. O objetivo dessa etapa é modelar o desempenho baseado em três métricas (tempo de resposta médio por segundo das consultas, vazão sendo as transições por um determinado de tempo e violação por segundo do ANS), sendo gerado para cada métrica um conjunto de dados.

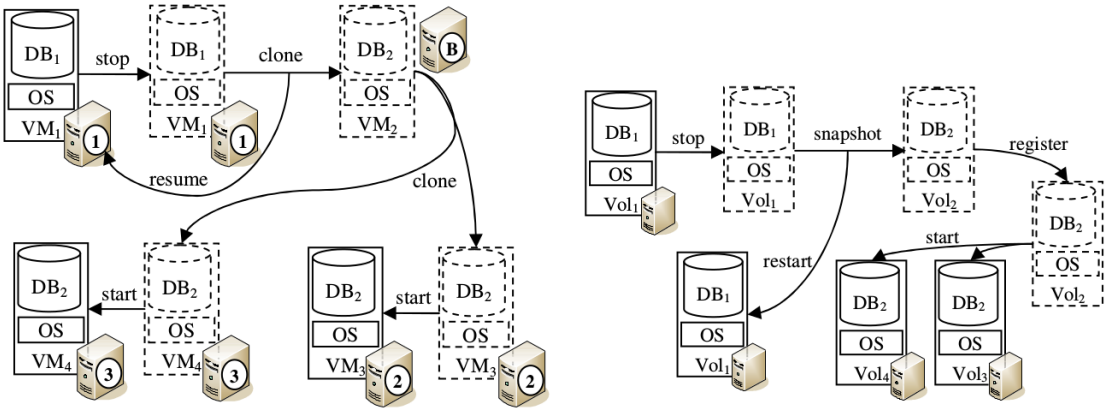
Após a primeira etapa, é realizada a transformação das características dos parâmetros. As características foram definidas em dois tipos de manipulação, sendo características lineares e quadráticas. Na terceira etapa, possuem os parâmetros necessários para um conjunto de n dígitos, cada um com sua categoria, resultando assim no modelo preditivo. Avaliam a acurácia de cada modelo gerado com validação cruzada *K-fold* [Refaeilzadeh et al., 2009].

Para realizar o provisionamento, o autor utiliza o modelo gerado pelas fases descritas da Figura 2.3, verificando se existe a necessidade de provisionamento. O modelo recebe as

características da carga de trabalho executada e define a alocação ótima dos recursos. Para alocação de recursos, é considerado o ANS, buscando maximizar o lucro do provedor de serviços.

Em [Cecchet et al., 2011], é apresentada uma ferramenta denominada *Dolly*, que baseia seu provisionamento de recursos em *snapshots* de VM como apoio para a replicação do estado do banco de dados. Os autores apresentam uma abordagem de provisionamento dinâmico preditivo, levando em consideração a necessidade de replicação do banco de dados. *Dolly* realiza a clonagem de uma VM existente, com seu ambiente de funcionamento e o banco de dados com todas as suas definições de configurações. *Dolly* utiliza *snapshots* de VM para a fase de *backup* e utiliza clonagem de VM para a nova réplica. Os autores apresentam o provisionamento de recursos utilizando *Dolly* em nuvens privadas e públicas (utilizando Amazon EC2 [sit, 2015a]), conforme as Figuras 2.4(a) e 2.4(b).

A Figura 2.4(a) exemplifica a geração de duas novas réplicas em uma nuvem privada, A máquina 1 é interrompida para clonagem de *backup* na máquina B, em seguida são criadas duas novas réplicas clonadas da máquina B. A Figura 2.4(b) apresenta a criação de duas VM em uma nuvem pública (Amazon EC2) que oferece um serviço chamado de *Amazon Elastic Block Storage* (EBS)[2]. A máquina inicial é interrompida para fazer um *snapshots*, esse *snapshot* é registrado no serviço EBS do EC2 da Amazon. Em seguida, os dois clones são criados e sincronizados.



(a) Réplica spawning em uma nuvem privada [Cecchet et al., 2011].

(b) Réplica spawning em uma nuvem pública [Cecchet et al., 2011].

Figura 2.4: Replicação *Dolly* [Cecchet et al., 2011] .

Dolly possui um motor de provisionamento que indica a carga de trabalho, observando as mudanças na carga de trabalho do sistema para realizar o provisionamento de recursos. Para dispor de certa capacidade em um determinado prazo, são agendadas ações de capacidade de provisionamento baseado no tempo de replicação do estado do banco de dados, este tempo de replicação é utilizado na predição. A Figura 2.5 apresenta um exemplo dos autores, que

[2]Fornece volumes de armazenamento em níveis de bloco persistentes, cada volume EBS é replicado automaticamente na sua zona de disponibilidade [sit, 2015a].

demonstra a janela de provisionamento executada por *Dolly*, sendo d_1, d_2, d_3 os prazos em que são marcadas as decisões de provisionamento e as mudanças na demanda da capacidade de previsão.

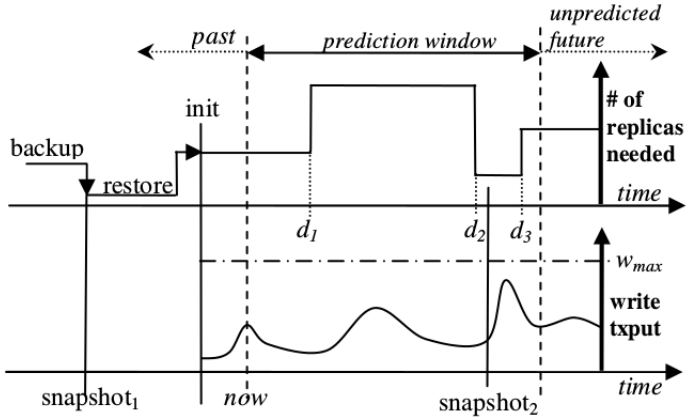


Figura 2.5: Exemplo de predição de *Dolly* [Cecchet et al., 2011]

Em [Minhas et al., 2012], os autores procuram utilizar a flexibilidade, oferecida pelo particionamento de banco de dados para fornecer o provisionamento de recursos, por meio de mais nós do banco. Para isso, propõem iniciar com poucas máquinas servidoras, gerenciando todas as partições do banco de dados e, quando necessário, o provisionamento adiciona novas máquinas servidoras e redistribuir as partições entre elas. Os autores têm como objetivo principal encontrar os problemas em fornecer provisionamento baseado em partições.

Para os experimentos, os autores utilizaram o SGBD VoltDB[3] [sit, 2015b]. Com o objetivo de crescimento dinâmico no número de nós no *cluster*, os autores utilizam uma versão modificada do mecanismo *rejoin* do VoltDB. São criados nós denominados de nó *scale-out*, que inicialmente não possuem partições do banco de dados. Os nós de *scale-out* não armazenam dados ou processam transações, quando é necessário o provisionamento estes nós recebem o mapeamento das partições do banco de dados. O sistema é iniciado com um número fixo de nós *scale-out*.

Desse modo, os autores fazem o VoltDB tratar os nós de *scale-out* como se tivessem falhado e que voltam para o estado de ativo. No estado ativo, os nós de *scale-out* necessitam de uma cópia de dados de um nó já existente. O VoltDB implementa um mecanismo de recuperação depois que um nó retornou ao *cluster*, os dados são copiados de uma partição de origem para uma partição de destino. Para o provisionamento, os autores fizeram alterações no mecanismo de recuperação do VoltDB. Quando o nó de *scale-out* retorna ao *cluster*, será executada a recuperação de todas as partições mapeadas para ele.

A Figura 2.6 mostra o mapeamento de partições de hospedeiros em diferentes pontos, juntamente com o mapeamento das partições de núcleos dentro do hospedeiro, que são gerenciados pelo sistema operacional.

[3] VoltDB é um SGBD distribuído

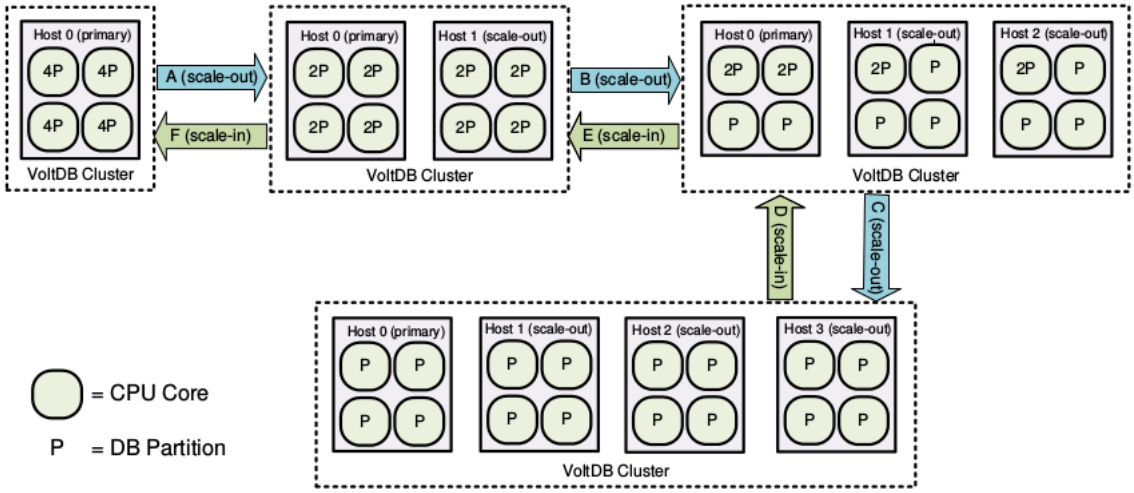


Figura 2.6: Ilustração de *scale-out* [Minhas et al., 2012]

Para escalar um novo nó do banco de dados, os autores monitoram a utilização de CPU pelo sistema. Quando o uso chegar a 80% de utilização de CPU um novo nó é ativado, quando apresentar 20% de uso CPU, um nó é desativado. Os autores não definem exatamente uma técnica de provisionamento, mas como criar um SGBD elástico.

Em [Cong and Martin, 2015], os autores apresentam uma abordagem de provisionamento preditiva, que monitora o desempenho do sistema e define o número núcleos (*cores*) de CPU necessários para atender a carga de trabalho. Para avaliar qual a quantidade de CPU necessária, é utilizado a lógica *fuzzy* assim é possível determinar a quantidade de recursos, conforme a demanda do sistema. Para o desenvolvimento do controlador de provisionamento de recursos, os autores definem que a curva de desempenho de carga de trabalho do SGBD é côncava, com o aumento de *cores* de CPU, a curva diminui a sua derivada[4], definindo o problema como encontrar o ponto estacionário da curva. Para definir a quantidade de *cores* os autores apresentam $f(x)$ como o desempenho do SGBD, quando o $f(x)$ é negativo são adicionados mais *cores* de CPU.

Na Figura 2.7, pode ser vista a arquitetura do controlador apresentanda por [Cong and Martin, 2015], o controlador *fuzzy* possui duas entradas sendo estas $dy(k)$ quantidade de núcleos e $du(k)$ o rendimento do sistema. No processo de *fuzzification*, são transformadas as entradas numéricas em conjuntos *fuzzy* que quantificam as regras (*Fuzzy Rules*). O *Inference Mechanism* define quais regras são pertinentes, em seguida ocorre a *defuzzification* que determina o valor numérico representante da quantidade de *cores* de CPU necessárias. Quando a carga de trabalho modifica, o controlador inicia novamente para redefinir o número ideal de *cores* de CPU.

[4] Dada função de uma variável, sua taxa de variação em relação à variável em determinado ponto, definida como o limite do quociente entre o incremento da função e o da variável quando o acréscimo da variável, tende a zero.

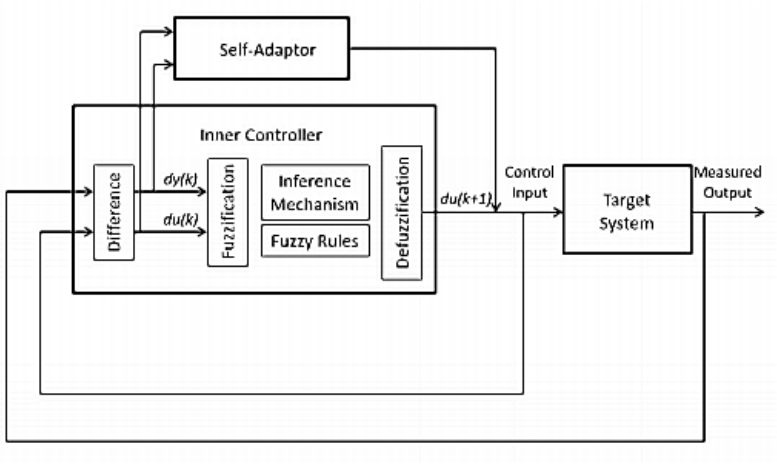


Figura 2.7: Arquitetura do controlador [Cong and Martin, 2015].

2.2 Modelagem

Segundo [T. J., 1974] *"A simulação implica na modelagem de um processo ou sistema, de tal forma que o modelo imite as respostas do sistema real numa sucessão de eventos que ocorrem ao longo do tempo"* . Além de uma formalização do provisionamento de recursos computacionais para os SGBD, o modelo pode apresentar todas as ações ocorridas no sistema modelado. Na modelagem, as variáveis simuladas podem apresentar o mesmo comportamento dinâmico de um sistema real.

Considerando abordagens que utilizam modelos para representar o desempenho de banco de dados, podemos citar a Database State Machine (DSM) [Meira et al., 2014]. Os autores apresentam uma máquina de estados para SGBD que representa os diferentes estados de desempenho no processamento de transações concorrentes. O modelo é composto por cinco estados: *WarmUp*, *Steady*, *Under Pressure*, *Stress*, *Thrashing* (ver Figura 2.8). O objetivo deste trabalho é realizar teste de estresse baseado em modelos para apontar possíveis defeitos do sistema testado.

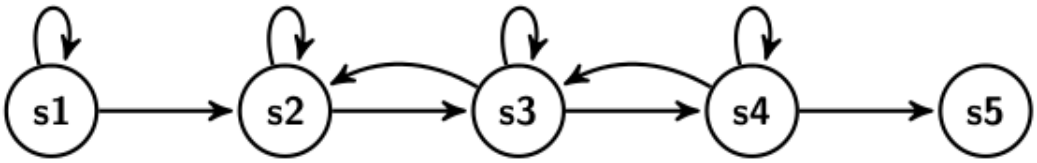


Figura 2.8: Máquina de estados de banco de dados [Meira et al., 2014].

A máquina de estados é utilizada para relacionar os diferentes estados de desempenho do SGBD com crescentes cargas de trabalho e defeitos relacionados. O estado *S1* (*Warmup*) considera o processo de inicialização realizado pelo SGBD. O estado *S2* (*Steady*) é o estado estável, no qual o SGBD é capaz de lidar com todas as transações requisitadas. O estado *S3* (*Under Pressure*) representa o SGBD em seu limite de performance. Nesse estado, o sistema ainda é capaz de lidar com a mesma quantidade de transações do estado anterior, mas não é

capaz de escalar desempenho, apresentando falta de recursos computacionais. No estado *S4* (*Stress*) o SGBD se encontra além de seus limites de desempenho e já não é capaz de manter desempenho anterior. O estado *S5* (*Trashing*) é quando o SGBD utiliza uma grande quantidade de recursos para uma quantidade mínima de trabalho, sendo incapaz de lidar com qualquer nova transação. As transições entre os estados são baseadas na variação de diferentes parâmetros de desempenho.

No trabalho apresentado em [Coulden et al., 2013], o autor utiliza uma extensão de RdP com o nome de *Queueing Petri Nets* (QPNs), que combina Redes de Fila com RdP para modelar o desempenho de banco de dados. Segundo os autores, o desempenho de sistemas de banco de dados é influenciado por funcionalidades complexas e interdependentes (transação, gerenciamento de *cache* de banco de dados, contenção de disco, concorrência, contenção de bloqueios). O modelo apresentado pelos autores pode ser visto na Figura 2.9. Os clientes são

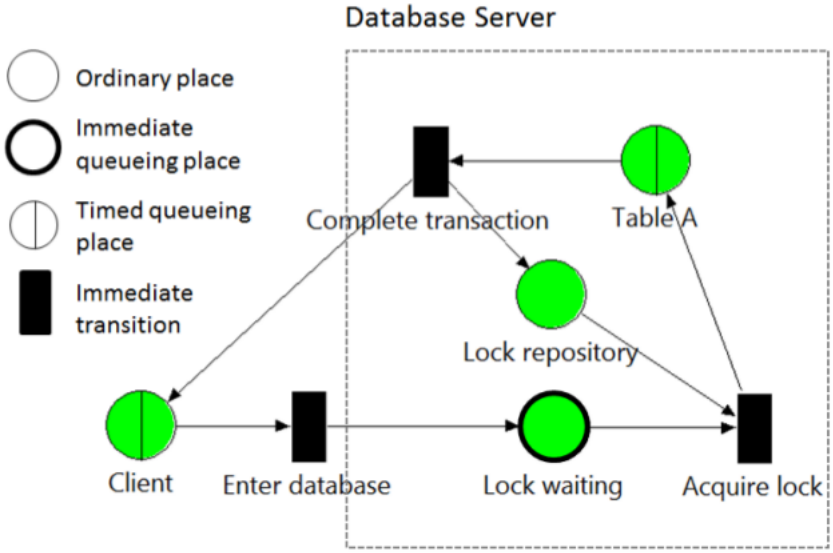


Figura 2.9: *Queueing Petri net model* [Coulden et al., 2013].

representados como um lugar na fila, esse lugar possui duas cores que representam um cliente com diferentes tipos de transação. O cliente envia transações para o servidor de banco de dados, em seguida a transação entra em bloqueio, esperando no lugar denominado tabela para após ser liberada. O lugar que corresponde ao bloqueio de espera é um lugar de fila FIFO, garantindo que as transações sejam atendidas na ordem do sistema.

