Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2014

Hugo Miguel de Almeida Oliveira

Alta Disponibilidade e Escalabilidade de um Sistema de Bases de Dados usando Pgpool-II

Hugo Miguel de Almeida Oliveira

Alta Disponibilidade e Escalabilidade de um Sistema de Bases de Dados usando Pgpool-II

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requesitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica de José Manuel Matos Moreira, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president	**Prof. Dr. Joaquim Sousa Pinto** Professor Auxiliar da Universidade de Aveiro
vogais / examiners committee	**Prof. Dr. Alexandre Miguel Barbosa Valle de Carvalho** Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto
	Prof. Dr. José Manuel Matos Moreira Professor Auxiliar da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Gostaria de prestar os devidos agradecimentos a todos os que me ajudaram neste percurso, especialmente à minha família que se uniu pelo meu futuro, cada um a dar um pequeno empurrão para o meu sucesso. Também agradeço à minha namorada e aos meus amigos, por me recordarem a importância de não desistir de concluir os meus objectivos!

Um agradecimento especial ao meu orientador, Professor José Moreira, pela paciência quase infinita e por não ter desistido de mim...

Resumo

Os sistemas de bases de dados relacionais são responsáveis por preservar uma grande parte da informação que temos à nossa disposição, e são a base de inúmeras aplicações que utilizamos diariamente. Todos os dias, utilizadores pesquisam informação, esperando que esta lhes seja rapidamente apresentada.

O objectivo desta dissertação é estudar diferentes abordagens para escalar um sistema de bases de dados e garantir um elevado nível de disponibilidade, para que possa responder a um maior número de pedidos. Esta dissertação também aborda os diferentes componentes de um sistema de bases de dados, e os impactos que cada um deles pode causar na performance do sistema. Para este estudo foram utilizados diferentes ambientes de testes, baseados em PostgreSQL, e utilizando também o software mediador Pgpool-II para implementar replicação de dados. Para avaliar os impactos na performance do sistema foram executados testes de referência seguindo a norma TPC-C.

Abstract

Relational database systems are responsible for preserving a great deal of the information available to us, and are the basis for several applications we use daily. Everyday, users search for information, expecting it to promptly be displayed.

The goal of this dissertation is to study different approaches to scale a relational database system, and to ensure a high level of availability, for it to be able to respond to a greater number of requests. This dissertation also addresses the different components of a database system, and the impacts each can cause on the system's performance. Different test environments were used for this study, based on PostgreSQL, and also using the middleware Pgpool-II to implement data replication. To assess the performance impacts on the system, several TPC-C benchmarks were executed.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

- SQL - Structured Query Language
- OLTP - OnLine Transaction Processing
- OLAP - OnLine Analytical Processing
- DML - Data Modification Language
- DDL - Data Definition Language
- SGBD - Sistema de Gestão de Bases de Dados

Capítulo 1

Introdução

O mundo em que vivemos tem evoluído muito, especialmente nas últimas décadas. Hoje em dia vivemos em plena Era da Informação [3], em que podemos consultar dados sobre praticamente qualquer coisa, a partir de qualquer ponto do globo. No entanto, apesar de a informação poder ser consultada virtualmente, com o recurso a computadores e outros dispositivos, ela tem de ficar persistente em armazenamento físico.

Os desafios relativos à salvaguarda e acesso à informação são cada vez maiores, tendo em conta que há cada vez mais informação e cada vez mais pessoas a procurar ter acesso a ela. É neste contexto que os Sistemas de Gestão de Bases de Dados (SGBD) têm um papel determinante. São estes sistemas que têm de dar resposta aos utilizadores, permitindo que estes acedam aos dados que desejam consultar em tempo útil.

1.1 Conceitos

1.1.1 Comparação entre Instância e Base de Dados

Ao utilizar bases de dados é natural pensar que se consulta uma única entidade, que vai executar o pedido que fizemos. No entanto, uma base de dados é dividida em duas partes: a base de dados (física) e a instância (lógica).

Dependendo da tecnologia, a instância pode ser apenas uma plataforma virtual de acesso aos dados de uma base de dados, ou pode conter várias bases de dados. Por exemplo, em PostgreSQL e Microsoft SQL Server, a instância dá acesso a um número limitado de bases de dados. Já em Oracle, a relação entre instância e base de dados é de um para um, e a divisão entre diferentes contextos é feita recorrendo a "schemas", e não a bases de dados.

Uma instância de bases de dados é um conjunto de estruturas de memória e processos que acedem a um conjunto de ficheiros de dados. Esta camada lógica permite controlar o acesso aos dados, tanto na sua vertente de permissões e pedidos de utilizadores, como numa vertente mais operacional, de gestão de recursos e optimização de tempos de resposta.

1.1.2 Escalabilidade

Quando surge a necessidade de evoluir um sistema, com o objectivo de conseguir processar um maior volume de dados, existem dois tipos de escalabilidade possíveis: horizontal ou vertical [4].

A escalabilidade vertical é normalmente associada a uma melhoria da estrutura existente, sem adicionar mais nós (servidores). A escalabilidade horizontal tem como objectivo adicionar mais nós à estrutura do sistema, nós estes que não tem de ser necessariamente iguais aos restantes.

Tendo em conta o exemplo de um sistema baseado apenas em um servidor, a escalabilidade vertical representa melhorar esse servidor, ou substituí-lo por um com mais recursos, nomeadamente CPU, memória RAM, etc.
Para o mesmo exemplo, escalabilidade horizontal será adicionar um servidor à estrutura, mantendo o servidor original. Esta solução implica algum tipo de mecanismo de distribuição da carga do sistema pelos dois servidores, e se possível, aproveitar as capacidades desse mecanismo para garantir a alta disponibilidade do sistema.

Uma vantagem clara da escalabilidade horizontal é a simplicidade com que se consegue adicionar um novo nó, com indisponibilidade mínima do sistema. Isto acontece porque normalmente não há a necessidade de indisponibilizar o nó original durante muito tempo, apenas copiar os dados relevantes para o segundo nó e configurar o novo nó no software responsável pela distribuição de carga. Já na escalabilidade vertical, melhorar o sistema implica migrar do nó original para o novo nó, tendo necessariamente de haver um período de indisponibilidade maior durante a transição. No entanto, ambas as abordagens são válidas, e têm aplicações práticas nos sistemas actualmente em uso.

1.1.3 Alta Disponibilidade

Em sistemas sem grandes requisitos relativamente a tempos de resposta, em que seja admissível que haja períodos em que não se consiga aceder à base de dados, alta disponibilidade pode ser uma característica supérflua.
No entanto, em sistemas que nunca podem estar indisponíveis, esta é uma característica imprescindível.

Quando um sistema de bases de dados dispõe de Alta Disponibilidade [5], isto quer dizer que este sistema está preparado para se manter activo mesmo em caso de falha grave. Para esta característica ser obtida, é necessário um elevado grau de redundância, tanto ao nível virtual como ao nível físico.

Em termos de redundância ao nível virtual, falamos de duas ou mais instâncias de bases de dados, em servidores diferentes. Assim, na falha de uma instância, uma das restantes toma conta das operações, e garante que o serviço continua. A indisponibilidade, neste caso, apenas acontece durante o tempo de "failover", ou seja, o tempo que demora o controlo do serviço passar de uma instância para a outra. Caso exista um software mediador que torne invísivel a camada de base de dados, como é o caso do Pgpool-II, não existe período de in-

disponibilidade imediato porque o software mediador gere as ligações às instâncias de bases de dados e direciona os pedidos para todas as instâncias disponíveis.

Há várias abordagens que podem conseguir um nível elevado de disponibilidade. Quando existe armazenamento centralizado que pode ser rapidamente alocado a uma instância, pode apenas ser necessário mover os dados em disco, para que a instância que controla o serviço possa ter todo o controlo sobre os dados. No entanto, em abordagens em que o armazenamento é partilhado não temos realmente alta disponibilidade porque temos um ponto único de falha. Nestas condições, se existir um problema ao nível do armazenamento o serviço não pode continuar enquanto este não estiver resolvido.

Pelas razões evidenciadas acima, para garantir alta disponibilidade é necessário ter redundância ao nível da instância, e também ao nível dos dados. Desta forma, existe uma cópia integral sempre pronta a disponibilizar o serviço. O custo principal desta solução é que o armazenamento necessário multiplica-se pelo número de instâncias do serviço em operação.

Podemos ter diversas configurações com alta disponibilidade. Uma configuração possível consiste em manter uma instância activa a sincronizar as restantes, estando estas indisponíveis. Outra possibilidade consiste em manter uma instância activa a sincronizar as restantes, mas estas estão disponíveis apenas para consulta de dados. A última possibilidade é termos várias instâncias activas, tanto para leitura como para escrita. No entanto, esta é uma opção mais rara e que não está disponível na maioria dos sistemas de bases de dados relacionais.

1.1.4 Sincronização

Para garantir alta disponibilidade é necessário que as várias instâncias do mesmo serviço tenham exatamente os mesmos dados. Naturalmente, terá de haver um mecanismo que garanta que qualquer alteração aos dados, ou à estrutura dos objectos da base de dados, seja aplicada a todas as instâncias envolvidas. Este mecanismo é então responsável por sincronizar essas alterações, para que os dados estejam totalmente (ou parcialmente) coerentes entre as instâncias.

Os mecanismos de sincronismo podem ser síncronos ou assíncronos.
Uma replicação de dados síncrona é mais fiável, e garante que os dados estão totalmente consistentes entre as várias instâncias, mas o custo operacional é maior comparativamente a uma replicação assíncrona, porque a operação só fica completa com a confirmação do destino. Caso a rede não seja de grande qualidade, pode não ser viável fazer uma replicação síncrona. Uma replicação de dados assíncrona é naturalmente mais rápida por não exigir a confirmação do destino, mas poderá haver perda de dados, pelo que apenas poderá ser aplicada em projectos que tolerem falhas.