2.3 Discussão

Os trabalhos relacionados ao provisionamento de recursos para SGBD apresentam diferentes estratégias para realizar a adição e remoção de recursos computacionais. Na maioria dos trabalhos a abordagem utilizada trata-se de provisionamento dinâmico proativo/preditivo.

Os trabalhos relacionados que fazem uso de modelagem, não tratam o provisionamento de recursos, mas avaliação e teste de desempenho do SGBD.

No trabalho [Minhas et al., 2012], apresenta uma estratégia utilizando as partições do banco de dados para provisionar recursos. Nos experimentos realizados, é monitorada a utilização de processamento pelo banco de dados, mas não é definido um provisionador. Os trabalhos [Farias et al., 2014], [Ghanbari et al., 2007] e [Cecchet et al., 2011] apresentam abordagens de provisionamentos de predição que diferem da abordagem adotada neste trabalho, que utiliza a abordagem dinâmica reativa. No trabalho [Cong and Martin, 2015], é apresentada uma abordagem proativa que determina qual a quantidade de núcleos de CPU é necessária para atender uma carga de trabalho estável mantendo o desempenho do SGBD. Essa abordagem difere deste trabalho por ser proativa, embora busque a alocação de recursos computacionais para manter/melhorar o desempenho do SGBD.

Não se encontra nos trabalhos citados acima, uma modelagem abstrata detalhada do provisionamento de recursos. A modelagem se torna necessária para apresentar o comportamento do SGBD, quando é realizado o provisionamento, facilitando a análise de todo o processo necessário para definir quando será realizado o provisionamento. Em [Meira et al., 2014], os autores modelam através de uma máquina de estado o comportamento do SGBD em situações de mudanças na carga de trabalho e, em [Coulden et al., 2013], os autores modelam o desempenho de um banco de dados. Ambos, no entanto, focam em análise e teste desempenho, em vez de provisionamento.

Para melhor visualização e entendimento das vantagens e limitações, a Tabela 2.1 resume as características das abordagens apresentadas. Consideramos quatro características de cada trabalho explicitado, sendo elas: objetivo do trabalho, abordagem do provisionamento, modelo de abstração e o método de Provisionamento.

O próximo capítulo apresenta conceitos e definições importantes sobre Redes de Petri, utilizada para a criação do modelo abstrato de provisionamento de recursos computacionais para SGBD. Descreve as Redes de Petri denominadas de baixo nível e de alto nível, enfatizando as Redes de Petri Predicado/Transição, utilizada para a criação do modelo abstrato de provisionamento de recursos computacionais para SGBD apresentada nesse trabalho.

Tabela 2.1: Análise Comparativa entre os Trabalhos Relacionados

Trabalho	Objetivo do Trabalho	Abordagem	Modelo de abstração	Método Utilizado	Carga de Trabalho
[Cecchet et al., 2011]	Provisionamento	Dinâmico Proativo	não se aplica	Snapshots de Máquinas Virtuais	Transacional
[Coulden et al., 2013]	Modelagem	não se aplica	*Queueing Petri Nets*	*	Transacional
[Farias et al., 2014]	Provisionamento	Dinâmico Proativa	não se aplica	Virtualização	**
[Meira et al., 2014]	Modelagem	não se aplica	Máquina de estados	Virtualização	Transacional
[Minhas et al., 2012]	Provisionamento	não definida	não se aplica	Partições do banco de dados	Transacional
[Farias, 2016]	Provisionamento	Dinâmico Proativa	não se aplica	Nós	NoSQL
[Cong and Martin, 2015]	Provisionamento	Dinâmico Proativa	não se aplica	Núcleos de CPU	Transacional
Nossa proposta	Provisionamento	Dinâmico Reativo	Redes de Petri	Núcleos de CPU	Leitura

*O autor não deixa claro, qual método Utilizado. **O autor não deixa claro a carga de trabalho.

Capítulo 3

Redes de Petri

As Redes de Petri (RdPs) foram inicialmente propostas por Carl Adam Petri no ano de 1962, quando apresentou sua tese de doutorado na Faculdade de Matemática e Física da Universidade Técnica de *Darmstadt* na Alemanha. O objetivo de Petri foi apresentar um grafo bipartido com estados associados para estudar a comunicação entre autômatos[1] [Murata, 1989].

As RdPs são amplamente utilizadas nas mais diversas áreas na avaliação de desempenho, especificação e projetos de *softwares*, em protocolos de comunicação e diagnóstico de falhas [Maciel et al., 1996]. Segundo [Murata, 1989], as RdPs são formadas por dois tipos de componentes, os quais são denominados de lugares (elemento passivo) e transições (elemento ativo). O primeiro componente demonstra as variáveis de estados e o segundo, os eventos do sistema. Os lugares e as transições são ligados por arcos, desconectadas entre si. Os arcos definem a ordem da execução da rede. Um lugar tem suas informações alteradas somente na execução de uma ação. Os lugares são representados por círculos, as transições por barras ou traços, e os arcos por setas, ver Figura 3.1.

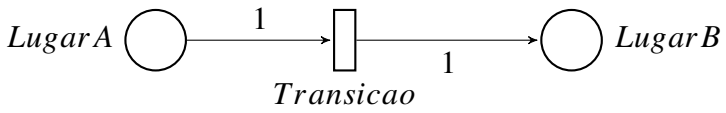


Figura 3.1: Elementos de uma Rede de Petri

Para a habilitação do disparo de uma transição, é necessário atender uma condição. Essa condição recebe através dos arcos as marcas (*fichas*). As marcas representam as informações presentes nos lugares. O número de marcas em um lugar corresponde ao seu comportamento em um determinado momento. Na Figura 3.2, pode ser observada a habilitação de uma transição. O conjunto de marcas e a distribuição das mesmas em uma Rede de Petri representam o estado do modelo.

Após a transição disparar, o estado inicial dos lugares que corresponde à ação é alterado. Os valores que acompanham os arcos definem a quantidade de marcas que serão utilizadas

[1] Um autômato é uma máquina que se opera de maneira automática.

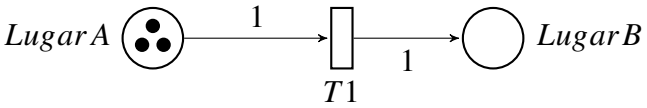
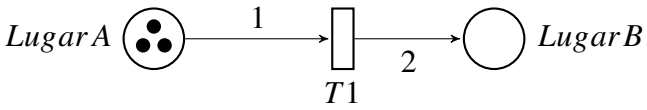
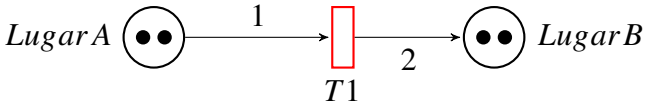


Figura 3.2: Marcação de lugares

para a habilitação da transição e a quantidade de marcas que serão transferidas para o lugar correspondente. A quantidade de marcas transferidas para os lugares não são necessariamente iguais às marcas utilizadas para o disparo das transições [Cardoso and Valette, 1997]. Nas Figuras 3.3(a) e 3.3(b), pode ser observado o disparo de uma transição. Do $LugarA$ para a transição $T1$ é repassada uma marca, e da transição $T1$ para o $LugarB$, duas marcas.



(a) RdP antes do Disparo



(b) RdP depois do Disparo

Figura 3.3: Exemplo disparo de uma Rede de Petri

A Rede de Petri pode evoluir de diferentes formas de iteração ou roteamento, sendo: sequencial, se uma tarefa é executada após a outra; concorrentes, quando as transições concorrem por um mesmo lugar; paralela, se os eventos ocorrem de forma simultânea sem apresentar dependências nesse caso, podem escolher entre duas ou mais tarefas e a mesma tarefa pode ser executada várias vezes [Rodovalho, 2012].

Formalmente uma RdP é uma quádrupla $R =< P, T, Pre, Post >$ [Murata, 1989], onde:

- P é um conjunto finito de lugares de dimensão n;
- T é um conjunto finito de transições de dimensão m;
- $Pre : PXT \longrightarrow \mathbb{N}$ é a aplicação de entrada (lugares precedentes ou incidência anterior), com $\mathbb{N}$ sendo um conjunto de números naturais;
- $Post : PXT \longrightarrow \mathbb{N}$ é a aplicação de saída (lugares seguintes ou incidência posterior), com $\mathbb{N}$ sendo um conjunto de números naturais;

Por exemplo, considere a quádrupla $R =< P, T, Pre, Post >$, com $P = \{p_1, p_2, p_3\}, T = \{a, b, c, d\}$ e os valores da aplicação de entrada dado por, $Pre(p_2, c) = 3, Pre(p_1, b) = Pre(p_2, a) = Pre(p_3, d) = 1$, e os valores de aplicação de saída sendo, $Post(p_2, d) = 3$ e $Post(p_1, a) = Post(p_2, b) = Post(p_3, c) = 1$ é uma Rede de Petri [T. J., 1974] (Figura 3.4).

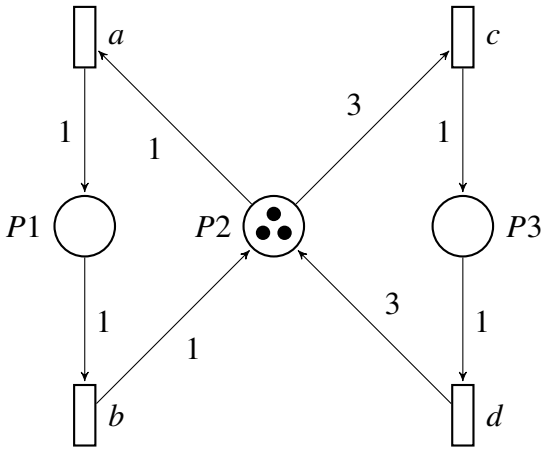


Figura 3.4: Exemplo RdP pela definição

As RdPs permitem a modelagem, análise e simulação de diversos sistemas, em diferentes áreas e podem ser utilizadas para geração de modelos e/ou ferramentas computacionais. As RdPs podem ser classificadas com base em sua complexidade: as RdPs de baixo nível, são aquelas nas quais as marcas depositadas nos lugares possuem um significado, quando relacionadas à estrutura da rede que estão associadas [Rodovalho, 2012]; as RdPs de alto nível que são descritas na seção seguinte.

3.1 Redes de Petri de Alto nível

As RdPs de alto nível derivam das RdPs de baixo nível e surgiram da necessidade de se utilizar condições que determinem o fluxo de controle e temporização [Penha et al., 2004]. As marcas representam informações relevantes sobre o sistema modelado e podem ser de diferentes tipos, cores e valores. Entre as RdPs de alto nível estão:

- Redes de Petri Temporais: nesta rede, a habilitação de uma transição está relacionada com um tempo, o qual descreve o tempo mínimo em que a transição deve esperar para ser disparada [Salvi, 2009].

- Redes de Petri Coloridas: As marcas neste tipo de rede possuem cores, os conjuntos de cores representam tipos de dados. As Redes de Petri Coloridas representam através de marcas distintas, diferentes processos ou recursos de um sistema, o que reduz o tamanho do modelo [Borges, 2008];

- Redes de Petri Contínuas: as marcas nesta rede são números reais. A transição é disparada em um fluxo contínuo. As marcas são transferidas respeitando uma velocidade de disparo [Rodovalho, 2012];

- Redes de Petri Híbridas: esta rede se originou na combinação entre Redes de Petri Contínuas e Temporizadas [Rodovalho, 2012].

- Rede de Petri a Objetos: nesta rede, são adicionados conceitos relacionados à orientação a objetos. Os lugares são as classes, as transições, as operações realizadas e as marcas são os objetos [Rodrigues, 2004];

- Redes de Petri Predicado/Transição: nesta rede as marcas podem ser informações diferentes do sistema modelado. A cada transição é associado um predicado de primeira ordem, que para habilitar a transição, necessariamente deve ser atendido, no disparo desta transição são associados ações a serem realizadas.

Em [Xu et al., 2002], os autores utilizam uma RdP de alto nível para a modelagem e análise de comportamentos de sistemas multi-agentes, compostos por múltiplos agentes com comportamento autônomo, os quais interagem entre os outros agentes do mesmo sistema. No trabalho [Aloini et al., 2012], utilizam as RdP para a modelagem de fatores de risco em projetos ERP (*Enterprise Resource Planning*). Em [Rodovalho, 2012], o autor utiliza RdP para modelagem de comportamentos biológicos, mais especificamente as PrT diferenciais[2] que facilitam a modelagem de processos biomatemáticos. Em [Perkusich et al., 1999], os autores apresentam um método para a modelagem de Banco de Dados em Tempo-real, utilizando Redes de Petri a objetos.

Na seção subsequente, serão apresentados conceitos relacionados à RdP de alto nível, denominadas Redes de Petri Predicado/Transição (PrT) [Genrich, 1987], utilizada, neste trabalho, como modelo de alto nível para provisionamento de recursos computacionais para SGBD. A PrT carrega em suas marcas informações do sistema modelado e permite criar condições para a habilitação de uma transição através de predicados de primeira ordem. Desse modo, é modelado o comportamento do provisionamento de recursos computacionais para SGBD. Na marcação da PrT, são repassadas informações de quantidade de uso de recursos e recursos disponíveis. Nas condições das transições, é avaliado a necessidade de adicionar e remover recursos e a disponibilidade para tais ações, não prejudicando o desempenho do sistema.

3.2 Redes de Petri Predicado/Transição

As PrT modelam sistemas dinâmicos com um conjunto de indivíduos estruturados por um conjunto de funções e relações, podendo modelar as propriedades dos indivíduos [Barros, 1996]. As PrT permitem estudar melhor as propriedades estruturais e comportamentais do sistema.

Formalmente a PrT marcada é uma tripla $N_{pt} =< R, A, M_0 >$ sendo [Genrich, 1987]:

- R é uma Rede de Petri definida pela quádrupla $R =< P, T, Pre, Post >$.

[2]A partir das PrT, foram propostas as PrT diferenciais que combinam eventos discretos e contínuos. Nas PrT diferenciais as variáveis associadas às fichas podem ser modificadas continuamente. Aos lugares são associados equações diferencias que descrevem a evolução das variáveis contínuas. As transições continuam possuindo a condição para a habilitação e as ações a serem realizadas.

- $A = (X, A_x, A_c, A_a)$ é uma quádrupla onde:
  - X é um conjunto de variáveis formais;
  - A_x é uma aplicação que associa a cada arco um vetor de variáveis de X;
  - A_c é uma aplicação que associa uma condição para cada transição, formando um predicado que utiliza as variáveis de X;
  - A_aé uma aplicação que associa uma ação para cada transição que afeta o valor das variáveis de X;
- M_0 é a marcação inicial, associada a cada lugar p de P uma ou n-uplas de variáveis;

Os lugares em uma PrT são chamados de predicados, as marcas contidas no lugares definem seu estado atual. Nas transições, são associadas condições que são escritas, utilizando as variáveis dos arcos de incidência anterior. As condições das transições precisam respeitar a lógica de predicados de primeira ordem. Os arcos de incidência posterior e anterior possuem rótulos que representantes das variáveis que são transferidas de um lugar para uma transição [Cantú, 1990].

As variáveis dos arcos associadas à marcação dos lugares definem a habilitação de uma transição. Em cada transição, é associada uma ação que define as variáveis que serão transferidas nos arcos de incidência posterior.

Após um transição disparar, a ação pode modificar e criar novas variáveis que serão repassadas para o lugar correspondente através do arco de incidência posterior. A Figura 3.5 mostra o disparo de uma transição em uma PrT. Pela definição, pode-se identificar:

- X sendo: d, r, q, v, sendo, d=4, r=3, q=6 e v=3 ou 5;
- Os vetores associados ao arcos são $A_{x1}(P_1, t_1) = < v >$, $A_{x2}(P_2, t_1) = < r, q >$ e $A_{x3}(P_3, t_1) =< d, q >$;
- $A_c = (v + r > q)$;
- $A_a = (d = 4)$;
- $M_0 = M(p_1) = 2$, $M(p_2) = 1$ e $M(p_3) = 0$

[Rodovalho, 2012].

Para habilitar a transição $T1$ da Figura 3.5, a marcação de $P1 < 5 >$ com a marcação de $P2 < 3, 6 >$ atendem a condição $(v + r > q)$ $(5 + 3 > 6)$. A ação adiciona $d = 4$, repassando $< d, q >$ $(4, 6)$ para $P3$. A marca 3 presente em $P1$ não atende a condição da transição.

Cada variável existente nessa Rede de Petri carrega a identidade de seus elementos, bem como a relação dinâmica entre os mesmos. Para o disparo da transição de uma Rede de Petri Predicado/Transição ocorrer, as variáveis da fórmula lógica presente na mesma devem ser substituídas por valores que estejam presentes na marcação da Prt e o predicado associado deve ser validado [Cantú, 1990].

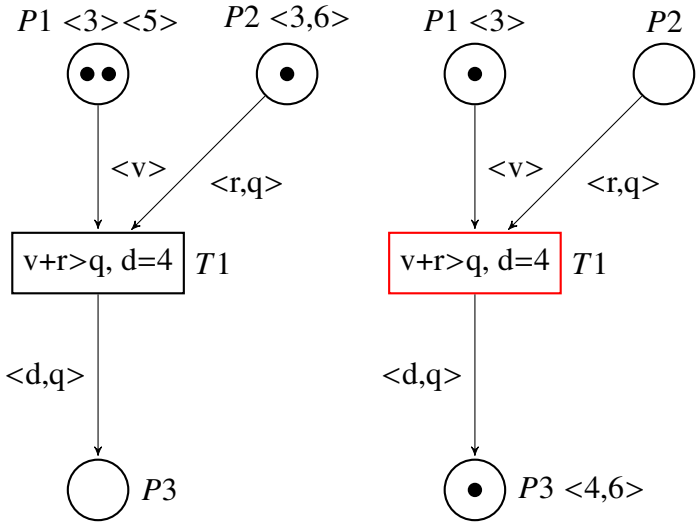


Figura 3.5: Disparo de uma Rede Predicado/Transição [Rodovalho, 2012]

3.3 Discussão

As RdPs permitem a compreensão e representação matemática de variados sistemas, através da modelagem. Segundo [Murata, 1989], é possível configurar equações de estados, algébricas e modelos matemáticos referentes ao comportamento do sistema. As marcas de uma RdP simulam as atividades simultâneas e dinâmicas de um sistema. Nas RdPs de alto nível, as marcas da Rede de Petri representam qualquer informação referente ao sistema modelado. Os lugares representam os estados de um sistema e as transições, as ações realizadas.

As características das RdPs de alto nível contribuem para a criação de um modelo de provisionamento de recursos. Utilizando as RdPs, é possível: monitorar a quantidade de recursos disponíveis simultaneamente com a avaliação do uso de recursos pelo SGBD; definir ações de provisionamento a partir da habilitação das transições, adicionando as restrições adequadas para cada ação esperada do provisionamento de recursos computacionais, possibilitando, através da marcação da RdP de alto nível, demonstrar os estados do sistema avaliado, baseado na informação contida nas marcas, facilitando a compreensão da evolução do provisionamento de recursos de forma que atenda a demanda do SGBD.

Neste capítulo, foram apresentados as definições e conceitos relacionados a Redes de Petri, Redes de Petri de alto nível e Redes de Petri Predicado/Transição, contextualizando as suas características, que são importantes para a criação da RdP utilizada neste trabalho.

O próximo capítulo apresenta a PrT de provisionamento de recursos computacionais para SGBD (PRT-PRO), os testes realizados no modelo abstrato e a descrição da criação do protótipo utilizado nos experimentos.

Capítulo 4

Provisionamento de Recursos Computacionais baseado em Redes de Petri

Neste capítulo, apresentamos a definição de valor ótimo, a PrT-PRO para SGBD, os testes realizados com objetivo de verificar o comportamento da rede e detalhes da implementação do protótipo utilizado nos experimentos. A finalidade de construir um modelo de alto nível baseado em Redes de Petri Predicado/Transição para provisionamento de recursos computacionais (PrT-PRO) para SGBD é modelar e/ou entender o comportamento do provisionamento de recursos e construir através da PrT-PRO um gerenciador de adição e remoção de recursos. A técnica de provisionamento adotada foi dinâmica com abordagem reativa, que se baseia em regra-condição-ação através de regras pré-definidas por uma métrica do sistema.

O protótipo desenvolvido da PrT-PRO para SGBD foi utilizado para melhorar o gerenciamento de recursos por parte do mesmo, assegurando que não existiria a falta ou ociosidade de recursos. A PrT-PRO para SGBD busca encontrar o valor ótimo de recursos que atenda uma certa demanda do sistema.

4.1 Definições

O valor ótimo de recursos computacionais (ver Definição 4.1) consiste na quantidade de recursos necessária para atender uma determinada carga de trabalho. Para a definição de valor ótimo, tomemos como exemplo de recurso a ser provisionado, o número de *cores* de uma CPU (*Central Processing Unit*).

Definimos como valor ótimo:

$$\forall\, w\ \exists\, r | (th_{min} < u < th_{max}) \wedge d(r) \geq d(rd) \tag{4.1}$$

onde:

- w é a carga de trabalho;
- r : recurso ótimo;
- rd é o total de recursos disponíveis;
- u utilização de recursos em porcentagem (%);
- th_{min} o *threshold* mínimo (%);
- th_{max} o *threshold* máximo (%);
- *d(x)* desempenho do SGBD em função do número de recursos usados, onde x assume r ou rd;

Para toda carga de trabalho, existe um valor ótimo de recursos, tal que o uso de recurso esteja entre o *threshold* mínimo e máximo e o desempenho do SGBD seja igual ou maior ao desempenho, utilizando todos os recursos disponíveis no *hardware*. Abaixo é apresentado o algoritmo para o cálculo do valor ótimo.

Algoritmo 1: Valor ótimo de recursos (cores)

Entrada: $r, u, rd, th_{min}, th_{max}$
Saída: Valor ótimo de cores r

```
1 início
2     enquanto r ≤ rd faça
3         se th_min < u < th_max então
4             retorna r
5         fim
6         r ← r + 1
7     fim
8 fim
```

O Algoritmo 1 apresenta a lógica utilizada pela PrT de provisionamento de recursos computacionais (PrT-PRO) para definir o valor ótimo de recursos que atenda a demanda do SGBD. Essa lógica é utilizada para gerenciar os recursos, provisionando de acordo com a quantidade de recursos disponíveis no *hardware*. A PrT-PRO monitora continuamente o sistema, garantindo que o valor ótimo de recursos atenda a demanda apresentada, não excedendo a quantidade de recursos disponível e/ou removendo mais que o necessário.

4.2 Modelagem da Rede de Petri Predicado/Transição

Nesta seção, definiremos a PrT-PRO para SGBD com seus componentes, sendo cinco lugares e oito transições. Os lugares são divididos em lugares representantes dos estados do SGBD, baseados no uso dos recursos computacionais e lugares auxiliares que possuem

informações sobre quantidade de recursos os quais podem ser provisionados e quanto foi utilizado dos recursos já provisionados. Os lugares capazes de representar os estados são:

- *Idle*: Quando o SGBD apresenta baixa utilização de recursos computacionais. Neste estado, a validação é verificar se existem recursos em excesso, em caso afirmativo, a ação a ser executada é a remoção;
- *Stable*: O SGBD se encontra estável, apenas o monitoramento é executado;
- *Overload*: O SGBD está sobrecarregado, quando entra neste estado, a ação a ser executada é a adição de recursos.

Os lugares auxiliares são:

- *Checks*: A quantidade de uso dos recursos pelo SGBD;
- *Provision*: A quantidade de recursos disponível para provisionamento;

O lugar *Checks* recebe valores de uso de recurso e *Provision*, o número de recursos para provisionamento. Para a adição e remoção dos recursos, a PrT-PRO trata os recursos computacionais como unidade, ou seja, são provisionados um a um. Os resumos dos lugares e sua descrição podem ser observados na Tabela 4.1.

Tabela 4.1: Lugares pertencentes a PrT-PRO para SGBD.

Lugares	**Descrição**
Checks	Uso de recurso computacional atualizado.
Provision	Quantidade de recursos computacionais disponível.
Idle	Baixo uso de recurso computacional.
Stable	Uso de recurso computacional estável.
Overload	Alto uso de recurso computacional.

Com base nos lugares, foram descritos os eventos esperados pela PrT-PRO para SGBD. Analisando o que é esperado pela PrT-PRO, foram definidas oito transições descritas na Tabela 4.2. As váriaveis **u** e **r** representam respectivamente o uso de recurso pelo sistema e a quantidade de recurso disponível para provisionar. Os *thresholds* definidos nos estados variam, conforme a escolha do recurso analisado. Os valores são denominados nas transições como *threshold* que definem a condição para indicar qual é a ação a ser executa, delimitando a sequência de disparos das transições da rede. Na PrT-PRO, o *th_min* é o *threshold* inferior, em que o SGBD se encontra *idle* e, posteriormente, é avaliada a necessidade de remoção do recurso. O *th_max*, representa o *threshold* superior, o SGBD passa para o estado *overload*, sendo adicionado mais recursos baseado na disponibilidade do mesmo.

Nas Figuras 4.1, 4.2 e 4.3, são representandos as sub-redes da PrT-PRO para SGBD, respectivamente a sub-rede *Idle*, *Stable* e *Overload*. Com base nos valores de uso de recurso

pelo SGBD, será designado qual é o caminho seguido pela PrT-PRO. Os valores são definidos para cada lugar apresentado na Tabela 4.1. Cada atualização de uso de recurso que o lugar *Checks* recebe, verifica qual transição será habilitada, sendo elas T_0 segue sub-rede *Idle*, T_1 para sub-rede *Overload* e T_2 na sub-rede *Stable*. O rd presente na sub-rede é o recurso disponível, este não atualizado pelo sistema.

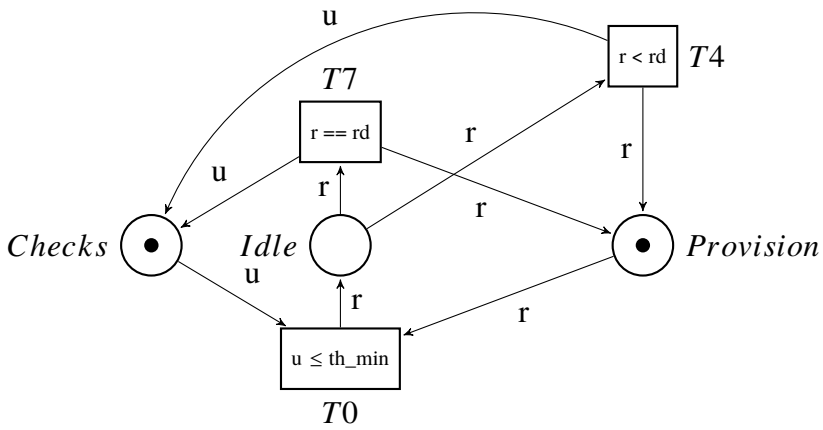


Figura 4.1: Sub-rede Predicado/Transição *Idle*

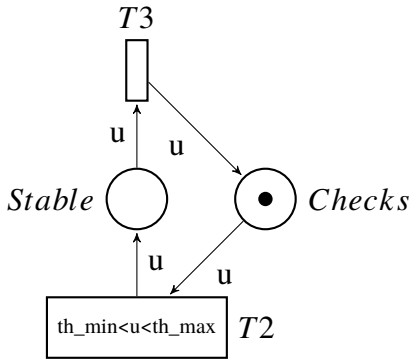


Figura 4.2: Sub-rede Predicado/Transição *Stable*

As sub-redes reunidas formam o modelo abstrato da PrT-PRO para SGBD que pode ser observada na Figura 4.4, com os lugares, transições e seus arcos de conexão, como também a marcação inicial. As marcas necessárias para a execução da rede são o uso de recursos pelo SGBD, bem como a quantidade de recursos disponível para provisionamento. A marcação inicial M_0 se encontra nos lugares *Checks* que contêm o uso de recurso atualizado pelo sistema, e o lugar denominado *Provision* possui a quantidade de recurso disponível para provisionar. Os valores de uso de recursos são provenientes do próprio sistema, os recursos disponíveis para provisionamento (r) são designados antes da execução da rede e alterados, conforme a ação realizada. Quando habilitada uma transição onde os arcos de post repassam as marcações pra *Checks* e *Provision*, as variáveis são atualizadas com a avaliação do sistema, acontecendo externamente da PrT-PRO, ou seja a PrT-PRO é alimentada com estas informações. A próxima

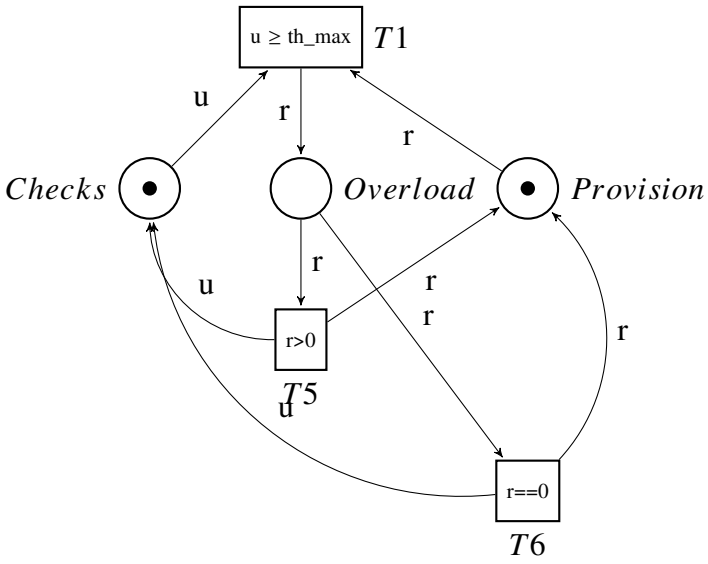


Figura 4.3: Sub-rede Predicado/Transição *Overload*

seção descreve como foram realizados os testes no modelo abstrato, e qual recurso computacional (métrica) adotado para provisionamento, ou seja, que assumirá os valores de **r** e **u**.

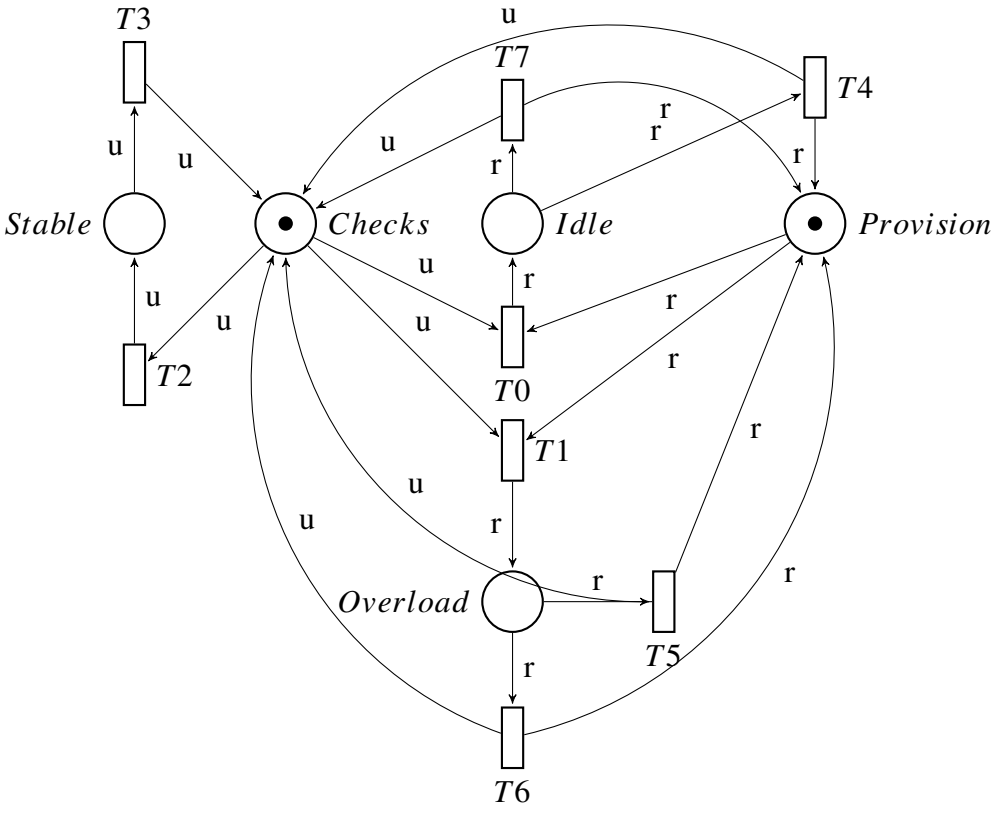


Figura 4.4: Rede de Predicado/Transição de provisionamento de recursos para SGBD

Tabela 4.2: Transições da PrT-PRO

Transição	**Descrição**
T_0: **u** $\leq$ *min_threshold*	O uso de recurso se encontra baixo o valor de **u** passa para o lugar *Idle*.
T_1: **u** $\geq$ *max_threshold*	O uso de recurso é considerado alto, o valor de **u** passa para o lugar *Overload*.
T_2: *min_threshold* $<$ **u** $<$ *max_threshold*	O uso de recurso está estável, o valor de **u** passa para o lugar *Stable*.
T_3	É atualizado o valor de **u** com o sistema
T_4: r< recurso Disponível	Se afirmativo remove recurso e atualiza valores de **u** e **r**
T_5: **r**> 0	Se for afirmativo adiciona recurso, atualiza os valores de **u** e **r**
T_6: r== 0	Uso de recurso se encontra alto, mas sem recurso atualiza os valores de **u** e **r**
T_7: **r** == recurso disponível	Sistema com baixo uso de recursos, atualiza **u** e **r**

4.3 Testes no Modelo Abstrado

A métrica adotada, foi a porcentagem (%) de uso de processamento (CPU) pelo SGBD. Em seguida, foram definidos valores de uso de recurso (**u**), para a adição e remoção de recursos, baseado em trabalhos anteriores [Arrieta and García, 2014, Minhas et al., 2012] que usam diferentes estratégias. Sendo, no primeiro, os autores apresentam valores para avaliação de consumo de CPU, definindo a utilização como:

- Excelente: 0% a 10%;
- Bom: 11% a 40%;
- Regular: 41% a 70%;
- Ruim: 71% a 100%.