Como sempre, há que pesar performance contra fiabilidade, e só em cada caso podemos medir melhor as vantagens e desvantagens de cada tipo de sincronismo.

1.1.5 Balanceamento de Carga

Numa instância de base de dados única, os dados são guardados apenas num servidor. Podemos conseguir optimizar o acesso aos dados, tendo vários discos independentes para armazenamento, minimizando assim a contenção de I/O [6], pois várias consultas diferentes vão poder aceder a diferentes discos em paralelo. No entanto, os restantes recursos (memória, CPU, ligações à base de dados) continuam a ser partilhados.

Tal como referido anteriormente, podemos ter uma configuração que garanta alta disponibilidade, com uma instância activa e as restantes disponíveis apenas para leitura. Com esta configuração podemos balancear a carga [7], permitindo que as leituras possam ser feitas em qualquer instância. Para que esta implementação seja possível, é necessário um sistema que distribua igualmente as consultas feitas pelas várias instâncias disponíveis. Assim, o acesso para consulta de dados é sempre direcionado a esse sistema, que distribui os pedidos de acordo com a política de distribuição de carga implementada.

Dividir um problema grande em diversos problemas pequenos simplifica o processo, e traz vantagens que não se conseguem obter atacando um problema grande. Assumindo condições ideais, a adição de mais instâncias ao sistema permitiria que a resposta às consultas seja paralelizada pelo número de instâncias disponíveis.
Por exemplo, no caso de termos duas instâncias para leitura e a política de distribuição de carga dividir os pedidos igualmente pelas duas, um grupo de mil consultas seria teoricamente dividido em quinhentas consultas por instância, reduzindo o tempo total de resposta aproximadamente para metade.

1.2 Problema

Tendo um sistema funcional assente em bases de dados num único servidor, como podemos fazer com que este escale para servir mais utilizadores? E como podemos minimizar a indisponibilidade do sistema?

O caso de estudo que temos em análise prende-se com as questões indicadas.
Neste caso, focamo-nos no SGBD PostgreSQL, por ser uma opção gratuita que tem evoluído muito nos últimos anos.
Apesar desta evolução, o PostgreSQL não tem suporte nativo que permita a alta disponibilidade do sistema, nem o balanceamento de carga entre diversas bases de dados.
Estas capacidades podem ser encontradas em software mediador, que utiliza os recursos existentes no PostgreSQL e na rede para permitir resolver uma série de problemas associados ao sistema.

1.3 Objectivo

O principal objectivo desta dissertação é o estudo de diferentes configurações que permitam obter Alta Disponibilidade e Balanceamento de Carga num sistema construído com recurso ao SGBD PostgreSQL. No caso em estudo, assumimos que o sistema existente assenta sobre máquinas com performance baixa, mas de fácil aquisição, pelo que a opção mais

natural é a escalabilidade horizontal. É neste sentido que se pretende estudar configurações de software que permitam distribuir a carga do sistema por vários nós com os mesmos dados, permitindo assim a escalabilidade horizontal do sistema, com recurso a um software mediador de nome Pgpool-II.

Como um sistema de bases de dados precisa de hardware corretamente dimensionado para as suas necessidades, é também objectivo desta dissertação identificar os impactos causados pela configuração de CPU, memória e rede do sistema.
Com a escalabilidade horizontal é também analisada a possibilidade de configurar o sistema para suportar alta disponibilidade, e os benefícios provenientes desta solução.

1.4 Estrutura do Documento

No Capítulo 2 é feita uma apresentação dos problemas e possíveis soluções associadas a este tema, bem como da ferramenta Pgpool-II e dos testes de referência utilizados. No Capítulo 3 é descrito o ambiente de testes configurado, e também os diferentes testes executados para análise de performance do sistema. No Capítulo 4 são apresentados os resultados obtidos pelos testes de referência executados, e a análise dos mesmos. No Capítulo 5 apresentam-se as conclusões do trabalho desenvolvido e abordagens diferentes que podem ser exploradas.

Capítulo 2

Enquadramento

Este capítulo aborda a replicação de dados e a alta disponibilidade, temas principais desta dissertação, bem como os problemas que resolve e os que surgem com a sua implementação num sistema. Também são apresentados softwares mediadores capazes de extender as funcionalidades do PostgreSQL, focando no Pgpool-II, software utilizado nesta dissertação. Para concluir, é apresentado o HammerDB, software utilizado para realizar testes de referência segundo as normas TPC, igualmente apresentadas.

2.1 Trabalhos Recentes

A replicação de dados é um tema muito discutido nos últimos anos [8, 9, 10, 11, 12, 13, 14], principalmente pelo facto de não haver uma solução única que sirva as necessidades de todos os sistemas. A sua implementação tem de estar de acordo com as necessidades e capacidades do sistema em causa. Tipicamente a replicação pode ser configurada para melhorar a performance ou aumentar a disponibilidade de um sistema, existindo uma relação de compromisso entre ambas [8].

A utilização de replicação apenas ao nível de software mediador oferece flexibilidade ao sistema, e evita alterações dispendiosas ao nível das bases de dados [12].
No entanto, esta abordagem implica um conhecimento limitado por parte do software mediador dos dados acedidos pelas transações, e pode resultar em redução de concorrência e/ou aumento de transações abortadas [10].

Num sistema com várias réplicas disponíveis torna-se importante distribuir corretamente a carga, de acordo com as necessidades da aplicação. Esta tarefa é responsabilidade de software, ou hardware, capaz de distribuição de carga. Todos os pedidos de aplicações clientes são direcionados para o mediador para que estes sejam executados em servidores distintos, sendo estes servidores selecionados dinâmicamente com base na política de distribuição implementada [14].

O balanceamento de carga pode ser implementado ao nível da ligação, da transação ou da operação SQL [8].
Com balanceamento ao nível da ligação, todas as transações e pedidos de um cliente são efectuados para o mesmo servidor, até à ligação ser fechada. É uma implementação simples

e dependendo de como a gestão de ligações ao distribuidor é configurada, pode proporcionar uma fraca distribuição, pelo facto de que podem haver clientes que executem um maior número de pedidos do que outros, acabando por sobrecarregar o nó a que estão ligados, e deixando outros com uma carga inferior.
Balanceamento ao nível da transação ou operação SQL é mais preciso do que balanceamento ao nível da ligação, e pode distribuir com maior exatidão a carga pelas várias réplicas. Isto acontece porque o mediador vai distribuir a carga tendo em conta um nível de granularidade mais fino, encarando cada operação individualmente.
É no entanto uma abordagem de mais difícil implementação, e que pode necessitar de cuidados ao nível do código executado, para que as operações possam concluir com sucesso. Têm de haver mecanismos ou metodologias que garantam que a execução de código com dependências entre si não é distribuída pelas várias réplicas, com risco de falhar a execução e de gerar diferenças entre as réplicas.
No caso de existirem várias réplicas com possibilidade de escrita no mesmo sistema, o balanceamento de operações de modificação de dados implica que sejam implementados mecanismos de resolução de conflitos, para que todas as réplicas se mantenham no mesmo estado de consistência. Um bom exemplo de gestão de conflitos é a distribuição das transações para réplicas específicas, mediante o tipo de transação, os seus parâmetros e os conflitos típicos associados [10]. Existem também outras alternativas, como garantir ao nível do código que operações de modificação de dados não são distribuídas, sendo executadas sempre em todas as réplicas, tal como acontece com o software mediador Pgpool-II [15].

Apesar da implementação de replicação poder trazer vantagens a um sistema, e de o mesmo poder tirar partido de funcionalidades como alta disponibilidade e balanceamento de carga, é necessário analisar o efeito da replicação no sistema, e os impactos que surgem.
Por exemplo, a replicação da invocação de "stored procedures"pode implicar um desperdício de recursos quando as mesmas apenas executem operações de seleção de dados, porque a mesma pesquisa é feita por cada nó replicado, apesar de a resposta ser sempre a mesma independentemente do nó que responde.
Outro possível impacto é a recuperação ou manutenção de réplicas. Ao ser removida do sistema, por falha ou por uma operação de manutenção planeada, existe a necessidade de recuperar essa réplica até ao mesmo estado de consistência que as restantes réplicas ainda activas. Esta recuperação pode ser demorada, e pode também envolver a paragem de todo o sistema.
Também o próprio funcionamento da replicação pode causar impactos na performance do sistema. É ainda necessário analisar e testar os impactos directos de mover uma dada carga para um sistema com replicação, comparativamente com a performance da mesma carga numa única base de dados, sem contenção associada.

2.2 Pgpool-II

Existem vários softwares mediadores que permitem extender as capacidades do PostgreSQL, especialmente nos casos da replicação de dados e alta disponibilidade.

Dos projectos activos (Figura 2.1), as opções com mais adesão são Slony, Bucardo e Pgpool-II (evolução do Pgpool-I).
A replicação de dados do Slony [16] é feita com recurso a "triggers"nos objectos das bases de

Program	License	Maturity	Replication Method	Sync	Connection Pooling	Load Balancing	Query Partitioning
PgCluster	BSD	Stalled	Master-Master	Synchronous	No	Yes	No
pgpool-I	BSD	Stable	Statement-Based Middleware	Synchronous	Yes	Yes	No
Pgpool-II	BSD	Recent release	Statement-Based Middleware	Synchronous	Yes	Yes	Yes
slony	BSD	Stable	Master-Slave	Asynchronous	No	No	No
Bucardo	BSD	Stable	Master-Master, Master-Slave	Asynchronous	No	No	No
Londiste	BSD	Stable	Master-Slave	Asynchronous	No	No	No
Mammoth	BSD	Stalled	Master-Slave	Asynchronous	No	No	No
rubyrep	MIT	Stalled	Master-Master, Master-Slave	Asynchronous	No	No	No
BDR (Bi-Directional Replication)	PostgreSQL (BSD)	Beta	Master-Master (no triggers needed)	Asynchronous	No	No	No

Figura 2.1: Comparação entre software mediador para PostgreSQL [1]

---

dados do servidor principal, que replicam de forma assíncrona as alterações para as restantes instâncias (no máximo até 20). As restantes instâncias apenas estão disponíveis para leitura. Tal como o Slony, o Bucardo [17] também recorre a "triggers"para replicar os dados, mas é capaz de ter vários "masters". Todos os "masters"podem sofrer alterações isoladas, sendo depois da responsabilidade do Bucardo a sincronização entre os nós, e a resolução de conflitos que possam surgir pelas diferentes alterações entre os mesmos.