No segundo, os autores definem dois valores para o provisionamento de recursos sendo um o pico de consumo de CPU e outro o mínimo:

- Pico consumo: 80%;
- Mínimo de consumo: 20%.

Para o teste no modelo foram utilizados os valores apresentados na Tabela 4.3, que fazem parte da condição das transições da PrT-PRO. O *th_min* (limiar inferior) foi 10% e o *th_max* (limiar superior) em 70%.

Tabela 4.3: Valores Definidos para Utilização de Processamento (CPU).

Intervalo	**Estado**
$x \leq 10$	*Idle*
$10 < x < 70$	*Stable*
$x \geq 70$	*Overload*

Depois da modelagem da PrT-PRO, foram testadas algumas situações que podiam ocorrer com base no uso de CPU pelo SGBD. Será demonstrado passo a passo, qual é o caminho de execução da PrT-PRO, conforme os valores adotados descritos na Tabela 4.4, que apresenta também, o caminho que a rede percorre. Cada valor refere-se às variáveis **u** e **r** na rede.

A Tabela 4.4, apresenta possíveis valores de uso de recurso (**u**), neste caso CPU. A quantidade de recursos (**r**) correspondente a **u**, ou seja, quantos recursos ainda estão disponíveis baseado na porcentagem de uso. Identifica o caminho percorrido pela PrT-PRO baseado em **u** e **r** é demonstrado por Transição-Lugar-Transição. Para a simulação considera-se um *hardware* de 6 *cores*.

Tabela 4.4: Valores Utilizados na Simulação.

CPU (u)	**Recurso (r)**	**Caminho**
10%	5	T_0 - *Idle* - T_7
20%	5	T_2 - *Stable* - T_3
40%	5	T_2 - *Stable* - T_3
90%	5	T_1 - *Overload* - T_5
80%	4	T_1 - *Overload* - T_5
40%	3	T_2 - *Stable* - T_3
20%	3	T_2 - *Stable* - T_3
10%	3	T_0 - *Idle* - T_4
10%	4	T_0 - *Idle* - T_4
30%	4	T_2 - *Stable* - T_3
80%*	0	T_1 - *Overload* - T_6
80%*	0	T_1 - *Overload* - T_6

*Valores adicionados para simular quando não existe mais recursos disponíveis.

Quando o uso de CPU for igual ou inferior a 10% a PrT-PRO somente irá percorrer a sub-rede especificado na Figura 4.1. Nesta sub-rede considera-se que o uso baixo de CPU, se não estiver com recursos ociosos passará somente pela transição T_7. Caso contrário, passará pela transição T_4 e removerá o recurso ocioso.

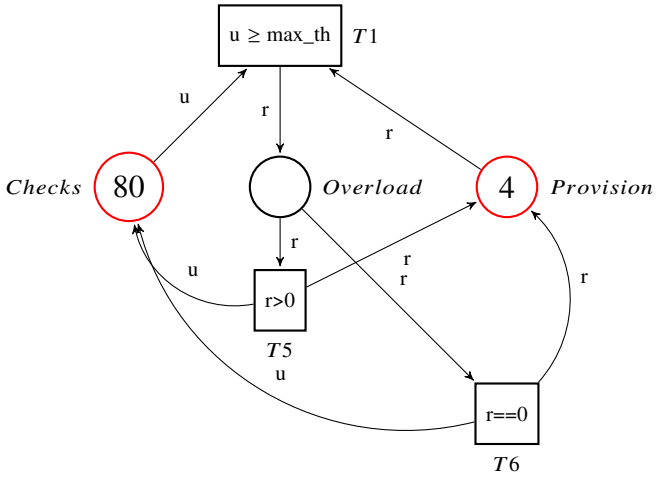
Se o uso de CPU estiver entre o intervalo de 11% à 69%, percorrerá a sub-rede da Figura 4.2. Quando percorrer está sub-rede o uso de CPU se encontra estável. Assim a PrT-PRO não irá realizar o provisionamento, somente a atualização do valor de uso de CPU (**u**).

Caso o uso de CPU (**u**) se encontre maior ou igual que 70%, é considerado *overload* ou seja, alto uso de CPU, assim o caminho percorrido na PrT-PRO, será pela sub-rede da Figura 4.3. A transição T_5 será habilitada se existir recursos disponíveis para provisionar, se não existir mais recursos T_6 habilitará atualizando o valor de uso de CPU.

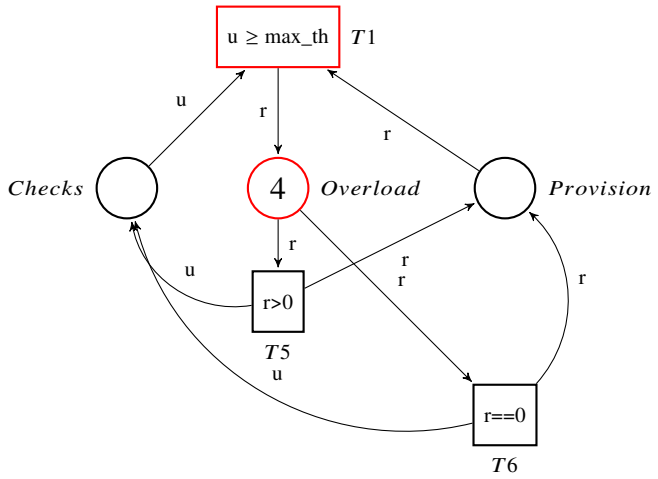
Um exemplo de disparo da PrT-PRO pode ser verificado na Figura 4.5, que demonstra na sub-rede *overload*, o que acontece quando tem um uso de CPU pelo SGBD de 80% e 4 recursos (*cores*) ainda disponíveis. Em vermelho estão os disparos das transições e os lugares que possuem as marcas para o disparo. Ao disparar T_1, é considerado que o sistema encontra-se no estado de *overload* sendo alto uso de CPU a marca r passa para o lugar *Overload*, a validação da transição T_1 ocorre baseada nos *thresholds*. Então T_5 é habilitada e dispara, a validação desta transição ocorre baseado no número de recursos ainda disponível. Ao disparar T_1 as variáveis u e r, recebem os valores de uso de recursos pelo sistema e quantidade de recursos disponível para provisionar atualizados com dados do sistema e o lugar *Provision* e *Checks* recebem a marcação, respectivamente u e r.

Na simulação do modelo é considerado a ideia de alteração desses valores, no caso da Figura 4.5, *Checks* recebeu 40% e *Provision* 3, representa como se o sistema reduziu a porcentagem de uso de recurso quando este foi adicionado e consequentemente diminuiu a quantidade de recursos disponível. Neste caso T_6 não é habilitada, pois ainda existem recursos

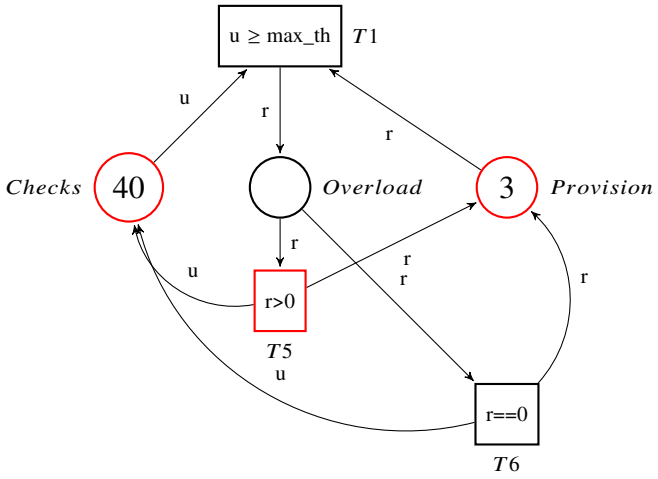
disponíveis para provisionamento. Na sequência novamente será avaliado T_0, T_1 e T_2 para verificar qual transição vai ser habilitada e consequentemente qual sub-rede percorrerá.



(a) Antes disparo, T1 habilitada.



(b) Depois Disparo T1, T5 habilitada.



(c) Depois Disparo T5.

Figura 4.5: Exemplo disparo da PrT-PRO para SGBD

4.4 Implementação a Rede de Petri Predicado/Transição

Para a implementação do protótipo da PrT-PRO, foi adotado a linguagem Java na versão 1.7.0_95. A escolha da linguagem se baseou principalmente na afinidade de programação com Java e considerando que é uma das linguagens mais utilizadas atualmente [Cass, 2015].

No protótipo, as transições, lugares e arcos são classes que representam os elementos da PrT-PRO. Para as transições foi criado uma classe abstrata, devido cada transição possuir métodos em comum, mudando somente como este é executado, como o exemplo dos Códigos 4.1 e 4.2. Pode ser visto o método *validatePredicate()*, realiza a validação das condições de cada transição. No Código 4.1 é validado a condição da transição t_0 e no Código 4.2 a condição de t_1.

Código 4.1: Exemplo método T0 do protótipo

```
1  /**
2  * Valida a Condição u<=minimo
3  */
4   @Override
5  public int validatePredicate() {
6       /**
7       *os arcos tem nomes em comum para lugar e transicao.
8       */
9      ArcPre tmp1 = new ArcPre();
10     tmp1= searchPre("arct0Checks");
11     ArcPre tmp2 = new ArcPre();
12     tmp2=searchPre("arct0Provision")
13         if ((tmp1.getToken()!=null) && (tmp2.getToken()!=null)) {
14             setU(tmp1.getToken().getValue());
15             setR(tmp2.getToken().getValue());
16             if (getU() <= getValueMin()) {
17                 action();
18                 log.log(getName(), "SGBD com baixo uso de recursos");
19             }else{
20                 t3.setRes(tmp2.getToken());
21             }
22         }
23     tmp1.setToken(null);
24     tmp2.setToken(null);
25     return 0;
26  }
```

Para a validação de t_0, são buscados em seus arcos correspondentes as variáveis de **r** e **u**, que são fornecidas pela classe *ArcPre*. Todos os lugares e transições possuem arcos de entrada e arcos de saída. O mesmo ocorre na validação de T_1, a diferença entre os dois métodos é a condição verificada. Para T_0 disparar é validado se o percentual de uso de recursos é menor que o *threshold* mínimo definido e em T_1 a condição utiliza o valor máximo adotado. Após

validar a condição em cada um dos métodos, são descritas informações em um *log* gerado pelo protótipo, em que é possível verificar qual método foi habilitado.

Código 4.2: Exemplo método T1 do protótipo

```
/**
* Valida a Condição T1 u>=maximo
*/
 @Override
 public int validatePredicate() {
    ArcPre tmp1 = new ArcPre();
    tmp1 = this.searchPre("arct1Checks");
    ArcPre tmp2 = new ArcPre();
    tmp2 = this.searchPre("arct1Provision");
    if ((tmp1.getToken()!= null) && (tmp2.getToken() != null)) {
        setU(tmp1.getToken().getValue());
        setR(tmp2.getToken().getValue());
        if (getU() >= getValueMax()) {
            action();
            log.log(getName(), "Detectado SGBD com alto uso de recurso");
        }else{
            t3.setRes(tmp2.getToken());
        }
    }
    tmp1.cleanup();
    tmp2.cleanup();
    setU(0);
    return 0;
}
```

A lógica utilizada nos arcos abrange arcos de entrada e saída. Os lugares e transições possuem arcos de entrada e saída: um arco de entrada de uma transição é um arco de saída de um lugar, e assim mutualmente. Tem-se então as classes *ArcPre* e *ArcPost*. Para controlar qual arco é de entrada de um elemento e qual é o arco de saída, cada arco pre criado para um elemento possui um arco post no elemento qual é conectado. A classe *ArcPre*, apresenta um método que informa o arco de saída que corresponde. Funciona como um *Observer*[1], quando o arco pre recebe a variável notifica o arco post relacionado.

A classe que representa os lugares possui um método que realiza a passagem da marca que se encontra em seu arco pre, para os arcos post conectados ao lugar. A passagem deste parâmetro é feita a todos os arcos pre relacionado ao lugar. Posteriormente cada transição relacionada verificará se sua condição é satisfeita, para definir a direção da PrT-PRO. No método da classe Lugar apresentado no Código 4.3, sempre é verificado se o lugar possui uma marca,

[1]Padrão de projeto, que especifica que os objetos dependentes são avisados quando o objeto de qual eles dependem muda de estado [Erich et al., 2000]

caso contrário é buscado nos arcos de pré e repassados para os arcos de post respectivos que modificaram os seus valores e os arcos pre correspondentes.

Código 4.3: Exemplo método da classe *Place* do protótipo

```
1  /**Passagem da marca de um lugar para os arcos correspondentes**/
2   @Override
3  public void run() {
4      List<ArcPre> arcPreList;
5      List<ArcPost> arcPostList;
6      arcPostList = getArcPostList();
7      arcPreList = getArcPreList();
8      /* Se estiver com a marca, ou seja primeira execucao */
9      if (token != null) {
10         log.log(getName(), "Marcacao Inicial, Valor:"+getToken().getValue()
               );
11         for (Iterator<ArcPost> it = arcPostList.iterator(); it.hasNext();)
               {
12             ArcPost arc;
13             arc = it.next();
14             Token tk = new Token();
15             tk = token;
16             arc.setToken(tk);
17         }
18         setToken(null);
19     } else {
20          for (Iterator<ArcPre> pre = arcPreList.iterator(); pre.hasNext();)
                {
21              ArcPre arc = pre.next();
22              this.setToken(arc.getToken());
23              arc.setToken(null);
24              if (getToken() != null) {
25                log.log(getName(), "Distribuindo Marcas, valor:"+getToken()
                      .getValue());
26                for (Iterator<ArcPost> post = arcPostList.iterator(); post.
                      hasNext();) {
27                    ArcPost arcp;
28                    arcp = post.next();
29                    arcp.setToken(token);
30                }
31                arc = null;
32             }
33          }
34         setToken(null);
35     }
36     return;
37 }
```

As marcas da PrT-PRO, formam uma classe denominada *Token*, que carrega os dois diferentes tipos utilizados que são **u** (uso de recurso pelo SGDB) e **r** (recursos disponíveis), esta classe somente armazena o valor da marca que corresponde. A classe *Token* é instanciada pelas classes que correspondem aos lugares, transições e arcos.

Para a criação da PrT-PRO, foi criado uma classe chamada de *PredicateTransictionNets*, responsável pela ligação entre os arcos pre e post (ver Código 4.4), adição dos elementos na rede e sua execução.

Código 4.4: Exemplo método da classe *PredicateTransictionNets* do protótipo

```
1  /*
2   * liga os lugares nas transicoes e vice e versa, recebe a entrada saida e
3   * os arcos de pre e post
4   ,,,
5   */
6  public void linkPt(Engineer input, String arcPre, Engineer output, String
       arcPost) {
7
8      ArcPost arc;
9      arc = input.searchPost(arcPost);
10     ArcPre pre;
11     pre = output.searchPre(arcPre);
12     pre.addObservable(arc);
13 }
```

Além das classes referentes a PrT-PRO para SGBD, foram criadas classes auxiliares, que buscavam as informações de uso de recursos pelo SGBD, como também variáveis auxiliares que mantinham a quantidade de recursos definida antes da execução da PrT-PRO, para serem utilizadas nas transições T_4 e T_7. Os valores referentes aos *thresholds* e quantidade de recursos, são lidos através de um arquivo de configuração, pré definido antes da execução da PrT-PRO.

A marcação inicial da PrT-PRO é lida de um arquivo que é alimentado pela ferramente de análise do sistema *Dstat* [Wieers, 2015]. Em cada transição em que os arcos de post estão passando as variáveis para os lugares $Checks$ e $Provision$, a ação da transição consiste em repassar o valor da quantidade de uso de recursos pelo SGBD e a quantidade ainda disponível de recursos atualizados pelo *Dstat*. A quantidade de uso de recursos por *core* era analisado a cada dez segundos, para evitar que o recurso adicionado não fosse removido, em caso de pico de carga momentâneo.

O protótipo da PrT-PRO para SGBD, foi elaborado para seguir as definições de Redes de Petri e validar o modelo abstrato verificando o desempenho do SGBD, baseado na ação de adicionar e remover os recursos de acordo com a necessidade encontrada. O uso de recurso é avaliado pelas transições e segue para o lugar correspondente. As variáveis são atualizadas sempre que necessário com informações do sistema.

4.5 Discussão

Foram apresentadas, neste capítulo, a PrT-PRO para SGBD, que para realizar o provisionamento se baseia em estados de desempenho que o SGBD experimenta, durante a execução de uma determinada carga de trabalho. Esses estados são alcançados, conforme a definição do recurso a ser observado e dos *thresholds* adotados. Apresentamos também, através de simulação, o comportamento da PrT-PRO, considerando a CPU como recurso observado.