O Pgpool-II [15] foi a escolha para implementar replicação de dados e alta disponibilidade neste estudo. É um software mediador que funciona entre servidores com instâncias em PostgreSQL e a camada aplicacional que se liga a esses servidores de bases de dados.
Este software mediador permite extender as capacidades de servidores com instâncias PostgreSQL, podendo estes tirar partido de funcionalidades como Replicação, Balanceamento de Carga e Recuperação Online, explicadas em detalhe nas próximas secções.

2.2.1 Replicação

Uma das principais e diferenciadoras funcionalidades do Pgpool-II é a replicação ao nível das instruções SQL. Pelo facto de qualquer instrução ser copiada integralmente para os vários servidores, estes conseguem ficar no mesmo estado de consistência.

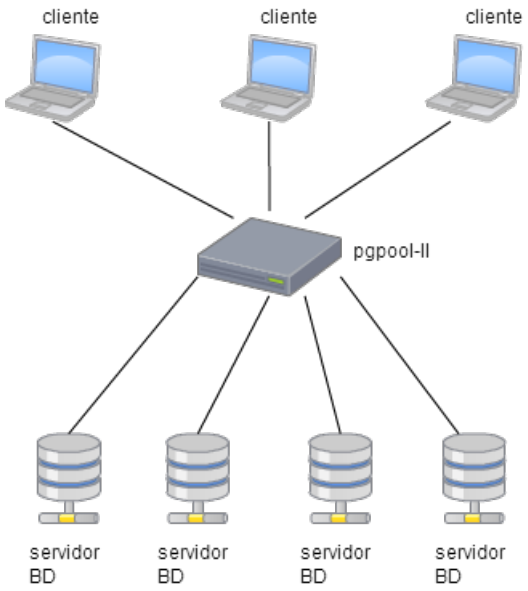

Figura 2.2: Arquitectura de um sistema com Pgpool-II

Tomando em consideração a arquitectura descrita pela Figura 2.2, e assumindo que foi activada a funcionalidade de replicação do Pgpool-II, o modo de funcionamento será a replicação de cada instrução SQL enviada para o Pgpool-II para todos os nós pertencentes a este sistema, independentemente de serem operações de inserção, modificação ou eliminação de dados (instruções DML), ou alteração da estrutura de dados (instruções DDL). No caso de operações de selecção de dados, poderemos configurar o Pgpool-II de forma a replicar ou não as mesmas, com base no parâmetros de configuração "replicate_select". A configuração por omissão é não replicar, fazendo com que todas as operações desse tipo acedam à primeira instância configurada no Pgpool-II (backend0). Caso o parâmetro de configuração seja activado, todas as operações de selecção de dados são replicadas para cada nó.

Todos os nós ficarão num estado de consistência semelhante, e em caso de falha de um deles, os restantes garantem a continuidade do serviço.

O Pgpool-II permite escalar um sistema até 128 nós, o que significa que no máximo cada instrução que chega ao servidor Pgpool-II será replicada 128 vezes. Comparando com uma solução sem Pgpool-II, o custo adicional de replicar para 128 nós será apenas a passagem das instruções pelo Pgpool-II, e o seu desdobramento pelos vários nós. De resto, como o software mediador está a paralelizar uma instrução pelos nós, o tempo total de conclusão da instrução será semelhante à solução sem Pgpool-II, porque cada nó tem exatamente o mesmo grau de esforço. No caso de operações de selecção de dados, a resposta será dada apenas pelo nó a que o cliente foi ligado, sendo as restantes 127 respostas descartadas, representando um esforço inútil.

O Pgpool-II não tem controlo sobre a execução de uma operação nas instâncias, pelo que se uma delas ficar desatualizada face às restantes, não existe nenhuma maneira de corrigir a situação no momento. No entanto, para que exista controlo sobre o estado de sincronização,

cada pedido que passa pelo Pgpool-II devolve feedback ao mesmo sobre a sua conclusão.

No caso de ser detectado que uma operação falhou numa instância e concluiu nas restantes, o Pgpool-II quebra a ligação com a instância em que a falha ocorreu, marcando-a como desatualizada, para que os administradores de bases de dados possam ser alertados para a necessidade de recuperação da mesma. Para que este comportamente ocorra, é necessário que seja activado o parâmetro de configuração "replication_stop_on_mismatch". Caso não esteja activado, as falhas são ignoradas.

2.2.2 Balanceamento de Carga

Para resolver esta situação, o Pgpool-II tem a funcionalidade de balanceamento de carga. Quando activada, esta irá alterar o comportamento do Pgpool-II relativamente a pesquisas. Em vez de replicar uma pesquisa para todos os nós disponíveis no sistema, o Pgpool-II passa a balancear a carga, escolhendo um único nó para processar a pesquisa. Assim, o esforço total é menor, e mais direcionado a um nó apenas.

Desta forma, cada nó vai trabalhar menos, e em vez de terem um esforço desnecessário, podem processar no mesmo intervalo de tempo um número maior de pesquisas. Por exemplo, para um sistema com 4 instâncias, e assumindo que temos 4 pesquisas para processar sequencialmente que demoram 10 segundos cada, sem o balanceamento de carga activo o tempo total de execução será de 40 segundos. Com balanceamento de carga activo, cada nó processa uma operação, e teoricamente, o tempo total de execução passaria a ser 10 segundos.

Pelo facto de o pgpool-II interpretar operações SQL, para controlar que operações podem ser replicadas ou balanceadas é necessário ter cuidados ao nível do código SQL e/ou da configuração do Pgpool-II. Para definir que uma operação não deve ser balanceada, o comentário "/*NO LOAD BALANCE*/"deve ser acrescentado antes da operação "SELECT"[18].
Também se pode controlar o comportamento da replicação e balanceamento de funções SQL recorrendo à white_funcion_list e à black_function_list. A white_funcion_list é uma lista de funções que não fazem alterações na base de dados. Funções incluídas nesta lista são sempre replicadas e/ou distribuídas, mediante a configuração do Pgpool-II.
Pelo contrário, a black_function_list é uma lista de funções que fazem alterações na base de dados. Funções incluídas nesta lista nunca são replicadas e/ou distribuídas, mediante a configuração do pgpool-II.

Apenas uma das listas pode ser utilizada para estabelecer o comportamento pretendido. Tendo balanceamento de carga configurado, as funções que alterem dados devem ser aplicadas a todos os nós, e por esta razão a black_function_list deve incluir este tipo de funções.

2.2.3 Recuperação Online

Apesar de uma arquitectura com Pgpool-II permitir minimizar o tempo de indisponibilidade de um sistema, é necessário ter um plano para quando alguma das instâncias falhar. Para recuperar uma instância até ao estado de consistência das restantes é necessário restaurar as bases de dados a partir de cópias de segurança das bases de dados de uma das

instâncias em funcionamento.

Para que não sejam feitas modificações aos dados enquanto se recupera uma instância seria necessário parar o serviço. Fazer cópias de segurança completas das bases de dados, copiá-las para a instância a recuperar e aplicá-las pode ser tarefa demorada, dependendo do tamanho das bases de dados. Parar todo o serviço por esse tempo pode ser impossível dependendo da criticidade do serviço.

Para contornar este problema o Pgpool-II tem a funcionalidade de Recuperação Online. Esta permite fazer a recuperação de uma instância em duas fases, a primeira maior mas sem indisponibilidade, a segunda mais curta e com indisponibilidade do serviço. Recuperação Online tira partido de uma funcionalidade nativa do PostgreSQL, Point-In-Time-Recovery (PITR) [19]. PITR permite recuperar uma base de dados até a um período em específico, aplicando a informação guardada no Write-Ahead Log (WAL) da base de dados depois da aplicação de uma cópia de segurança completa da mesma.

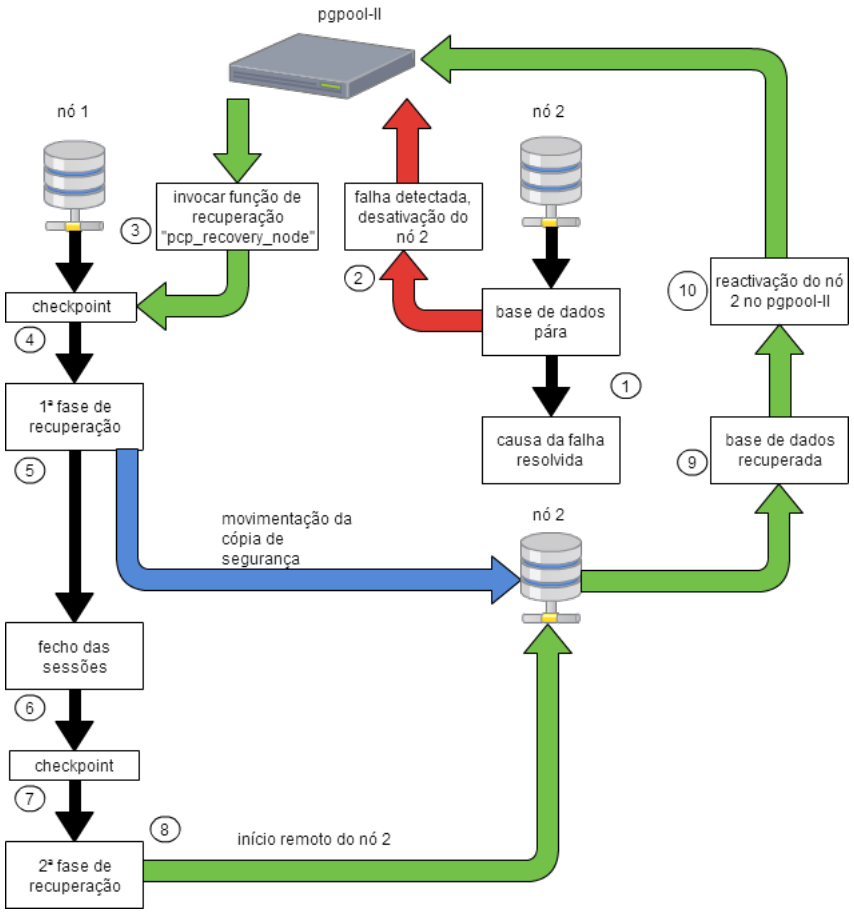

Figura 2.3: Recuperação online

Aproveitando as capacidades do PITR, a Recuperação Online pode ser configurada para

a recuperação de outra instância a partir de uma em funcionamento. Assim, numa primeira fase é feito um "checkpoint", seguido de uma cópia de segurança completa das bases de dados, que pode ser feita sem indisponibilidade, e em seguida a sua cópia para a instância a recuperar. Após fechar as ligações à instância e de esperar que as existentes se desliguem, é forçado um novo "checkpoint"e o backup do WAL para disco.

Em seguida, é invocada remotamente a recuperação que irá substituir os dados da instância pelos dados copiados com a cópia de segurança. O PostgreSQL detecta que é necessário recuperar até um certo período e para isso aplica o backup do WAL guardado na instância fonte. No fim da operação a instância a recuperar volta à configuração do Pgpool-II, e a instância fonte que tinha fechado as ligações volta a estar disponível.

2.3 Testes de Referência

Para avaliar a performance de um sistema devem ser executados testes de referência, seguindo normas bem definidas, que simulam ambientes reais e a carga associada ao seu funcionamento. Algumas entidades no mundo definem normas referentes aos mais diversos tipos de sistemas, sendo que é focada em seguida a entidade responsável pelas normas associadas a testes de referência de bases de dados.

2.3.1 Normas TPC

Transaction Processing Performance Council (TPC) [20] é a organização mundial que define as normas relativamente a testes de referência de bases de dados e publica informação fiável sobre os mesmos para o público. Os membros desta organização são empresas de grande nível no campo da tecnologia, como Microsoft, Oracle, SAP, IBM, HP, entre outras [21].

Todos os membros podem influenciar e ajudar a desenvolver e evoluir as normas TPC [22]. É desta união de vontades entre empresas concorrentes que nascem normas genéricas que permitem avaliar a performance de sistemas. A aplicação destas normas para avaliar e levar ao limite sistemas muito avançados e robustos permite evoluir a tecnologia, que mais tarde chega ao consumidor doméstico.

Da mesma forma que muitos fabricantes de automóveis participam na Fórmula 1 para testar o carro mais rápido que conseguem produzir, as empresas de tecnologia também fomentam a competição entre elas com os testes de referência TPC para obter os melhores resultados possíveis com os melhores sistemas que conseguem desenvolver.

Não é necessário ser membro do TPC para executar um teste de referência seguindo uma qualquer das normas disponíveis. No entanto, para que os resultados desses testes possam ser publicados é necessário a aprovação dos mesmos pelo TPC.

Ao longo dos anos foram criados diferentes normas e naturalmente alguns foram-se tornando obsoletos. Actualmente estão em vigor as normas TPC-C, TPC-DS, TPC-E,TPC-H, TPC-VMS, TPC-HS e TPC-Energy [23]. TPC-C e TPC-E estão orientados a aplicações OLTP, caracterizadas por um elevado número de transações curtas, como sistemas de venda de produtos. TPC-H e TPC-DS estão ligados a aplicações OLAP, que se caracterizam por um

número reduzido de operações de elevada complexidade e duração, como sistemas de suporte à decisão. TPC-VMS é um teste de referência utilizado para avaliar sistemas virtualizados, executando uma das quatro normas anteriores em três máquinas virtuais exatamente iguais. TPC-HS pretende avaliar sistemas orientados para tecnologia "Big Data", como Hadoop. Por último, a norma TPC-Energy é utilizada para avaliar o consumo de energia do hardware do sistema em testes.

No campo dos testes de referência para sistemas OLTP, apesar de a norma TPC-E ser mais recente e complexa, ainda é pouco adoptada (apenas foram publicados até ao momento resultados com Microsoft SQL Server). A norma TPC-C é mais antiga e foi adoptada por um maior número de fabricantes ao longo dos anos.
A norma TPC-C é definida por uma mistura entre operações de pesquisa e operações de alteração de dados intensivas que simulam as operações presentes em aplicações OLTP complexas [24]. As operações são:

- **Nova Compra -** Consiste em registar numa única transação uma nova compra. É a operação mais relevante do teste de referência, gerando uma carga elevada que reflete o quotidiano de ambientes produtivos.
- **Pagamento -** Altera o crédito do cliente e reflete o pagamento ao nível do distrito e armazém.
- **Entrega -** Representa o processamento de várias compras (até um máximo de dez) numa transação.
- **Estado da Compra -** Retorna o estado da última compra feita por um cliente.
- **Nível de Stock -** Retorna o número de produtos recentemente vendidos com stock abaixo de um limite definido

Os objectos de base de dados utilizados pela norma incluem nove tabelas: WAREHOUSE, DISTRICT, CUSTOMER, HISTORY, NEW-ORDER, ORDER, ORDER-LINE, ITEM E STOCK. A métrica de performance definida pela norma TPC-C é a tpmC, que representa as transações por minuto por teste ("transactions-per-minute-C"). Esta métrica é única e qualquer implementação da norma TPC-C não auditada pelo TPC não pode usar o termo tpmC.

2.3.2 HammerDB

HammerDB [25] é uma ferramenta de testes de carga, de código aberto, que suporta uma grande variadade de SGBDs, incluindo PostgreSQL. Permite executar testes de carga transacionais, baseando-se nas normas TPC-C e TPC-H.

Ao implementar versões muito aproximadas das normas TPC, o HammerDB consegue obter um nível de fiabilidade muito elevado, podendo executar testes independentes da plataforma, mas com resultados comparáveis.

Uma das garantias dadas pela ferramenta [2] é a consistência dos resultados entre testes nas mesmas condições, ou seja, se o um teste for executado várias vezes para a mesma configuração, os resultados serão os mesmos, com uma margem de 1%. Variações acima desta

margem indicam uma configuração de hardware deficiente, com possíveis problemas de performance que impedem a fiabilidade do sistema.

No entanto, os resultados produzidos pelo HammerDB não podem ser comparados com resultados oficiais auditados pelo TPC, tendo em conta que é uma implementação de baixo custo com o objectivo de permitir a execução de testes de carga em qualquer sistema.

No contexto do HammerDB, a implementação da norma TPC-C implementa um sistema capaz de processar compras de clientes. A arquitectura da norma identifica uma empresa que vende 100000 tipos de produtos, guardando o stock em armazéns. Cada armazém serve um distrito de vendas e cada distrito serve 3000 clientes.

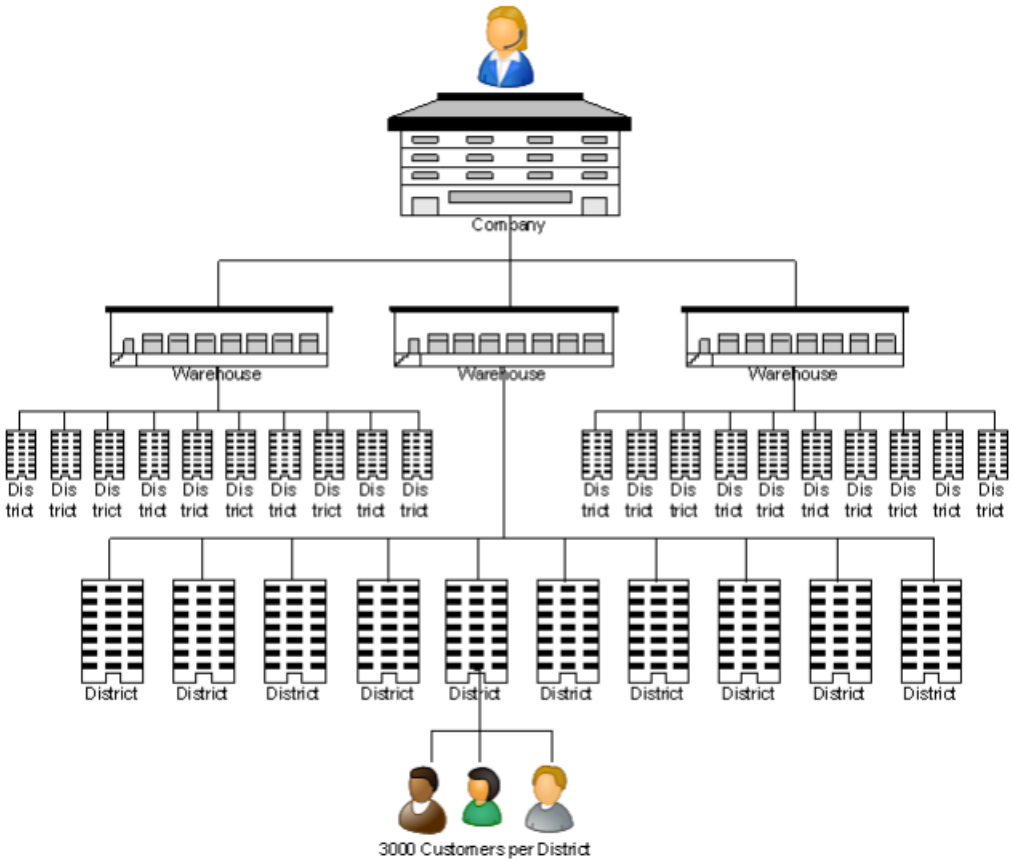

Figura 2.4: Arquitectura do sistema implementado pela norma TPC-C [2]

A carga de trabalho efectuado por um teste TPC-C é distribuída tendo em conta os valores da Tabela 2.1.