A criação do modelo da PrT-PRO permite uma visão mais abstrata dos recursos computacionais. Através da modelagem dos *thresholds*, é possível estender a PrT-PRO para qualquer outro recurso computacional. A marcação da PrT-PRO possibilita representar as ações dinâmicas e simultâneas que ocorrem no provisionamento de recursos. A PrT-PRO pode ser acoplada a diferentes SGBD, uma vez que não precisa conhecer seus aspectos internos, avaliando a necessidade de provisionamento a partir do uso do recursos e sua quantidade disponível.

A PrT-PRO é um modelo abstrato e realista do provisionamento de recursos computacionais. Neste trabalho, foram apresentadas partes importantes do código do protótipo desenvolvido, que modela lugares, transições e arcos.

O próximo capítulo apresenta a descrição do ambiente utilizado nos experimentos realizado com o protótipo criado, a carga de trabalho utilizada e os resultados alcançados utilizando a PrT-PRO para SGBD.

Capítulo 5

Experimentos

Este capítulo descreve a metodologia utilizada para realização dos experimentos e apresenta o ambiente experimental. Também serão expostos os resultados obtidos nos experimentos utilizando a PrT-PRO para SGBD.

5.1 Ambiente Experimental

5.1.1 Hardware e Software

Para a execução dos experimentos, nós utilizamos duas máquinas com processador Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GHz 20-Core de 64 bits. Cada processador possui 2 processadores com 10 *cores* e com suporte a 2 *threads* por *core*. A memória *cache* L1 possui 32KB por *core*, memória L2 com 256KB por *core* e L3 com 25MB por *chip*. Cada máquina possui 32GB (Gigabyte) de memória RAM, com 500 GB de armazenamento em disco rígido sobre o sistema operacional Debian 8, versão "jessie"com kernel 4.9.2-10 X86_64. Nós utilizamos o SGBD PostgreSQL [PostgreSQL, 2015] na versão 9.4.5 obtido diretamente do site do desenvolvedor.

Todos os experimentos foram realizados exclusivamente nesse ambiente. Para analisar o uso de CPU pelo SGBD, nós utilizamos a ferramenta *Dstat* [Wieers, 2015] que permitiu o monitoramento por *core* do sistema. O *Dstat* é uma ferramenta de estatísticas de recursos, sendo útil para monitorar o sistema durante avaliações de desempenho. Utilizamos a ferramenta *Likwid* [Treibig et al., 2010] para avaliar o acesso à memória pelo processador e também a funcionalidade *cgroups* do *kernel*[1] para limitar e controlar os recursos de *hardware*.

[1]Componente do Sistema Operacional.

5.1.2 Cgroups

Cgroups ou grupos de controle é um recurso fornecido pelo *kernel* do Linux[2], onde é possível alocar recursos como rede, memória, CPU e largura de banda para grupos que executam processos no sistema. Utilizando *cgroups*, é possível definir o acesso a certos recursos computacionais, monitorar quem os utiliza e controlar os recursos.

Utilizando *cpuset*, subsistema de *cgroups*, é possível restringir o uso da CPU, controlando quais os núcleos (*cores*) que podem ser utilizados pelo SGBD em nossos experimentos, através de um arquivo de configuração que define os *cores* a serem utilizados. O custo dessa operação é muito pequeno, viabilizando a sua utilização para nossos experimentos, pois a adição ou remoção desses recursos não gera sobrecarga significativa, minimizando interferências na avaliação do desempenho do sistema.

Com *cgroups*, foi criado um grupo de controle para utilização das tarefas do SGBD. Durante a execução da PrT-PRO, o arquivo de configuração é alterado, quando detectada a necessidade de adição ou remoção de *cores*, limitando assim quais os *cores* estão disponíveis para o uso do SGBD. A modificação do arquivo de configuração é realizada através de dois códigos escritos em *shell script*[3] apresentados em Código 5.1 e Código 5.2. Esses códigos são chamados pela PrT-PRO ao detectar a necessidade de provisionamento. Deste modo, pode ser definido como será adicionado e removido o recurso sem alteração da PrT-PRO.

Código 5.1: Adição de recursos em *shell-script*

```
1 #!/bin/bash
2 cp -a /etc/cgconfig.conf cgconfig-tmp.conf;
3 VALOR= $(awk -F '-' '/cpuset\.cpus=/ { V=$2; } END { V++; print V;}' /etc/
      cgconfig.conf;)
4 sed -E "s/cpuset.cpus=[0-9]+\-[0-9]+/cpuset\.cpus=0\-$VALOR/g" cgconfig-tmp
      .conf > /etc/cgconfig.conf;
5
6 rm cgconfig-tmp.conf;
7 cgconfigparser -l /etc/cgconfig.conf
```

Código 5.2: Remoção de recursos chamado pela PrT-PRO em *shell-script*

```
1 #!/bin/bash
2 cp -a /etc/cgconfig.conf cgconfig-tmp.conf;
3 VALOR=$(awk -F '-' '/cpuset\.cpus=/ { V=$2; } END { V--; print V; }'/etc/
      cgconfig.conf;)
4 sed -E "s/cpuset\.cpus=[0-9]+\-[0-9]+/cpuset\.cpus=0\-$VALOR/g" cgconfig-
      tmp.conf > /etc/cgconfig.conf;
5
6 rm cgconfig-tmp.conf;
7 cgconfigparser -l /etc/cgconfig.conf
```

[2]Sistema Operacional

[3]*Shell script é uma linguagem de script, usada em diversos sistemas operacionais.*

5.2 Metodologia de Testes

Nesta seção, está descrito como foram realizados os experimentos, o cálculo do tempo de execução, as bases de dados e qual foi o critério para a escolha das consultas.

5.2.1 Base de Dados e Consultas

Para a realização dos experimentos com a PrT-PRO para SGBD, nós utilizamos uma base de dados que tivesse como característica o alto uso de processamento (CPU), recurso adotado para monitoração da PrT-PRO. Ainda lançamos mão dois *benchmarks* para a criação da base de dados e a carga de trabalho: o TPC-DS [Council, 2015a] e o TPC-H [Council, 2015b]. O TPC-DS e o TPC-H são referências de banco de dados de suporte a decisão.

O *benchmark* TPC-DS utiliza um modelo de negócio baseado em uma grande empresa de varejo que possui lojas distribuídas pelo país e com vendas via catálogos e internet. As consultas apresentam alta complexidade operacional e diferentes exigências (OLAP (*Online Analytical Processing*, *ad-hoc*, mineração de dados, relatórios). O *benchmark* TPC-DS é caracterizado pelo alto uso de CPU e memória. Seu esquema possui 24 tabelas, demonstradas na Tabela 5.1, essas tabelas são divididas em tabelas de fatos e dimensões e é composto por 99 consultas que buscam resolver problemas complexos de negócios com uma variedade de padrões de acesso, operadores e restrições.

Tabela 5.1: Tabelas base de dados TPC-DS

Tabelas	
Fatos	store_sales, store_returns catalog_sales, catalog_returns web_sales, web_returns e inventory
Dimensões	store, call_center, promotion catalog_page, web_site, reason web_page, warehouse, ship_mode customer, customer_address, time_dim customer_demographics, date_dim household_demographics, item, income_band customer_demographics, date_dim

O *benchmark* TPC-H não modela um segmento de mercado específico, mas atividades comuns a variados segmentos. Possui consultas com alta complexidade, que demonstram as atividades de vendas de um fornecedor. Seu esquema é composto de 8 tabelas, sendo tabelas de fatos e dimensões apresentadas na Tabela 5.2. Possui 22 consultas, que representam situações de uma aplicação de análise de negócios complexas.

Foram criados dois bancos de dados, um para TPC-H e outro para o TPC-DS de 100GB cada uma. Foi utilizado, dos respectivos *benchmarks*, o esquema do banco de dados, a base de dados e algumas consultas. Como o objetivo dos experimentos realizados com a PrT de

Tabela 5.2: Tabelas base de dados TPC-H

Tabelas	
Fatos	lineitem, orders
Dimensões	customer, nation part, partsupp region e supplier

provisionamento de recursos computacionais para SGBD, era avaliar o uso de CPU, não foram utilizadas nos testes todas as consultas do TPC-DS e do TPC-H, a seguir está descrito como foram selecionadas.

5.2.2 Roteiro de Testes

Cada consulta de sistemas de apoio à decisão tem o seu padrão de utilização de recursos de *hardware*, ou seja, algumas utilizam mais processamento e outras, mais memória. Segundo [Meikel et al., 2007], no TPC-H as consultas que utilizam mais CPU, possuem as características da consulta 1, apresentada no Código 5.3. No TPC-DS, possuem as características da consulta 70, presente no Código 5.4.

Código 5.3: Consulta com características de uso de processamento - TPC-H

```
1  SELECT
2
3          l_returnflag,
4          l_linestatus,
5          sum(l_quantity) as sum_qty,
6          sum(l_extendedprice) as sum_base_price,
7          sum(l_extendedprice *(1 - l_discount)) as sum_disc_price,
8          sum(l_extendedprice *(1 - l_discount) * (1 + l_tax)) as sum_charge,
9          avg(l_quantity) as avg_qty,
10         avg(l_extendedprice) as avg_price,
11         avg(l_discount) as avg_disc,
12         count(*) as count_order
13 FROM
14         lineitem
15 WHERE
16         l_shipdate <= date '1998-12-01' - interval ':1' day
17 GROUP BY
18         l_returnflag,
19         l_linestatus
20 ORDER BY
21         l_returnflag,
22         l_linestatus
23 LIMIT 1;
```

Código 5.4: Consulta com características de uso de processamento - TPC-DS

```
SELECT
    sum(ss_net_profit) as total_sum,
    s_state,
    s_county,
    grouping(s_state)+grouping(s_county),
    rank()over(partition by grouping(s_state)+grouping(s_county),
    case when grouping(s_county)=0
    then s_state end
order by sum(ss_net_profit) desc)
FROM store_sales,
     date_dim,
     store
WHERE d_year = [YEAR]
    AND d_date_sk = ss_sold_date_sk
    AND s_store_sk = ss_store_sk
    AND s_state in
(SELECT s_state
 FROM (SELECT
  s_state,
  rank()over(partition by s_state
  order by sum(ss_net_profit)desc) as r
 FROM store_sales,
  store,
  date_dim
 WHERE d_year =[YEAR]
  AND d_date_sk = ss_sold_date_sk
  AND s_store_sk = ss_store_sk
 GROUP BY s_state)
 WHERE r <= 5)
 GROUP BY ROLLUP(s_state,s_county)
 ORDER BY
  lochierarchy desc,
 CASE WHEN lochierarchy = 0 THEN s_state END,
  rank_within_parent;
```

As consultas do TPC-DS e do TPC-H que utilizam mais CPU, realizam operações complexas de funções de agrupamento e classificação que colaboram para uma sobrecarga de CPU, tendo assim uma utilização intensiva de tal recurso [Meikel et al., 2007]. Assim, foram escolhidas cinco consultas de cada *benchmark* que tivessem essas características. As consultas foram colocadas em cinco arquivos (usuários), cada um dos quais possui as cinco consultas em ordens diferentes.

Na execução dos experimentos, os cinco arquivos que nomeados de usuários, foram executados concorrentemente. Ao terminar todas as consultas, era calculado o tempo de execução, métrica adotada para avaliação do desempenho (tempo final menos o tempo inicial).

Nós executamos os experimentos com diferentes *thresholds*, os quais foram primeiramente baseados nos seguintes trabalho [Arrieta and García, 2014, Minhas et al., 2012]. A Tabela 5.3, apresenta os *thresholds* utilizados nas transições da PrT-PRO para validar os predicados. A validação dos predicados determina em que estado de uso de recursos se encontra o SGBD. Os valores são definidos em *threshold_max* que é utilizado para verificar quando o uso de CPU pelo SGBD se encontra no estado de *Overload*, e o *threshold_min* quando o uso de CPU pelo SGBD está *Idle*. O estado *Stable* é quando o uso de CPU está entre esses dois valores. Os *thresholds* foram utilizados durante a execução de ambos os benchmarks: TPC-DS e TPC-H.

Tabela 5.3: *Thresholds* Definidos nos Experimentos

Thresholds (PrT-PRO)	***threshold_min***	***threshold_max***
1	10%	70%
2	20%	70%
3	10%	80%
4	20%	80%

Além dos experimentos utilizando a PrT-PRO para SGBD, foram realizados experimentos com o número total de recursos (*cores*) que as máquinas utilizadas possuem e com o mínimo possível de CPU. Os experimentos foram utilizados como base para comparar o desempenho quando a PrT-PRO é executada. Os resultados são apresentados na seção subsequente.

5.3 Resultados

O objetivo desta seção é comprovar a hipótese de que, utilizando a PrT-PRO para SGBD, é possível encontrar um valor ótimo de recursos, no qual a demanda da carga de trabalho será atendida e, ao provisionar o valor ótimo de recursos computacionais, é presumível que o desempenho do SGBD melhore com o aproveitamento da *cache* do processador através do aumento de *hits* e diminuição de *misses*.

Inicialmente, nós apresentamos o tempo médio de execução obtido em cada carga de trabalho utilizada. Cada experimento foi executado dez vezes e os resultados consistem da média dessas execuções. Nas Tabelas 5.4 e 5.5, apresentamos o tempo médio de execução obtido nos experimentos iniciais, o desvio padrão[4] e o coeficiente de variação (CV)[5] que em todos os experimentos ficou menor que 20%, caracterizando uma amostra homogênea. Compara os *thresholds* utilizados na PrT-PRO (PrT-PRO de um a quatro), com o RD (recurso disponível no *hardware*) e com menos *cores*.

[4] O desvio padrão é a medida que mostra a variação existente em relação a média.
[5] Utilizado para analisar a dispersão relativo ao seu valor médio.

Tabela 5.4: Tempo médio de execução obtido nos experimentos com TPC-DS (minutos)

	Sem a PrT		**Com PrT**			
	20 cores -RD	*1 core*	*PrT-PRO 1*	*PrT-PRO 2*	*PrT-PRO 3*	*PrT-PRO 4*
Média	44,52	55,23	19,41	23,05	21,15	20,05
Desvio Padrão	5,45	6,25	0,92	2,20	2,33	2,62
CV	12%	11%	4%	9%	11%	13%

Nas Tabelas 5.4 e 5.5, encontram-se os resultados obtidos com a execução da PrT-PRO para SGBD como provisionador dos recursos. Além disso, foram utilizados os *thresholds* definidos na Tabela 5.3. Constam também nas tabelas os experimentos nos quais limitamos a execução, utilizando apenas um recurso (1 *core*), bem como aqueles em que utilizamos todos os recursos presente na máquina utilizada (20 *cores*) definindo como **RD** (recursos disponíveis no *hardware*). Já era esperado que a execução dos experimentos com o mínimo possível de *cores* obtivesse o pior tempo de resposta, confirmando que a falta de recursos prejudica o desempenho do sistema.

Os experimentos realizados com a carga de trabalho gerada através do TPC-DS, ao executar a PrT-PRO para SGBD, atingiu os melhores tempos de resposta, em especial o *threshold* com remoção do recurso em 10% e adição de recursos em 70%, apresentou o melhor desempenho. A média do tempo de execução foi de 43% comparado com o experimento executado com o RD utilizada e de 35% comparado com menos recursos.

Tabela 5.5: Tempo médio de execução obtido nos experimentos com TPC-H (minutos)

	Sem a PrT		**Com PrT**			
	20 cores -RD	*2 cores**	*PrT-PRO 1*	*PrT-PRO 2*	*PrT-PRO 3*	*PrT-PRO 4*
Média	432,15	593,52	275,31	296,49	295,14	292,84
Desvio Padrão	79,19	75,42	40,72	54,01	49,08	50,32
CV	18%	12%	14%	18%	16%	17%

*Não foi possível executar com apenas um recurso, a máquina reiniciava a cada tentativa

Os experimentos realizados com a carga de trabalho baseada no TPC-H apresentaram os resultados que podem ser verificados na Tabela 5.5, como ocorreu com a carga de trabalho TPC-DS, os piores tempos de execução são dos experimentos utilizando menos recursos (2 *cores*)[6] e com o máximo de recursos (20 *cores* - **RD**). Quando utilizado a PrT de provisionamento de recursos, foi encontrado o melhor tempo médio de execução novamente nos *thresholds* em que se removiam os recursos com 10% de uso de CPU e adicionavam os recursos com 70% de uso de CPU.

Comparado com a execução da carga de trabalho com todos os recursos, utilizando a PrT-PRO, a carga de trabalho executou em 64% do tempo de resposta com o RD. Como já

[6] Não foi possível executar com apenas um recurso, a máquina reiniciava a cada tentativa.

apresentado nos resultados do TPC-DS, a falta de recursos afeta o desempenho da aplicação, tal fato foi demonstrado nos resultados obtidos. O melhor tempo médio de execução alcançado com a PrT-PRO executou em 46% do tempo dos experimentos com menos recursos.

O objetivo era encontrar o número ideal de recursos, em que a carga de trabalho fosse executada com um bom desempenho e o uso de recursos computacionais pelo SGBD encontrasse em valores estáveis. Entre os *thresholds* definidos, ao executar os experimentos com a PrT de provisionamento de recursos computacionais para SGBD para ambas as cargas, foram adicionados os recursos de modo que atende-se o Algoritmo 1.