Operação	Valor
Nova Compra	45%
Pagamento	43%
Entrega	4%
Estado da Compra	4%
Nível de Stock	4%

Tabela 2.1: Distribuição de Carga de um teste TPC-C

O HammerDB tem 2 métricas definidas, TPM (Transactions per minute) e NOPM (New Orders per minute) [2]:

- **TPM -** Métrica transacional, que consiste no número de commits mais o número de rollbacks realizados durante a execução de um teste.
- **NOPM -** Métrica do próprio teste, independente de plataforma, que identifica o número de novas compras por minuto. Como a métrica TPM depende da forma como cada SGBD trata as transações, os resultados podem variar entre diferentes SGBD's, sendo que a métrica mais fiável para comparar entre plataformas é a NOPM.

2.4 Síntese

Existem diversas abordagens para implementar replicação e alta disponibilidade. A escolha da implementação mais adequada depende das características do sistema a melhorar, como o tipo de operações, a carga diária, o tempo disponível para manutenção. No entanto a própria implementação de replicação introduz custos. Ainda é necessário um maior conhecimento sobre a degradação de performance introduzida com a replicação, face ao mesmo sistema sem replicação [8].

O Pgpool-II é capaz de replicar os dados e de gerir nativamente o balanceamento de carga, entre outras funcionalidades. De todas as opções para o PostgreSQL é a mais completa actualmente disponível, pela maturidade e número de funcionalidades disponíveis. Por esta razão foi o software mediador escolhido para este estudo.

A norma TPC-C é mais antiga e foi adoptada por um maior número de fabricantes ao longo dos anos, sendo por estas razões a norma escolhida para os testes de referência deste estudo, executados com recurso ao HammerDB.

Capítulo 3

Implementação

Este capítulo apresenta as diferentes arquitecturas do ambiente de testes utilizado para executar os testes de referência, e a sua configuração. São também apresentados os diferentes tipos de testes executados. Os testes procuram identificar os impactos na performance performance do sistema associados à utilização de diferentes configurações do Pgpool-II, que garantam a escalabilidade horizontal do sistema e a sua alta disponibilidade.

Também é avaliado o impacto na performance do sistema para diferentes valores de memória disponíveis, para uma arquitectura com as máquinas virtuais de bases de dados em servidores diferentes para isolar o CPU, e para uma arquitectura em que todas as máquinas virtuais residem no mesmo servidor, evitando o uso da rede.

3.1 Estrutura do Ambiente de Testes

O ambiente de testes necessário para este estudo tem alguns requisitos muito particulares, que para serem resolvidos precisaram de uma infraestrutura mais controlada. Por estas razões, a arquitectura utilizada envolve duas máquinas, um computador de secretária ("Desktop") e um computador portátil ("Laptop")

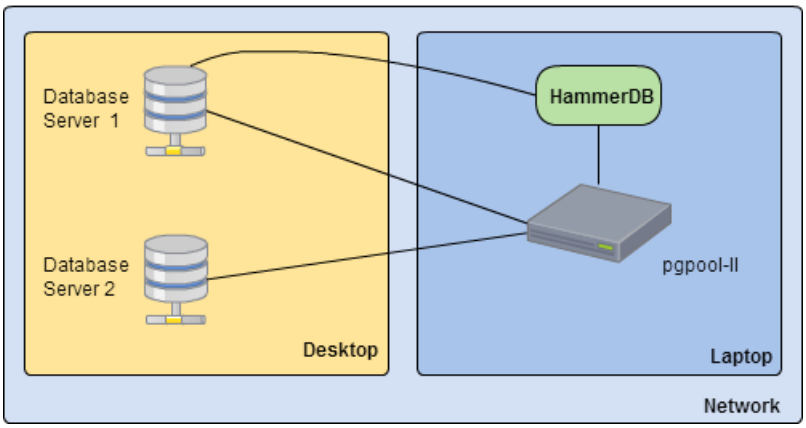

Figura 3.1: Ambiente de Testes

A Figura 3.1 ilustra a arquitectura do ambiente de testes utilizado. No computador "Desktop"foram configuradas duas máquinas virtuais idênticas, cada uma contendo a sua instância PostgreSQL. No computador "Laptop"foi configurada uma máquina virtual, em que foi instalado o Pgpool-II e o PgpoolAdmin, uma ferramenta que facilita a administração e configuração do Pgpool-II. Ainda no computador "Laptop", foi instalado o HammerDB, para execução dos testes.

As duas máquinas encontram-se na mesma rede wireless. O HammerDB foi instalado no "Laptop"para que qualquer execução de testes de referência seja sempre feita pela rede, quer sejam testes directos a uma das máquinas virtuais PostgreSQL, quer sejam testes orientados à máquina virtual com Pgpool-II. Desta forma, a entropia da rede, apesar de presente e de não haver infraestrutura disponível para corrigir esta situação, é semelhante em todos os tipos de testes.

A configuração das máquinas virtuais foi pensada com o intuíto de minimizar ao máximo a partilha de recursos da máquina. Neste caso, existem 3 componentes essenciais: memória RAM, discos e CPU. A falta ou partilha de qualquer um destes recursos pelas duas máquinas virtuais podem causar problemas ao sistema, o que podem levar a impactos nos testes.
Mais especificiamente, quando uma instância recebe uma operação SQL, o trabalho feito por esta para dar a resposta é maioritariamente feito em memória. Para isto, é analisado o plano de execução da operação, e o próprio motor de base de dados verifica onde está fisicamente a informação necessária. Os blocos de disco necessários são assim copiados para páginas em memória, para que a operação possa ser processada. Analisando este fluxo, podemos perceber que se houver pouca memória disponível, o motor de base de dados terá de estar constantemente a limpar dados não usados da memória, para poder colocar informação relevante.

Também é de notar que apesar de poderem ter volume para guardar muita informação, os discos têm sempre canais de comunicação limitados para o acesso ao mesmo. Quando temos um fluxo de acesso ao disco demasiado elevado para os canais disponíveis, encontramo-nos perante contenção de I/O ("input"/"output") [6].
Perante este processo, o motor de base de dados poderá gerar uma carga muito elevada para o CPU, já que todas as operações são invocadas pelas "threads"de CPU associadas ao motor de base de dados.

Qualquer um dos três componentes identificados pode causar degradação de performance no sistema se não estiverem configurados de acordo com a carga prevista para o funcionamento do mesmo. Para minimizar estes riscos procurou-se isolar os recursos de cada máquina virtual, de acordo com as possibilidades da infraestrutura. Mais especificamente, as máquinas virtuais de base de dados foram configuradas com 2GB de memória RAM cada, dos 8GB disponíveis na máquina onde estão alojadas.
Para evitar problemas de contenção de I/O, cada máquina virtual tem o seu próprio disco, evitando assim que outros processos acedam a esses discos e congestionem o acesso aos mesmos.

Relativamente ao CPU, cada máquina virtual utiliza o máximo de núcleos disponíveis na máquina hospedeira em que estão alojadas, 4 núcleos no "Desktop"e 2 núcleos no "Laptop". Pelo facto de não existirem dois servidores idênticos para isolar as máquinas virtuais de forma a não partilharem CPU, este torna-se o ponto crítico do sistema.

3.2 Configurações de Software

Software	Versão
Pgpool-II	3.3.3
PgpoolAdmin	3.3.1
PostgreSQL	9.3
CentOS	6.5
HammerDB	2.16
VMWare Player	6.0.3

Tabela 3.1: Versões do software utilizado

Tal como identificado na Tabela 3.1, as máquinas virtuais foram construídas com recurso ao software de virtualização VMWare Player e à distribuição Linux CentOS. No caso do VMWare Player, a escolha deveu-se à sua facilidade de configuração. No caso do CentOS, a escolha deveu-se à compatibilidade com o Pgpool-II. Apesar de a documentação do Pgpool-II indicar que este funciona na maioria das distribuições Linux, é dada alguma preferência ao CentOS [26].

Relativamente ao PostgreSQL, foi instalada a versão 9.3, última versão estável disponibilizada pela EnterpriseDB, recorrendo ao seu instalador gráfico. Face à configuração por omissão presente no ficheiro *postgresql.conf*, presente no Apêndice A, foram alterados os seguintes parâmetros:

- listen_addresses = '*'
- max_connections = 1000
- shared_buffers = 32MB

O parâmetro "listen_addresses"identifica quais os endereços IP a que a instância de base de dados deve estar à escuta, "max_connections"indica o número máximo de ligações concorrentes à instância e "shared_buffers"indica o valor disponível de memória partilhada da instância, o qual foi definido com 32MB por ser cerca de 25% do tamanho ocupado pela base de dados de teste criada pelo HammerDB. Limitar a memória disponível obriga assim a que a instância não possa trabalhar quase sempre em memória, forçando o acesso a disco, que representa o comportamento normal de uma base de dados [27, 28].

O HammerDB não tem uma configuração inicial complexa, pelo que o devido foco será dado na configuração dos testes na próxima secção.

Após o descarregamento do Pgpool-II para a máquina virtual correspondente, foram executados pela linha de comandos do sistema operativo um conjunto de instruções (Apêndice B), para instalar e configurar o Pgpool-II e PgpoolAdmin [29]. Estes passos garantem as permissões e configurações necessárias para que o processo Apache possa aceder aos executáveis (em "/usr/bin") do Pgpool-II e aos ficheiros de configuração (em "/etc/pgpool-II"). No fim, é reiniciado o serviço do Apache para que as alterações se evidenciem. Como o PgpoolAdmin é uma ferramenta que funciona via web browser, o processo Apache tem que estar activo para

permitir a interação.

Após todos estes passos, o Pgpool-II está pronto a ser configurado via PgpoolAdmin, de acordo com as necessidades. O PgpoolAdmin permite editar o ficheiro de configuração do Pgpool-II ("/etc/pgpool-II/pgpool.conf") e controlar o serviço. O serviço pode ser iniciado ou parado via browser, podendo também invocar operações mais complexas como a recuperação de um nó ou a remoção do serviço. Também permite identificar visualmente o estado de cada nó, para que em caso de falha se possa actuar.

As Figuras 3.2, 3.3 e 3.4 evidenciam as configurações no PgpoolAdmin relevantes para o estudo, bem como a visualização do estado do serviço na Figura 3.5.

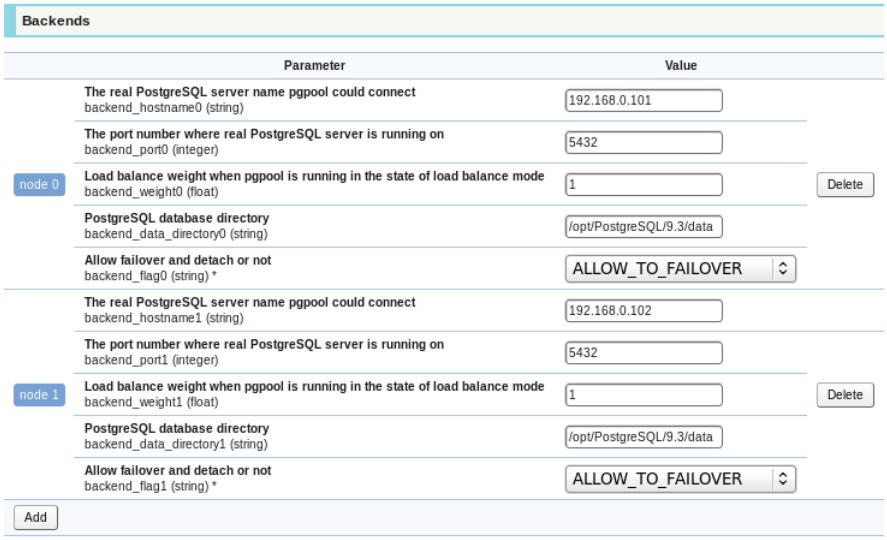

Figura 3.2: Configuração das Instâncias do serviço no PgpoolAdmin

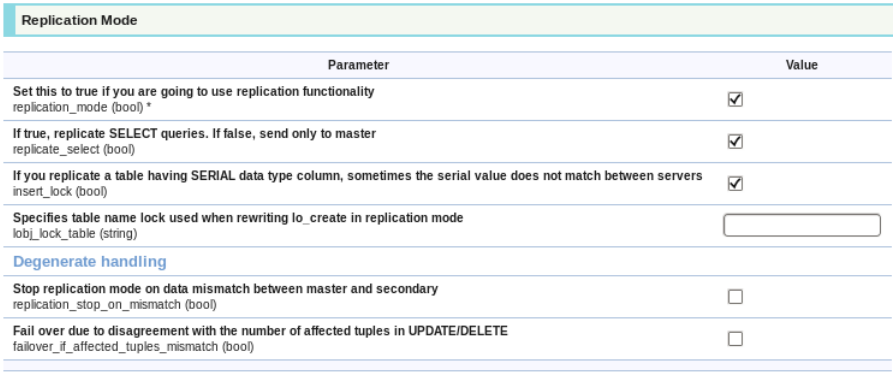

Figura 3.3: Configuração de Replicação no PgpoolAdmin

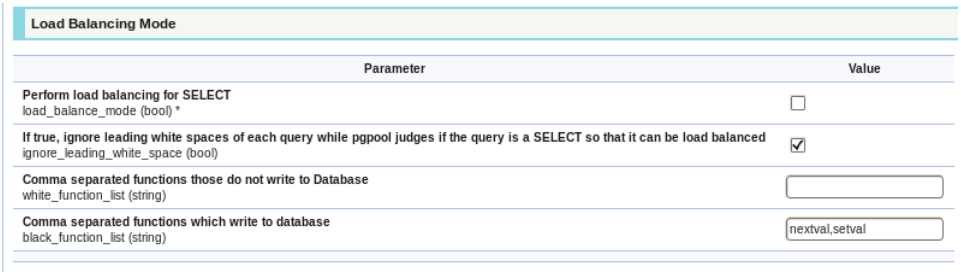

Figura 3.4: Configuração de Balanceamento de Carga no PgpoolAdmin

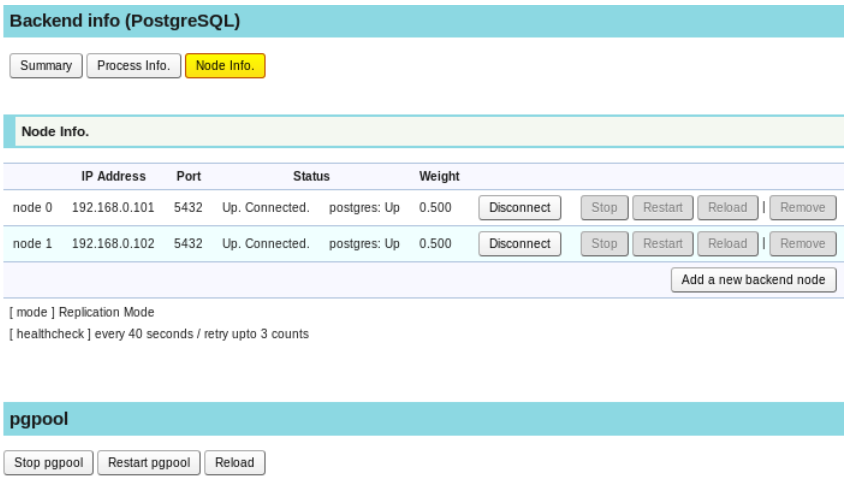

Figura 3.5: Estado do Serviço no PgpoolAdmin

3.3 Plano de Testes

Com recurso ao HammerDB, foram configurados e executados diversos testes seguindo a norma TPC-C, para quantificar a possível degradação de performance existente na utilização do Pgpool-II como ponte entre o cliente e o servidor de bases de dados.
Para todos os testes, o Pgpool-II foi configurado para um dos seguintes modos de funcionamento:

- directo
- pgpool
- pgpool replicate_select
- load balance

Tendo em consideração a Figura 3.1, o modo "directo"representa um teste feito a partir do HammerDB directamente ao "Database Server 1". O modo "pgpool"representa testes em que o destino é o endereço e porto (9999) do Pgpool-II, e este tem a configuração "replicate_mode"activada, para permitir a replicação. O modo "pgpool replicate_select"também

identifica testes direcionados ao Pgpool-II, mas além da configuração "replication_mode", também a configuração "replicate_select"está activada, para forçar a replicação de operações de pesquisa. O modo "load_balance"define testes direcionados ao Pgpool-II, com as configurações "replication_mode"e "load_balance_mode"activadas, para que as operações de pesquisa sejam distribuídas pelos nós disponíveis.

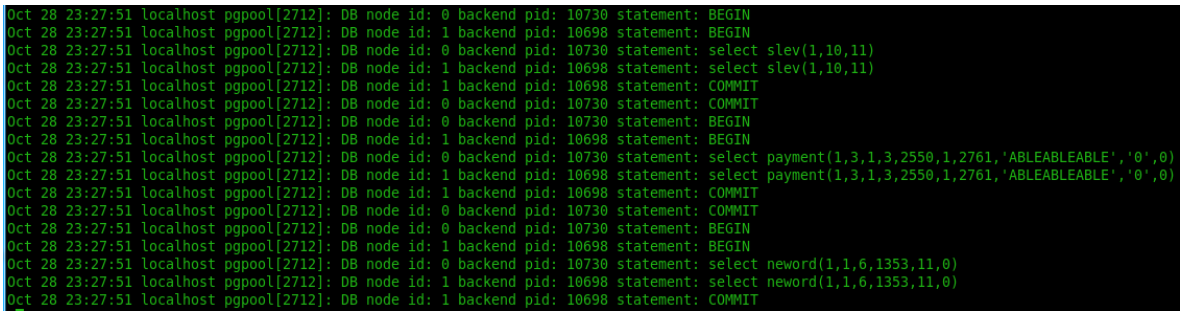

Figura 3.6: Comportamento com replicate_select activo

Os testes efectuados com "replicate_select"activo são os que identificam o funcionamento da replicação com um teste HammerDB. Sem "replicate_select"activo foi identificado que as alterações e inserções de dados não eram replicadas, pelo facto de o HammerDB apenas executar funções, sendo que muitas destas funções fazem alterações de dados. Como podemos concluir pela Figura 3.6, com este parâmetro activado as operações "SELECT"são replicadas para os diferentes nós. Para este exemplo, com dois nós configurados, cada operação é invocada no "DB node id: 0"e no "DB node id: 1". Caso "replicate_select"não estivesse configurado, as operações "SELECT"iriam apenas para o priemiro nó configurado, o "DB node id: 0".
Assim sendo, sem "replicate_select"activo temos um teste que apenas evidencia a degradação de performance da passagem das operações pelo Pgpool-II, e com "replicate_select"testamos efectivamente o comportamento e a performance associadas à replicação das operações SQL.