Nos experimentos com a carga de trabalho criada através do TPC-DS, o melhor tempo médio de execução, obtido com o *threshold*, 10% remove recursos e 70% adiciona recursos. O número de recursos provisionado pela PrT-PRO para SGBD foi de oito recursos, sendo que a execução iniciava com um recurso e foram provisionados sete recursos pela PrT-PRO. Os resultados alcançados estão apresentados na Figura 5.1, que compara os tempos médios de execução de cada resultado apresentado anteriormente com o RD. PrT-PRO seguida de um a quatro, representa cada *threshold* definido na Tabela 5.3, utilizados nos experimentos.

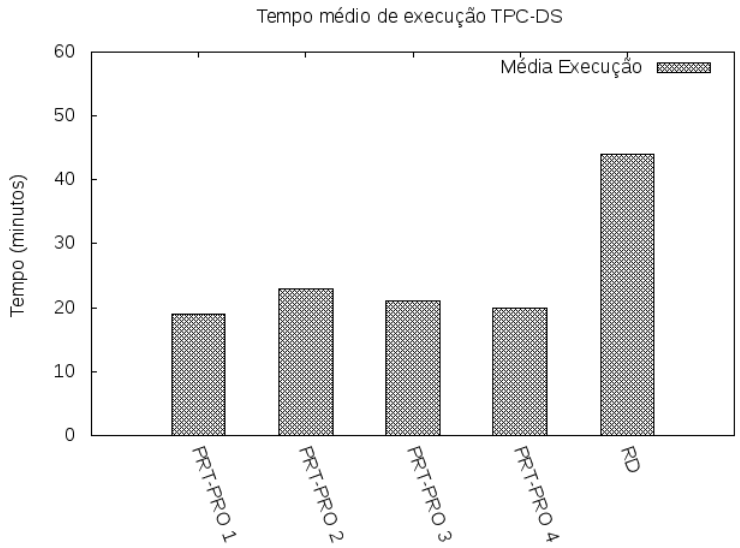


Figura 5.1: Tempo médio de execução obtido - TPC-DS

Nos experimentos com a carga de trabalho baseada no TPC-H, o melhor tempo médio de execução obtido com o *threshold* 10% remove recursos e 70% adiciona recursos. O número de recursos provisionados pela PrT-PRO para SGBD foi de nove recursos, sendo que a execução iniciava com dois recursos e foram provisionados sete recursos pela PrT-PRO. O tempo médio de execução alcançado está demonstrado na Figura 5.2, que compara os *thresholds* utilizados na PrT-PRO (PrT-PRO de um a quatro), com o RD.

As Figuras 5.3 e 5.4 demonstram o comportamento elástico apresentado pelo provisionamento de recursos computacionais para SGBD e o comportamento sem a execução da PrT-PRO, mas com o número de *cores* que a PrT-PRO provisionou para atender a carga de

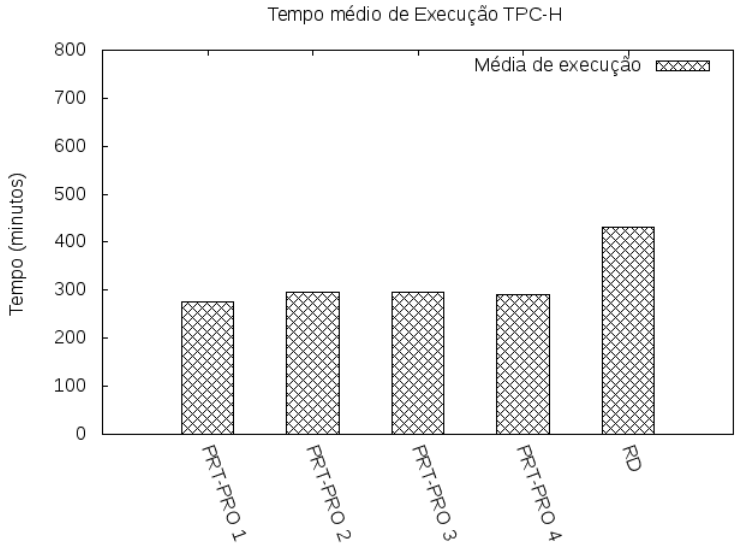


Figura 5.2: Tempo médio de execução obtido - TPC-H

trabalho. Os *thresholds* do comportamento apresentado é 10% remove recursos e 70% adiciona, sendo demonstrada somente parte da execução da PrT-PRO.

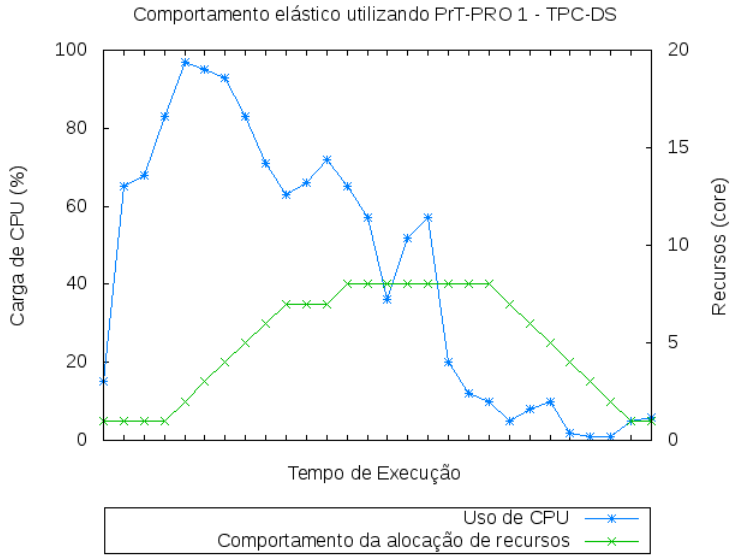
Pode ser verificado o comportamento elástico dos recursos ao utilizar a PrT-PRO para SGBD nas Figuras 5.3(a) e 5.4(a). Ao detectar o aumento no uso de CPU, a PrT-PRO adiciona mais *cores*, que são removidos se o uso de CPU for igual ou menor ao *threshold* definido como mínimo. A PrT-PRO atende a necessidade de demanda da carga de trabalho.

As Figuras 5.3(b) e 5.4(b) demonstram o comportamento do uso de CPU com o valor ótimo de recursos (*cores*), quando não temos a ação da PrT-PRO. Iniciou com um número fixo de *cores* e o mesmo seguiu em toda a execução da carga de trabalho com estável de CPU. Se houver, porém, aumento na demanda, essa quantidade de recurso pode não atendê-la e o tempo médio de execução pode não ser mais satisfatório, diminuindo o desempenho do SGBD. Então, se for detectado aumento ou diminuição de demanda, a PrT-PRO irá prontamente agir para provisionar adição ou remoção de recursos e, então, tentar evitar ociosidade nos recursos sistema.

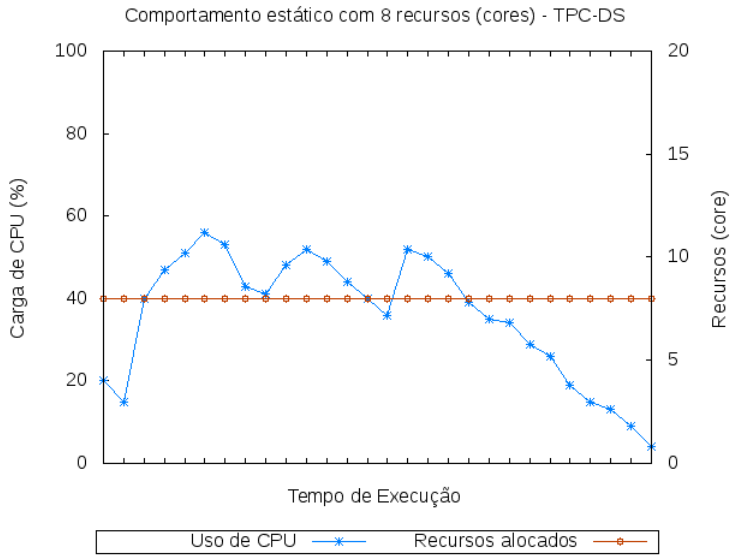
Quando é utilizado a PrT-PRO, pode ser notado a diferença no tempo médio de execução comparado com os experimentos em que foram utilizados todos os recursos (*cores*) presentes no *hardware*. Tal diferença é resultado da afinidade de *cache* entre processador e o processo, quando os dados acumulados no *core* são aproveitados. Para verificar se esses dados estão sendo aproveitados e como está sendo utilizada a memória *cache* do processador, nós avaliamos os *caches misses* e *hit* de cada experimento executado. O *cache miss* acontece, quando o dado buscado não é encontrado na memória *cache*, e *cache hit*, quando o dado é encontrado [Patterson and Hennessy, 2007].

O *miss rate* corresponde a:

$$Miss\ rate = 1 - (hit\ rate); \tag{5.1}$$



(a) Comportamento elástico do provisionamento de recursos com a PrT-PRO.



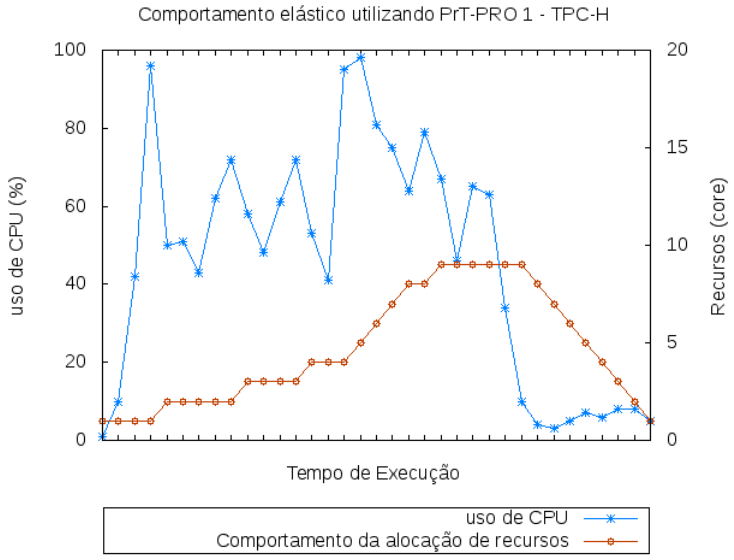
(b) Comportamento estático com 8 recursos (*cores*)

Figura 5.3: Comportamento dos recursos com o processamento das consultas - TPC-DS

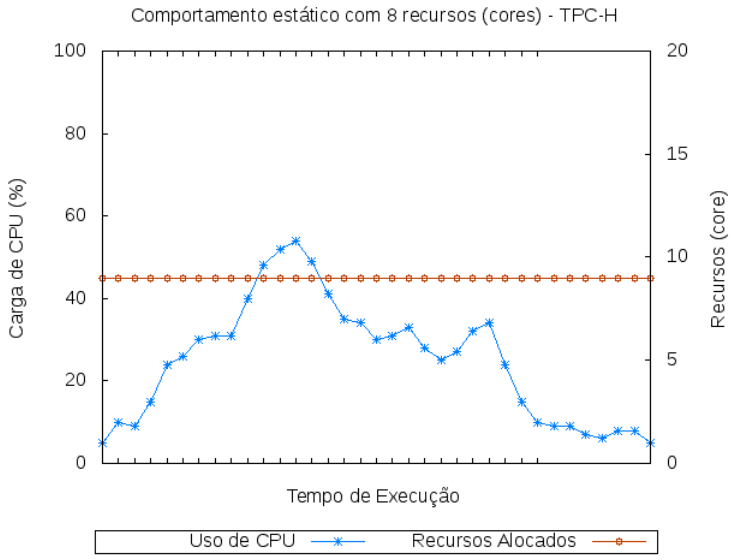
O *miss rate* é demonstrado por *core* do processador. É uma média calculada, através da soma das taxas divididas pelo número de *cores*, neste caso são 20. A taxa de *hit* corresponde à diferença da taxa de *miss*, até a soma dos dois juntos ser 0.1.

As Figuras 5.5 e 5.6 apresentam os resultados de *miss rate*, que é a taxa de erro de busca na memória. São avaliados *miss rate* e *hit rate*, da memória *L1* (Figura 5.5(a)) e *L2* (Figura 5.5(b)) dos experimentos realizados com o TPC-DS.

Na Figura 5.5(a), que apresenta da memória $L1$, a taxa de *miss* da execução com todos os *cores* disponíveis no *hardware* (RD) corresponde a uma média de $0,07$ por *core*. A taxa de *hit* do experimento com o RD resultou em $0,93$. Os experimentos com a PrT-PRO a taxa de



(a) Comportamento elástico do provisionamento de recursos com a PrT-PRO.

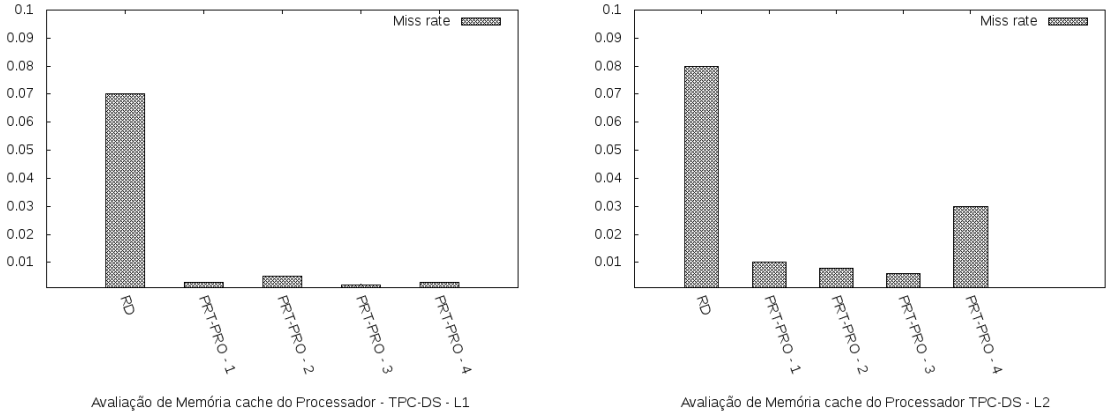


(b) Comportamento estático com 9 recursos (*cores*)

Figura 5.4: Comportamento dos recursos com o processamento das consultas - TPC-H

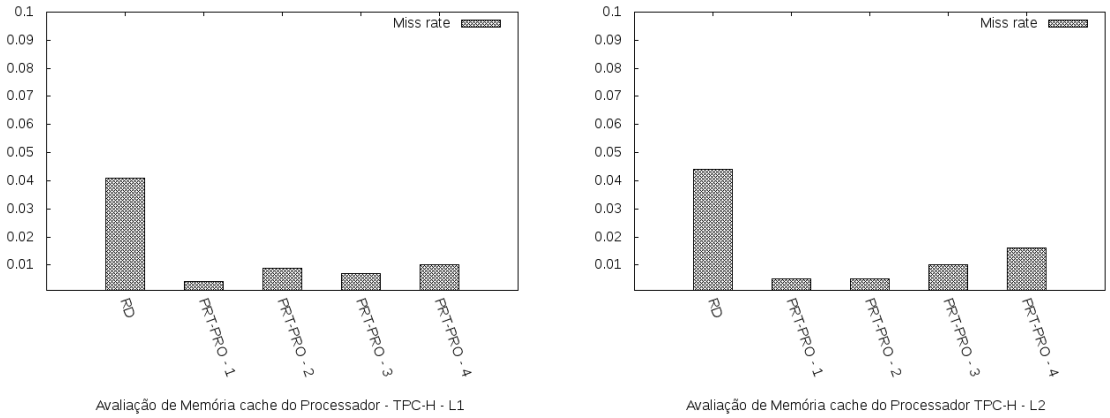
miss, não ultrapassou 0, 005. Quando se trata da Figura 5.5(b), em que são demonstrados o *miss rate* de $L2$, a execução dos experimentos com o RD apresenta 0, 08 de taxa de *miss* e 0, 92 de taxa de *hit*. A taxa de *miss* dos experimentos com a PrT-PRO não ultrapassou de 0, 03 de *miss rate* e 0, 994 de *hit rate*.

Na Figura 5.6(a), demonstra-se a taxa de *miss* $L1$ dos experimentos com o TPC-H. Nesse caso, a taxa de *miss* utilizando a PrT-PRO não ultrapassou 0, 01, enquanto a taxa de *miss* com todos os *cores* do *hardware* chegou a 0, 41. A Figura 5.6(b) apresenta a taxa de *miss* e da memória $L2$ do processador, chegando a 0, 44 com os experimentos com os *cores* do *hardware* e com a PrT-PRO não passou de 0, 01.



(a) Avaliação da memória *cache* L1. (b) Avaliação da memória *cache* L2.

Figura 5.5: Avaliação da memória *cache* do processador - TPC-DS



(a) Avaliação da memória *cache* L1 (b) Avaliação da memória *cache* L2 (*cores*)

Figura 5.6: Avaliação da memória *cache* do processador - TPC-H

Quando ocorre um *miss* de *cache*, a informação é buscada no nível inferior, ou seja, se procurar em $L1$ e não encontrar, busca em $L2$ e assim sucessivamente até alcançar a memória do *hardware*. Dessa maneira a diferença no tempo de execução é maior comparado a quando acontece mais *hits*, quanto mais inferior o nível, mais lento é a busca.