O aumento do número de armazéns aumenta o tamanho da base de dados de teste e faz com que haja maior diversidade nas operações invocadas, causando uma carga mais elevada. Também o aumento do número de utilizadores virtuais concorrentes aumenta o nível de carga do sistema. Tendo em conta que um armazém consegue servir até 3000 clientes, é pouco relevante aumentar o número de armazéns se não se conseguir utilizar mais de 3000 utilizadores (Secção 2.3.2). Por esta razão, para todos os testes foi utilizado um armazém, e o número de utilizadores virtuais concorrentes varia para produzir diferentes níveis de carga no sistema.

Antes de iniciar os testes é necessário criar os objectos de teste (Figura C.1). Após a criação da base de dados de testes "tpcc", o tamanho total é de 124MB. Tomando em consideração que cada teste executado vai escrever na base de dados, produzindo alterações e aumentando o tamanho da mesma, foi decidido que a base de dados de testes "tpcc"é destruída e criada novamente após a conclusão dos vários testes definidos para um modo de funcionamento.

Os testes planeados têm como objectivo quantificar a performance do sistema para diferentes configurações, procurando também isolar o efeito provocado por componentes específicas, como a qualidade da rede, a quantidade de memória da instância e a partilha de CPU:

- **Primeiro grupo de testes -** Pretende avaliar a degradação de performance introduzida pela utilização do Pgpool-II comparativamente com o sistema sem Pgpool-II.
- **Segundo grupo de testes -** Pretende avaliar o impacto na performance causado pela rede.
- **Terceiro grupo de testes -** Pretende avaliar o impacto na performance de diferentes volumes de memória configurados para uma instância de base de dados.
- **Quarto grupo de testes -** Pretende avaliar o impacto na performance causado pelo CPU.
- **Quinto grupo de testes -** Pretende avaliar a performance do sistema obtida utilizando balanceamento de carga.

Os primeiro grupo de testes analisa os modos de funcionamento "directo", "pgpool"e "pgpool replicate_select". Foram executados seis testes por modo de funcionamento, com a duração de trinta minutos (Figura C.2), em que o número de utilizadores virtuais foi 1, 2, 4, 8, 10 e 20. No caso de testes com um intervalo de tempo definido, tal como os executados, é necessário configurar um utilizador adicional como monitor do teste. Sendo assim, e tendo em conta os números anteriores, os testes foram configurados com 2, 3, 5, 9, 11 e 21 utilizadores concorrentes.
O mesmo plano de testes foi repetido para uma arquitectura de sistema diferente, em que a máquina virtual do Pgpool-II está na mesma máquina física que as restantes máquinas virtuais de bases de dados, para evidenciar o impacto da rede nos testes.

O segundo grupo de testes é idêntico ao primeiro, mas executado num ambiente com a arquitectura apresentada na Figura 3.7, em que ambas as instâncias e bases de dados e também a instância do pgpool-II estão no mesmo servidor, minimizando a influência da rede nos resultados, sendo apenas a sua invocação feita pelo HammerDB, alojado no "Laptop".

O terceiro grupo de testes executa testes directos a uma instância, fixando o número de utilizadores concorrentes e variando a memória disponível para a instância de bases de dados, através do parâmetro de configuração do PostgreSQL "shared_buffers". O impacto na performance do sistema é analisado não só pelas métricas TPM e NOPM, mas também através da utilização média do CPU e do valor médio de MB/s escritos e lidos em disco durante a execução de cada teste.

Para o quarto grupo de testes foi alterada novamente a arquitectura do sistema, de acordo com a Figura 3.8, para que cada máquina virtual de abses de dados fique num servidor diferente, atribuindo assim um CPU dedicado a cada uma. O impacto na performance do sistema é analisado apenas no modo "pgpool replicate_select"pelo facto de este modo de funcionamento introduzir a mesma carga em cada instância.

Pelo facto de o balanceamento de carga do Pgpool-II ser feito ao nível da sessão, o quinto grupo de testes não pode utilizar o teste de referência TPC-C do HammerDB para quantificar

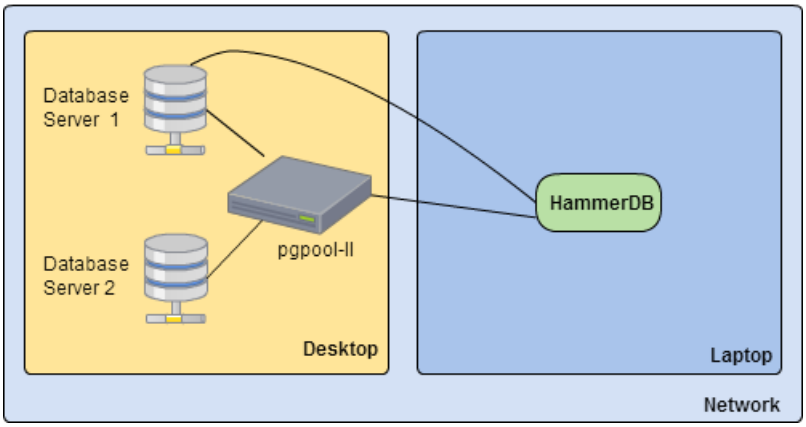

Figura 3.7: Ambiente de Testes 2 - máquinas virtuais apenas no Desktop

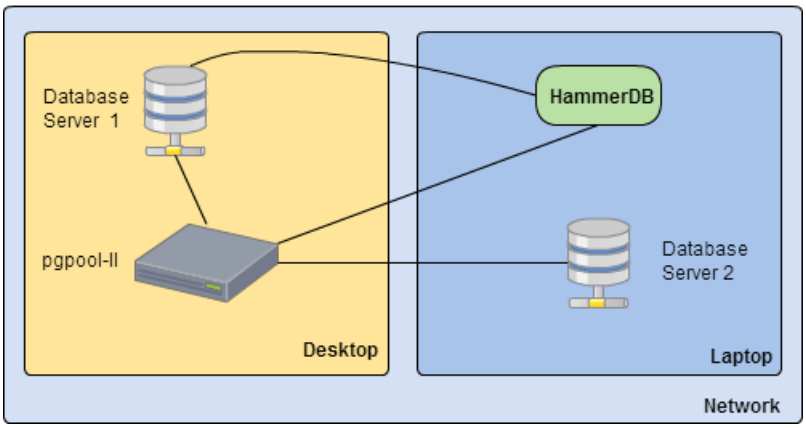

Figura 3.8: Ambiente de Testes 3 - máquinas virtuais de bases de dados isoladas

---

o ganho do modo "load balance", porque a sessão é a mesma durante a execução do teste. Para validar o impacto deste modo de funcionamento foi necessário alterar o script de teste original do HammerDB (Secção C.1), de forma a incluir as alterações efectuadas no código da Secção C.2 do Apêndice C. Essas alterações visaram a abertura e fecho de ligação por cada operação invocada (a verde na Secção C.2), e uma reorganização dos pesos de cada tipo de operação do teste de referência (a azul e vermelho na Secção C.2, identificando as trocas de prioridades entre operações da mesma cor), dando maior prioridade a operações de leitura, e menor prioridade a operações de escrita, de acordo com a Tabela 3.2:

3.4 Síntese

Para os testes que se pretende realizar foram necessárias três arquitecturas diferentes do sistema, variando a localização de cada máquina virtual conforme o objectivo de cada teste.

Foram identificados quatro modos de funcionamento, baseado nas configurações activadas

Operação	Valor
Nova Compra	4%
Pagamento	4%
Entrega	4%
Estado da Compra	45%
Nível de Stock	43%

Tabela 3.2: Distribuição de Carga focada em Operações de Leitura

---

no Pgpool-II: "directo", "pgpool", "pgpool replicate_select"e "load balance".

Cada um dos cinco grupos de testes definidos identifica testes de performance sobre o Pgpool-II, a rede, a memória da instância, o CPU e o comportamento com balanceamento de carga.

Capítulo 4

Resultados

Neste capítulo são apresentados os resultados obtidos pelos testes realizados seguindo o plano de testes definido na Secção 3.3.

4.1 Performance do Pgpool-II

A partir da máquina virtual com Pgpool-II, no "Laptop", foi executado o primeiro grupo de testes recorrendo ao HammerDB. Para cada execução foram recolhidos os resultados obtidos para as métricas TPM e NOPM.

Users	**Modo de Funcionamento**	**TPM**	**NOPM**
1	directo	6980	3002
1	pgpool	5734	2496
1	pgpool replicate_select	2090	903
2	directo	10029	4351
2	pgpool	7771	3364
2	pgpool replicate_select	3009	1323
4	directo	12902	5624
4	pgpool	8208	3516
4	pgpool replicate_select	4203	1842
8	directo	12071	5236
8	pgpool	8151	3549
8	pgpool replicate_select	3709	1625
10	directo	11550	4986
10	pgpool	7849	3390
10	pgpool replicate_select	3222	1392
20	directo	12386	5352
20	pgpool	8733	3777
20	pgpool replicate_select	4061	1748

Tabela 4.1: Resultados com Pgpool-II

Analisando a Tabela 4.1 verificamos que o aumento do número de utilizadores concorrentes produz uma melhoria gradual dos resultados, até ao ponto em que é atingido o limite do sistema. Este limite é atingido com 8 utilizadores nos modos "directo"e "pgpool"e com 4 utilizadores no modo "pgpool replicate_select". A partir deste número os resultados obtidos tendem a estagnar (assumindo alguma variação fruto da entropia gerada pelos diferentes componentes do sistema). Este comportamento é evidenciado para qualquer modo de funcionamento.