5.4 Discussão

Finalizando a análise dos resultados, pode ser constatado que, utilizando a PrT-PRO para SGBD, obteve-se bons resultados referentes ao tempo de execução e ao comportamento elástico do sistema. Essa conclusão decorre da observação dos resultados obtidos com a avaliação do recurso de processamento (CPU) e pela metodologia de testes utilizando o TPC-DS e o TPC-H.

Quando é observado o tempo médio de execução com o RD, tende a ser maior que o tempo de execução obtido executando a PrT-PRO para SGBD. A diferença do tempo deve-se ao aproveitamento de dados na memória *cache* do processador. Quando é utilizada a PrT-PRO

e realizado o provisionamento, são adicionados os *cores* de forma sequencial, sendo em todos os experimentos com a PrT-PRO utilizados os mesmos *cores*, ou seja, valor ótimo de *cores*. Aproveitam-se, assim, os estados acumulados no processador, que são instruções e dados de cache. Ao buscar um dado que se encontra na memória cache do processador, o tempo médio de execução diminui consideravelmente, se encontrado na memória L1 do processador. Caso contrário, o tempo médio de execução é maior, devido ao fato de os níveis inferiores de memória serem mais lentos.

O comportamento elástico, oferecido pela PrT-PRO, garante que por meio do monitoramento do SGBD é possível designar o número de recursos necessários para atender a carga de trabalho em execução. Os tempos de resposta demonstrados constatam que existe uma relação entre a quantidade de recursos necessárias para atender uma certa carga de trabalho e o desempenho do SGBD.

Em relação aos trabalhos relacionados, os resultados demonstrados comprovam que a escolha de um provisionamento dinâmico reativo, garantiu um tempo médio de execução da carga de trabalho ideal ao utilizar o provisionamento. Comparado com os trabalhos relacionados, foi possível apresentar que o modelo abstrato de provisionamento de recursos computacionais para SGBD é uma escolha viável para controlar a adição e a remoção de recursos.

Assim provamos nossa hipótese de que, utilizando nossa abordagem baseada em Redes de Petri Predicado/Transição, é possível encontrar um valor ótimo de recursos que atende a demanda do sistema, aumentando os *hits* de *cache* e diminuindo os *misses* de *cache* do processador. Consequentemente, diminuindo os *misses*, o tempo de execução de uma certa carga de trabalho é menor, quando comparado aos resultados obtidos com o uso de todos os recursos disponíveis no *hardware*.

Capítulo 6

Considerações Finais e Trabalhos Futuros

Nesse trabalho, nós apresentamos uma abordagem de provisionamento de recursos computacionais para SGBD baseado em Redes de Petri. Este trabalho decorreu da percepção da necessidade de adaptação do SGBD para características de provisionamento de recursos computacionais. A capacidade de alocar e remover recursos computacionais apresenta benefícios para os SGBD, de forma que, em um momento de uma carga elevada, o sistema seria capaz de alocar dinamicamente os recursos computacionais necessários.

Em particular, o foco deste trabalho foi apresentar uma solução na qual não seria alterado o SGBD para alocar dinamicamente recursos quando necessário, mas sim fornecer os recursos através de um modelo abstrato que controla o provisionamento de recursos computacionais. Esse modelo avalia o SGBD como uma "caixa preta", podendo ser utilizado em qualquer SGBD.

Para a criação do modelo abstrato, foi utilizado o formalismo presente nas Redes de Petri de alto nível denominada Predicado/Transição. Para direcionar o provisionamento de recursos computacionais utilizando a Redes de Petri, foram definidos estados que o SGBD poderia alcançar com base na utilização dos recursos computacionais.

Seguindo esse objetivo, o trabalho elucidou importantes conceitos relacionados ao provisionamento de recursos computacionais em *hardware multi-core* e para SGBD, apresentou soluções existentes para o provisionamento de recursos e descreveu aspectos relacionados à Rede de Petri de baixo nível e alto nível.

Após, foi descrito passo a passo a PrT-PRO para SGBD, apresentando seus estados (lugares) e ações (transições) que controlam a alocação e remoção de recursos. Foi detalhada a implementação do protótipo que instancia e valida através de experimentos a modelagem abstrata.

Por fim, foram apresentados os experimentos realizados demonstrando os benefícios da utilização do provisionamento dinâmico de recursos computacionais para SGBD através dos resultados alcançados.

O objetivo de apresentar um modelo abstrato de provisionamento de recursos computacionais para SGBD foi cumprido de modo que a Rede de Petri apresentada se mostrou eficiente no controle do provisionamento e melhorou o desempenho do SGBD, por meio da alocação

eficiente dos recursos e aproveitamento de *cache do processador*, encontrando uma quantidade de recursos que atendeu a demanda das cargas de trabalho utilizadas nos experimentos. Tal objetivo alcançado conclui que este trabalho possui contribuições relevantes para enriquecer o domínio de pesquisa que abrange, entretanto espera-se que esta iniciativa no futuro seja uma pequena parte das possibilidades de pesquisas que possam surgir envolvendo este assunto.

6.1 Trabalhos Futuros

Os experimentos com a PrT-PRO para SGBD foram executados em um ambiente centralizado *multi-core* inicialmente. Como trabalho futuro, nós planejamos novos experimentos, nos quais a PrT-PRO seja executada em um ambiente distribuído de nuvem computacional.

- A PrT-PRO para SGBD, monitoraria cada nó individualmente: cada nó estaria com a PrT-PRO, executando e controlando os recursos presentes na máquina física. Quando um dos nós não estiver com recursos ociosos, este seria removido e quando necessário adicionado.
- A PrT-PRO para SGBD estaria em um único nó do banco: a PrT-PRO monitoraria todos os nós e controlaria os recursos de todos os nós, avaliando cada nó em um intervalo de tempo.
- A avaliação do uso de recursos dos nós pela PrT-PRO seria pela média de todos os nós presentes na arquitetura distribuída. O provisionamento seria realizado pela criação de mais nós.

Referências Bibliográficas

[sit, 2015a] (2015a). Ec2 amazon auto scaling. http://aws.amazon.com/pt/autoscaling/.

[sit, 2015b] (2015b). Voltdb. http://voltdb.com/.

[Aloini et al., 2012] Aloini, D., Dulmin, R., and Mininno, V. (2012). Modelling and assessing erp project risks: A petri net approach. *European Journal of Operational Research*, pages 484–495.

[Arrieta and García, 2014] Arrieta, M. F. A. and García, J. D. P. (2014). Análisis de rendimiento entre postgresql y sql server usando hammerdb y engine aplicado al sistema acadêmico de conduespoch. Master's thesis, Escuela Superior Politécnica de Chimborazo.

[Barros, 1996] Barros, J. P. (1996). Cppnets: uma classe de redes de petri de alto-nível implementação de um sistema de suporte a sua aplicação e análise. Master's thesis, Universidade Nova de Lisboa.

[Borges, 2008] Borges, G. A. P. (2008). Fluxo de dados em redes de petri coloridas e em grafos orientados a atores. Master's thesis, Universidade de São Paulo.

[Cantú, 1990] Cantú, E. (1990). Uma abordagem para a representação, simulação e impremen-tação de sistemas baseada na rede de petri a objetos. Master's thesis, Universidade Federal de Santa Catarina.

[Cardoso and Valette, 1997] Cardoso, J. and Valette, R. (1997). *Redes de Petri*. Editora Universidade Federal de Santa Catarina.

[Cass, 2015] Cass, S. (2015). The 2015 top ten programming languages. `http://spectrum.ieee.org/computing/software/`. Acessado em 26/03/2016.

[Cecchet et al., 2011] Cecchet, E., Singh, R., Sharma, U., and Shenoy, P. (2011). Dolly: virtualization-driven database provisioning for the cloud. In *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE*, pages 51–62.

[Cong and Martin, 2015] Cong, G. and Martin, K. (2015). Towards adaptive resource allocation for database workloads. In *41st International Conference on Very Large Data Base*, School of Computer Science, University of Waterloo.

[Coulden et al., 2013] Coulden, D., Osman, R., and Knottenbelt, W. J. (2013). Performance modelling of database contention using queueing petri nets. In *ICPE '13 Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, pages 331–334.

[Council, 2015a] Council, T. P. (2015a). Tpc benchmark ds. `http://www.tpc.org/tpcds/`. Acessado em 11/11/2015.

[Council, 2015b] Council, T. P. (2015b). Tpc benchmark h. `http://www.tpc.org/tpch/`. Acessado em 25/11/2015.

[Das, 2011] Das, S. (2011). *Scalable and Elastic Transactional Data Stores for Cloud Computing Platforms*. PhD thesis, UNIVERSITY OF CALIFORNIA.

[Erich et al., 2000] Erich, G., Richard, H., Ralph, J., and John, V. (2000). Padrões de projeto. *Soluções Reutilizáveis de Software*.

[Farias, 2016] Farias, V. A. E. (2016). Uma abordagem para a modelagem de desempenho e de elasticidade para banco de dados em nuvem. Master's thesis, Universidade Federal do Ceará.

[Farias et al., 2014] Farias, V. A. E., Sousa, F. R. C., and Machado, J. C. (2014). Auto escalonamento proativo para banco de dados em nuvem. In *Simpósio Brasileiro de Banco De Dados*, pages 281–287.

[Fernandez, 2002] Fernandez, M. P. (2002). *Provisionamento de Recurso em Arquiterura DIFFSERV para Melhoria da Qualidade de Serviço (QoS)*. PhD thesis, Universidade Federal do Rio de Janeiro.

[Garg et al., 2011] Garg, S. K., Gopalaiyengar, S. K., and Buyya, R. (2011). Sla-based resource provisioning for heterogeneous workloads in a virtualized cloud datacenter. In *Proceedings of the 11th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part I*, ICA3PP'11, pages 371–384, Berlin, Heidelberg. Springer-Verlag.

[Genrich, 1987] Genrich, H. J. (1987). Predicate/transition nets. In *Proceeding Advances in Petri nets 1986, part I on Petri nets: central models and their properties*, pages 207–247.

[Ghanbari et al., 2007] Ghanbari, S., Soundararajan, G., Chen, J., and Amza, C. (2007). Adaptive learning of metric correlations for temperature-aware database provisioning. In *The 4th IEEE International Conference on Autonomic Computing*.

[Kongcharoen and Kruangpradit, 2013] Kongcharoen, C. and Kruangpradit, T. (2013). Autoregressive integrated moving average with explanatory variable (arimax) model for thailand export. In *33rd International Symposium on Forecasting*.

[Maciel et al., 1996] Maciel, P. R. M., Lins, R. D., and Cunha, P. R. F. (1996). Introdução as redes de petri e aplicações. In *X Escola de Computação*.

[Meikel et al., 2007] Meikel, P., Raghunath, O. N., and David, W. (2007). Why you should run tpc-ds:a workload analysis. In *7rd Very Large Database Endowment*, Izmir - Turkey.

[Meira et al., 2014] Meira, J. A., Almeida, E. C., and Traon, Y. L. (2014). A state machine for database non-functional testing. In *IDEAS 2014 : 18th International Database Engineering e Applications Symposium*.

[Meng et al., 2010] Meng, X., Isci, C., Kephart, J., Zhang, L., Bouillet, E., and Pendarakis, D. (2010). Efficient resource provisioning in compute clouds via vm multiplexing. In *Proceedings of the 7th International Conference on Autonomic Computing*, ICAC '10, pages 11–20, New York, NY, USA. ACM.

[Minhas et al., 2012] Minhas, U. F., Liu, R., Aboulnaga, A., Salem, K., Ng, J., and Robertson, S. (2012). Elastic scale-out for partition-based database systems. In *Int. Workshop on Self-managing Database Systems (SMDB)*.

[Murata, 1989] Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580.

[Patterson and Hennessy, 2007] Patterson, D. A. and Hennessy, J. L. (2007). *Computer Organization and Design*. Morgan Kaufmann.

[Penha et al., 2004] Penha, D. O., Freitas, H. C., and Martins, C. A. P. S. (2004). Modelagem de sistemas computacionais usando redes de petri: aplicação em projeto, análise e avaliação. *Pontifícia Universidade Católica de Minas Gerais*.

[Perkusich et al., 1999] Perkusich, M. L. B., de F. Q. V. Turnell, M., and Angelo, P. (1999). Modelagem de banco de dados em tempo-real. In *Simpósio Brasileiro de Banco de Dados*, Paraíba.

[PostgreSQL, 2015] PostgreSQL (2015). Postgresql. `http://www.postgresql.org/`. Acessado em 30/06/2015.

[Rajiv, 2013] Rajiv, M. (2013). A survey onadaptive resource provisioning for read intensive multi-tier applications in the cloud. *International Journal of Application or Innovation in Engineering Management*.

[Refaeilzadeh et al., 2009] Refaeilzadeh, P., Tang, L., and Liu, H. (2009). Cross-validation. In *Encyclopedia of Database Systems*.

[Reis, 2006] Reis, L. G. (2006). Proposta de um mecanismo para alocação otimizada de recursos em domínios diffserv. Master's thesis, Universidade Federal de Uberlândia.

[Rodovalho, 2012] Rodovalho, L. B. V. (2012). Modelagem e simulação de processos biomatemáticos usando redes de petri predicado transição diferenciais: Estudo de caso sobre o vírus hiv e o ciclo de vida do aedes aegypti.

[Rodrigo, 2013] Rodrigo, R. R. (2013). Elasticidade em cloud computing: conceito, estado da arte e novos desafios. *Revista Brasileira de Computação Aplicada*, 5.

[Rodrigues, 2004] Rodrigues, C. L. (2004). Verificação de modelos em redes de petri orientadas a objetos. Master's thesis, Universidade Federal de Campinas.

[Salvi, 2009] Salvi, J. S. (2009). Relacionamentos temporais entre redes de petri e planejamento automático. Master's thesis, Universidade Federal do Paraná.

[Schubert et al., 2011] Schubert, F., Rolim, C. O., and Westphall, C. B. (2011). Aplicação de algoritmos de provisionamento baseados em contratos de nível de serviço para computação em nuvem. In *IX Workshop em Clouds, Grids e Aplicações*, pages 175–187.

[Shivam et al., 2007] Shivam, P., Demberel, A., Gunda, P., Irwin, D., Grit, L., Yumerefendi, A., Babu, S., and Chase, J. (2007). Automated and on-demand provisioning of virtual machines for database applications. In *In Proc. ACM SIGMOD Int. Conf. on Management of Data*.

[Smola and olkopf, 2004] Smola, A. and olkopf, B. S. (2004). A tutorial on support vector regression. *Statistics and Computing*, pages 199–222.

[Sousa et al., 2010] Sousa, F. R. C., Moreira, L. O., and Machado, J. C. (2010). Gerenciamento de dados em nuvem: Conceitos, sistemas e desafios. In *In Proceedings of the Simpósio Brasileiro de Banco de Dados (SBBD)*, pages 101–130, Universidade Federal do Ceará.

[T. J., 1974] T. J., S. (1974). *Simulation using GPSS*. Wiley.

[Treibig et al., 2010] Treibig, J., Hager, G., and Wellein, G. (2010). Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In *Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures*, San Diego CA.

[Vergara et al., 2014] Vergara, G. F., Moura, B., and Araáujo, A. P. F. (2014). Elasticidade em uma infraestrutura de nuvens federadas bionimbuz. *Universidade de Brasília - UnB*.

[Wieers, 2015] Wieers, D. (2015). Dstat. `http://dag.wiee.rs/home-made/dstat/`. Acessado em 10/06/2015.

[Xu et al., 2002] Xu, D., Ioerger, R. V. T., and Yen, J. (2002). Modeling and verifying multi-agent behaviors using predicate/transition nets. In *SEKE '02 Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 193–200.

[Zhang et al., 2010] Zhang, W., Qian, H., Wills, C. E., and Rabinovich, M. (2010). Agile resource management in a virtualized data center. In *Proceedings of the First Joint WOSP/-SIPEW International Conference on Performance Engineering*, WOSP/SIPEW '10, pages 129–140, New York, NY, USA. ACM.

Apêndice A

Consultas do Benchmark TPC-DS Selecionadas para os Experimentos

Query 2

```
with wscs as
(select sold_date_sk
       ,sales_price
 from (select ws_sold_date_sk sold_date_sk
             ,ws_ext_sales_price sales_price
       from web_sales) x
       union all
      (select cs_sold_date_sk sold_date_sk
             ,cs_ext_sales_price sales_price
       from catalog_sales)),
wswscs as
(select d_week_seq,
       sum(case when (d_day_name='Sunday') then sales_price else null end)
           as sun_sales,
       sum(case when (d_day_name='Monday') then sales_price else null end)
           as mon_sales,
       sum(case when (d_day_name='Tuesday') then sales_price else  null
          end) as tue_sales,
       sum(case when (d_day_name='Wednesday') then sales_price else null
          end) as wed_sales,
       sum(case when (d_day_name='Thursday') then sales_price else null
          end) as thu_sales,
       sum(case when (d_day_name='Friday') then sales_price else null end)
           as fri_sales,
       sum(case when (d_day_name='Saturday') then sales_price else null
          end) as sat_sales
from wscs
    ,date_dim
where d_date_sk = sold_date_sk
```

```
 group by d_week_seq)
 select d_week_seq1,
        round(sun_sales1/sun_sales2,2),
        round(mon_sales1/mon_sales2,2),
        round(tue_sales1/tue_sales2,2),
        round(wed_sales1/wed_sales2,2),
        round(thu_sales1/thu_sales2,2),
        round(fri_sales1/fri_sales2,2),
        round(sat_sales1/sat_sales2,2)
 from
 (select wswscs.d_week_seq d_week_seq1,
         sun_sales sun_sales1,
         mon_sales mon_sales1,
         tue_sales tue_sales1,
         wed_sales wed_sales1,
         thu_sales thu_sales1,
         fri_sales fri_sales1,
         sat_sales sat_sales1
  from wswscs,date_dim
  where date_dim.d_week_seq = wswscs.d_week_seq and
        d_year = 1999) y,
 (select wswscs.d_week_seq d_week_seq2,
         sun_sales sun_sales2,
         mon_sales mon_sales2,
         tue_sales tue_sales2,
         wed_sales wed_sales2,
         thu_sales thu_sales2,
         fri_sales fri_sales2,
         sat_sales sat_sales2
  from wswscs,
       date_dim
  where date_dim.d_week_seq = wswscs.d_week_seq and
        d_year = 1999+1) z
 where d_week_seq1=d_week_seq2-53
 order by d_week_seq1;
```

Query 26

```
select i_item_id,
    avg(cs_quantity) as agg1,
    avg(cs_list_price) as agg2,
    avg(cs_coupon_amt) as agg3,
    avg(cs_sales_price) as agg4
from catalog_sales, customer_demographics, date_dim, item, promotion
where cs_sold_date_sk = d_date_sk and
    cs_item_sk = i_item_sk and
    cs_bill_cdemo_sk = cd_demo_sk and
    cs_promo_sk = p_promo_sk and
```

```
    cd_gender = 'F' and
    cd_marital_status = 'W' and
    cd_education_status = 'Primary' and
    (p_channel_email = 'N' or p_channel_event = 'N') and
    d_year = 1998
group by i_item_id
order by i_item_id
limit 100;

select c_customer_id as customer_id,
    c_last_name || ', ' || c_first_name as customername
from customer, customer_address, customer_demographics,
    household_demographics, income_band, store_returns
where ca_city = 'Summit'
    and c_current_addr_sk = ca_address_sk
    and ib_lower_bound >= 62699
    and ib_upper_bound <= 62699 + 50000
    and ib_income_band_sk = hd_income_band_sk
    and cd_demo_sk = c_current_cdemo_sk
    and hd_demo_sk = c_current_hdemo_sk
    and sr_cdemo_sk = cd_demo_sk
order by c_customer_id
limit 100;
```

Query 66

```
select
    w_warehouse_namE,
    w_warehouse_sq_ft,
    w_city,
    w_county,
    w_state,
    w_country,
    ship_carriers,
    year,
    sum(jan_sales) as jan_sales,
    sum(feb_sales) as feb_sales,
    sum(mar_sales) as mar_sales,
    sum(apr_sales) as apr_sales,
    sum(may_sales) as may_sales,
    sum(jun_sales) as jun_sales,
    sum(jul_sales) as jul_sales,
    sum(aug_sales) as aug_sales,
    sum(sep_sales) as sep_sales,
    sum(oct_sales) as oct_sales,
    sum(nov_sales) as nov_sales,
    sum(dec_sales) as dec_sales,
    sum(jan_sales/w_warehouse_sq_ft) as jan_sales_per_sq_foot,
```

```
    sum(feb_sales/w_warehouse_sq_ft) as feb_sales_per_sq_foot,
    sum(mar_sales/w_warehouse_sq_ft) as mar_sales_per_sq_foot,
    sum(apr_sales/w_warehouse_sq_ft) as apr_sales_per_sq_foot,
    sum(may_sales/w_warehouse_sq_ft) as may_sales_per_sq_foot,
    sum(jun_sales/w_warehouse_sq_ft) as jun_sales_per_sq_foot,
    sum(jul_sales/w_warehouse_sq_ft) as jul_sales_per_sq_foot,
    sum(aug_sales/w_warehouse_sq_ft) as aug_sales_per_sq_foot,
    sum(sep_sales/w_warehouse_sq_ft) as sep_sales_per_sq_foot,
    sum(oct_sales/w_warehouse_sq_ft) as oct_sales_per_sq_foot,
    sum(nov_sales/w_warehouse_sq_ft) as nov_sales_per_sq_foot,
    sum(dec_sales/w_warehouse_sq_ft) as dec_sales_per_sq_foot,
    sum(jan_net) as jan_net,
    sum(feb_net) as feb_net,
    sum(mar_net) as mar_net,
   sum(apr_net) as apr_net,
   sum(may_net) as may_net,
   sum(jun_net) as jun_net,
   sum(jul_net) as jul_net,
   sum(aug_net) as aug_net,
   sum(sep_net) as sep_net,
   sum(oct_net) as oct_net,
   sum(nov_net) as nov_net,
   sum(dec_net) as dec_net
 from (
    (select
   w_warehouse_name,
   w_warehouse_sq_ft,
   w_city,
   w_county,
   w_state,
   w_country,
   'AIRBORNE' || ',' || 'GERMA' as ship_carriers,
    d_year as year,
   sum(case when d_moy = 1
     then ws_ext_list_price* ws_quantity else 0 end) as jan_sales,
   sum(case when d_moy = 2
     then ws_ext_list_price* ws_quantity else 0 end) as feb_sales,
   sum(case when d_moy = 3
     then ws_ext_list_price* ws_quantity else 0 end) as mar_sales,
   sum(case when d_moy = 4
     then ws_ext_list_price* ws_quantity else 0 end) as apr_sales,
   sum(case when d_moy = 5
     then ws_ext_list_price* ws_quantity else 0 end) as may_sales,
   sum(case when d_moy = 6
     then ws_ext_list_price* ws_quantity else 0 end) as jun_sales,
   sum(case when d_moy = 7
     then ws_ext_list_price* ws_quantity else 0 end) as jul_sales,
```

```
sum(case when d_moy = 8
  then ws_ext_list_price* ws_quantity else 0 end) as aug_sales,
sum(case when d_moy = 9
  then ws_ext_list_price* ws_quantity else 0 end) as sep_sales,
sum(case when d_moy = 10
  then ws_ext_list_price* ws_quantity else 0 end) as oct_sales,
sum(case when d_moy = 11
  then ws_ext_list_price* ws_quantity else 0 end) as nov_sales,
sum(case when d_moy = 12
  then ws_ext_list_price* ws_quantity else 0 end) as dec_sales,
sum(case when d_moy = 1
  then ws_net_paid * ws_quantity else 0 end) as jan_net,
sum(case when d_moy = 2
  then ws_net_paid * ws_quantity else 0 end) as feb_net,
sum(case when d_moy = 3
  then ws_net_paid * ws_quantity else 0 end) as mar_net,
sum(case when d_moy = 4
  then ws_net_paid * ws_quantity else 0 end) as apr_net,
sum(case when d_moy = 5
  then ws_net_paid * ws_quantity else 0 end) as may_net,
sum(case when d_moy = 6
  then ws_net_paid * ws_quantity else 0 end) as jun_net,
sum(case when d_moy = 7
  then ws_net_paid * ws_quantity else 0 end) as jul_net,
sum(case when d_moy = 8
  then ws_net_paid * ws_quantity else 0 end) as aug_net,
sum(case when d_moy = 9
  then ws_net_paid * ws_quantity else 0 end) as sep_net,
sum(case when d_moy = 10
  then ws_net_paid * ws_quantity else 0 end) as oct_net,
sum(case when d_moy = 11
  then ws_net_paid * ws_quantity else 0 end) as nov_net,
sum(case when d_moy = 12
  then ws_net_paid * ws_quantity else 0 end) as dec_net
  from
       web_sales,
      warehouse,
      date_dim,
      time_dim,
  ship_mode
  where
     ws_warehouse_sk =  w_warehouse_sk
     and ws_sold_date_sk = d_date_sk
     and ws_sold_time_sk = t_time_sk
     and ws_ship_mode_sk = sm_ship_mode_sk
     and d_year = 2000
     and t_time between 8519 and 8519+28800
```

```
        and sm_carrier in ('AIRBORNE','GERMA')
      group by
        w_warehouse_name,
        w_warehouse_sq_ft,
        w_city,
        w_county,
        w_state,
        w_country,
        d_year
   )
union all
    (select
   w_warehouse_name,
   w_warehouse_sq_ft,
   w_city,
   w_county,
   w_state,
   w_country,
   'AIRBORNE' || ',' || 'GERMA' as ship_carriers,
    d_year as year,
   sum(case when d_moy = 1
     then cs_sales_price* cs_quantity else 0 end) as jan_sales,
   sum(case when d_moy = 2
     then cs_sales_price* cs_quantity else 0 end) as feb_sales,
   sum(case when d_moy = 3
     then cs_sales_price* cs_quantity else 0 end) as mar_sales,
   sum(case when d_moy = 4
     then cs_sales_price* cs_quantity else 0 end) as apr_sales,
   sum(case when d_moy = 5
     then cs_sales_price* cs_quantity else 0 end) as may_sales,
   sum(case when d_moy = 6
     then cs_sales_price* cs_quantity else 0 end) as jun_sales,
   sum(case when d_moy = 7
     then cs_sales_price* cs_quantity else 0 end) as jul_sales,
   sum(case when d_moy = 8
     then cs_sales_price* cs_quantity else 0 end) as aug_sales,
   sum(case when d_moy = 9
     then cs_sales_price* cs_quantity else 0 end) as sep_sales,
   sum(case when d_moy = 10
     then cs_sales_price* cs_quantity else 0 end) as oct_sales,
   sum(case when d_moy = 11
     then cs_sales_price* cs_quantity else 0 end) as nov_sales,
   sum(case when d_moy = 12
     then cs_sales_price* cs_quantity else 0 end) as dec_sales,
   sum(case when d_moy = 1
     then cs_net_paid_inc_ship * cs_quantity else 0 end) as jan_net,
   sum(case when d_moy = 2
```

```
    then cs_net_paid_inc_ship * cs_quantity else 0 end) as feb_net,
  sum(case when d_moy = 3
    then cs_net_paid_inc_ship * cs_quantity else 0 end) as mar_net,
  sum(case when d_moy = 4
    then cs_net_paid_inc_ship * cs_quantity else 0 end) as apr_net,
  sum(case when d_moy = 5
    then cs_net_paid_inc_ship * cs_quantity else 0 end) as may_net,
  sum(case when d_moy = 6
    then cs_net_paid_inc_ship * cs_quantity else 0 end) as jun_net,
  sum(case when d_moy = 7
    then cs_net_paid_inc_ship * cs_quantity else 0 end) as jul_net,
  sum(case when d_moy = 8
    then cs_net_paid_inc_ship * cs_quantity else 0 end) as aug_net,
  sum(case when d_moy = 9
    then cs_net_paid_inc_ship * cs_quantity else 0 end) as sep_net,
  sum(case when d_moy = 10
    then cs_net_paid_inc_ship * cs_quantity else 0 end) as oct_net,
  sum(case when d_moy = 11
    then cs_net_paid_inc_ship * cs_quantity else 0 end) as nov_net,
  sum(case when d_moy = 12
    then cs_net_paid_inc_ship * cs_quantity else 0 end) as dec_net,
    from
         catalog_sales,
        warehouse,
        date_dim,
        time_dim,
        ship_mode
    where
       cs_warehouse_sk =  w_warehouse_sk
       and cs_sold_date_sk = d_date_sk
       and cs_sold_time_sk = t_time_sk
       and cs_ship_mode_sk = sm_ship_mode_sk
       and d_year = 2000
       and t_time between 8519 AND 8519+28800
       and sm_carrier in ('AIRBORNE','GERMA')
    group by
       w_warehouse_name,
       w_warehouse_sq_ft,
       w_city,
       w_county,
       w_state,
       w_country,
       d_year
    )
) x
group by
       w_warehouse_name,
```

```
        w_warehouse_sq_ft,
        w_city,
        w_county,
        w_state,
        w_country,
        ship_carriers,
        year
 order by w_warehouse_name
 limit 100;
```

Query 73

```
select c_last_name,
       c_first_name,
       c_salutation,
       c_preferred_cust_flag,
       ss_ticket_number,
       cnt from
       (select ss_ticket_number,
          ss_customer_sk,
          count(*) as cnt
        from store_sales,date_dim,store,household_demographics
        where store_sales.ss_sold_date_sk = date_dim.d_date_sk
        and store_sales.ss_store_sk = store.s_store_sk
        and store_sales.ss_hdemo_sk = household_demographics.hd_demo_sk
        and (date_dim.d_dom between 1 and 3 or date_dim.d_dom between 25
           and 28)
        and (household_demographics.hd_buy_potential = '501-1000' or
         household_demographics.hd_buy_potential = 'unknown')
        and household_demographics.hd_vehicle_count > 0
        and (case when household_demographics.hd_vehicle_count > 0
            then household_demographics.hd_dep_count/
               household_demographics.hd_vehicle_count
            else null
            end)  > 1.2
        and date_dim.d_year in (1999,1999+1,1999+2)
        and store.s_county in ('Richland County','Walker County','Daviess
           County','Barrow County',
                        'Luce County','Ziebach County','Franklin Parish'
                           ,'Fairfield County')
         group by ss_ticket_number,ss_customer_sk) dn,customer
where ss_customer_sk = c_customer_sk
    and cnt between 15 and 20
    order by c_last_name,c_first_name,c_salutation,c_preferred_cust_flag
       desc;
```

Query 84

```
select  c_customer_id as customer_id,
```

```
      c_last_name || ', ' || c_first_name as customername
from customer,
     customer_address,
     customer_demographics,
     household_demographics,
     income_band,
     store_returns
where ca_city          =  'Summit'
     and c_current_addr_sk = ca_address_sk
     and ib_lower_bound   >=  62699
     and ib_upper_bound   <=  62699 + 50000
     and ib_income_band_sk = hd_income_band_sk
     and cd_demo_sk = c_current_cdemo_sk
     and hd_demo_sk = c_current_hdemo_sk
     and sr_cdemo_sk = cd_demo_sk
order by c_customer_id
limit 100;
```

Apêndice B

Consultas do Benchmark TPC-H Selecionadas para os Experimentos

Query 2

```
select
        s_acctbal,
        s_name,
        n_name,
        p_partkey,
        p_mfgr,
        s_address,
        s_phone,
        s_comment
from
        part,
        supplier,
        partsupp,
        nation,
        region
where
        p_partkey = ps_partkey
        and s_suppkey = ps_suppkey
        and p_size = :1
        and p_type like '%TIN''
        and s_nationkey = n_nationkey
        and n_regionkey = r_regionkey
        and r_name = ':3'
        and ps_supplycost = (
                select
                        min(ps_supplycost)
                from
                        partsupp,
                        supplier,
```

```
                        nation,
                        region
                where
                        p_partkey = ps_partkey
                        and s_suppkey = ps_suppkey
                        and s_nationkey = n_nationkey
                        and n_regionkey = r_regionkey
                        and r_name = 'EUROPE'
        )
order by
        s_acctbal desc,
        n_name,
        s_name,
        p_partkey
LIMIT 100;
```

Query 6

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= date '1997-01-01'
        and l_shipdate < cast(date '1997-01-01' + interval '1 year' as date
           )
    and l_discount between 0.05 - 0.01 and 0.05 + 0.01
        and l_quantity < 24
LIMIT 1;
```

Query 11

```
select
        ps_partkey,
        sum(ps_supplycost * ps_availqty) as value
from
        partsupp,
        supplier,
        nation
where
        ps_suppkey = s_suppkey
        and s_nationkey = n_nationkey
        and n_name = 'UNITED STATES'
group by
        ps_partkey having
                sum(ps_supplycost * ps_availqty) > (
                        select
```

```
                                sum(ps_supplycost * ps_availqty) * *
                                    0.0000010000
                        from
                                partsupp,
                                supplier,
                                nation
                        where
                                ps_suppkey = s_suppkey
                                and s_nationkey = n_nationkey
                                and n_name = 'UNITED STATES'
                )
order by
        value desc
LIMIT 1
```

Query 14

```
select
        100.00 * sum(case
                when p_type like 'PROMO%'
                        then l_extendedprice * (1 - l_discount)
                else 0
        end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from
        lineitem,
        part
where
        l_partkey = p_partkey
        and l_shipdate >= date '1996-02-01'
        and l_shipdate < date '1996-02-01' + interval '1' month
LIMIT 1;
```

Query 18

```
select
        c_name,
        c_custkey,
        o_orderkey,
        o_orderdate,
        o_totalprice,
        sum(l_quantity)
from
        customer,
        orders,
        lineitem
where
        o_orderkey in (
```

```
                select
                        l_orderkey
                from
                        lineitem
                group by
                        l_orderkey having
                                sum(l_quantity) > 315
        )
        and c_custkey = o_custkey
        and o_orderkey = l_orderkey
group by
        c_name,
        c_custkey,
        o_orderkey,
        o_orderdate,
        o_totalprice
order by
        o_totalprice desc,
        o_orderdate
LIMIT 100;
```