Tal como explicado na Secção 3.3, no modo "pgpool"nada é replicado para o nó 2, pelo facto do HammerDB trabalhar apenas com recurso a funções. Desta forma, o trabalho será sempre feito no "Database Server 1", fazendo com que os modos de funcionamento "directo"e "pgpool"tenham o mesmo trabalho, para o mesmo destino, sem replicar informação.

Verificamos que existe sempre uma perda de performance do modo "pgpool"face ao modo "directo". Apesar de na prática, ambos executarem o mesmo teste na mesma instância, a passagem das operações pela máquina virtual implica uma perda de performance, que vai aumentando consoante o número de utilizadores cresce, pelo facto de a perda de performance ter um efeito multiplicativo pelo número de operações executadas.

Relativamente ao modo "pgpool replicate_select", este tem um desempenho bastante pior que os restantes modos de funcionamento. Durante os testes não houve evidências de problemas de performance relativos a CPU, memória ou I/O que justifiquem valores tão baixos, pelo que a justificação está no próprio funcionamento do Pgpool-II. Por cada operação de modificação de dados replicada o Pgpool-II fica à espera de uma resposta de cada nó. Esta resposta serve para que possam ser identificadas discrepâncias entre diferentes nós, indicando essa falta de consistência aos administradores de bases de dados. Este mecanismo tem um custo elevado para este tipo de operações, que neste caso perfazem 92% das operações de cada teste (Tabela 2.1).

4.2 Impacto da Rede na Performance do Sistema

Para validar o impacto da qualidade da rede nos resultados foi alterada a arquitectura do sistema para que a máquina virtual do Pgpool-II esteja no mesmo servidor que as máquinas virtuais de bases de dados (Figura 3.7). Após a alteração, o grupo de testes anterior foi repetido.

Fazendo uma análise comparativa da Tabela 4.2 face à Tabela 4.1 verificamos que a diferença entre os resultados para o modo de funcionamento "directo"é residual, sendo por vezes melhor nos testes com passagem pela rede para 1,2 e 20 utilizadores, e melhor nos testes sem passagem pela rede para 4,8 e 10 utilizadores (Tabela 4.3).
Continuamos a verificar uma perda de performance da passagem pelo Pgpool-II no modo "pgpool"face ao modo "directo", mas esta é muito menos significativa nesta repetição dos testes. O mesmo acontece para o modo "pgpool replicate_select", com a excepção do teste com 1 utilizador.

Users	Modo de Funcionamento	TPM	NOPM
1	directo	4943	2152
1	pgpool	4809	2077
1	pgpool replicate_select	3158	1354
2	directo	9703	4179
2	pgpool	8071	3541
2	pgpool replicate_select	4341	1887
4	directo	13840	5997
4	pgpool	12128	5284
4	pgpool replicate_select	5970	2569
8	directo	15221	6642
8	pgpool	13502	5851
8	pgpool replicate_select	6533	2826
10	directo	12296	5351
10	pgpool	11542	5001
10	pgpool replicate_select	6523	2818
20	directo	10965	4775
20	pgpool	11338	4911
20	pgpool replicate_select	6155	2679

Tabela 4.2: Resultados com todas as máquinas virtuais no mesmo servidor

---

Por estes resultados conseguimos validar que a qualidade da rede tem impacto nos resultados, e o impacto é maior nos modos de funcionamento "pgpool"e "pgpool replicate_select"pelo facto de estes exigirem a passagem de todas as operações pelo Pgpool-II.

Arquitectura	Users	Modo de Funcionamento	TPM	NOPM
1	1	directo	6980	3002
2	1	directo	4943	2152
1	1	pgpool	5734	2496
2	1	pgpool	4809	2077
1	1	pgpool replicate_select	2090	903
2	1	pgpool replicate_select	3158	1354
1	2	directo	10029	4351
2	2	directo	9703	4179
1	2	pgpool	7771	3364
2	2	pgpool	8071	3541
1	2	pgpool replicate_select	3009	1323
2	2	pgpool replicate_select	4341	1887
1	4	directo	12902	5624
2	4	directo	13840	5997
1	4	pgpool	8208	3516
2	4	pgpool	12128	5284
1	4	pgpool replicate_select	4203	1842
2	4	pgpool replicate_select	5970	2569
1	8	directo	12071	5236
2	8	directo	15221	6642
1	8	pgpool	8151	3549
2	8	pgpool	13502	5851
1	8	pgpool replicate_select	3709	1625
2	8	pgpool replicate_select	6533	2826
1	10	directo	11550	4986
2	10	directo	12296	5351
1	10	pgpool	7849	3390
2	10	pgpool	11542	5001
1	10	pgpool replicate_select	3222	1392
2	10	pgpool replicate_select	6523	2818
1	20	directo	12386	5352
2	20	directo	10965	4775
1	20	pgpool	8733	3777
2	20	pgpool	11338	4911
1	20	pgpool replicate_select	4061	1748
2	20	pgpool replicate_select	6155	2679

Tabela 4.3: Análise comparativa entre Testes 1 e 2

4.3 Impacto da Memória na Performance do Sistema

Após a análise dos resultados anteriores para o modo de funcionamento "directo"foi identificado que os melhores resultados foram obtidos com 8 utilizadores concorrentes. Para este número de utilizadores foram executados diversos testes, alterando a memória disponível da instância de base de dados, de forma a validar o impacto nos resultados dos testes de referência. Também são recolhidos dados da utilização média do CPU do "Desktop", e do volume de escritas e leituras por segundo em disco, para identificar o impacto causado nestes componentes do sistema.

Memória	**TPM**	**NOPM**	**CPU**	**Escritas (KB/s)**	**Leituras (KB/s)**
1MB	10042	4414	50%	16670.72	233.16
2MB	10579	4622	48%	18155.52	290.24
4MB	11385	4969	49%	18524.16	320.53
8MB	11750	5106	48%	18944.01	342.89
16MB	11598	5028	47%	21022.72	483.55
32MB	12071	5236	41%	23203.84	531.26
512MB	16132	7025	41%	24422.56	672.84

Tabela 4.4: Resultados para diferentes volumes de memória

Analisando a Tabela 4.4 é evidente que o valor de memória configurado tem um impacto inversamente proporcional relativamente à utilização média do CPU. Este comportamento deve-se ao efeito de paginação da memória. Quando a memória disponível não chega para as operações, é utilizado parte do disco como substituto. Ao nível do sistema operativo este espaço em disco é chamado de "swap space"em sistemas Unix, ou "Page File"em sistemas Windows.

Apesar de haver 2GB de memória RAM disponível na máquina virtual, ao limitarmos a mesma na instância vamos obrigar a que esta tenha um comportamento semelhante, forçando mais operações ao acesso a disco. Este comportamento causa uma sobrecarga ao CPU, que passa a gastar mais tempo neste tipo de invocações a disco, tornando cada operação SQL mais demorada. Como os testes têm tempo limitado, esta demora nas operações causa um decréscimo no número total de operações concluídas.
Para o mesmo número de utilizadores, quanto menor a quantidade de memória disponível na instância, maior a utilização de CPU, e pior o resultado obtido nas métricas TPM e NOPM, fruto do desperdício de invocações a disco.

Apesar deste comportamento indicar que teremos mais acesso a disco para menos memória, os resultados obtidos relativamente a KB de informação escritos por segundo não evidenciam o mesmo. Isto acontece porque as operações de escrita de cada teste são de carga elevada e têm um peso maior que o efeito de paginação, pelo que o mesmo fica mascarado pela performance do próprio teste. Os valores de leituras em KB por segundo são desprezáveis, pelo facto de o teste de referência se focar em operações de escrita.

4.4 Impacto do CPU na Performance do Pgpool-II

Após a alteração da arquitectura do sistema segundo a Figura 3.8 foram executados testes no modo "pgpool replicate_select".

Users	**Modo de Funcionamento**	**TPM**	**NOPM**
1	pgpool replicate_select	1541	658
2	pgpool replicate_select	2632	1142
4	pgpool replicate_select	3974	1614
8	pgpool replicate_select	3327	1453
10	pgpool replicate_select	2943	1225
20	pgpool replicate_select	3490	1536

Tabela 4.5: Resultados com instâncias isoladas em diferentes máquinas

Apesar de isolarmos cada máquina virtual de base de dados em diferentes servidores, analisando a Tabela 4.5 verifica-se que os resultados obtidos são inferiores aos da Tabela 4.1. Este comportamento deve-se ao facto de o processador do "Laptop"ser inferior ao do "Desktop". Como o modo "pgpool replicate_select"produz a mesma carga em ambas as instâncias, os resultados ficam limitados pela instância com menores capacidades.

4.5 Performance com Balanceamento de Carga

Tal como referido anteriormente, como o Pgpool-II faz balanceamento ao nível da sessão não é possível quantificar os ganhos relativos a balanceamento com os resultados obtidos pelo HammerDB. As medidas tomadas ao nível do script de testes implicam que os resultados para a métrica TPM apenas possam ser quantificados de forma manual, somando o número de "commits"e "rollbacks"de cada nó, no fim de cada teste.

Users	**Modo de Funcionamento**	**TPM**
1	directo	3741
1	load balance	3436
2	directo	8452
2	load balance	8183
4	directo	11381
4	load balance	10874
8	directo	13457
8	load balance	12213
10	directo	11566
10	load balance	10763
20	directo	9894
20	load balance	9496

Tabela 4.6: Resultados com balanceamento de carga para TPC-C modificado

Verificamos na Tabela 4.6 que os resultados obtidos com balanceamento de carga são ligeiramente inferiores ao modo "directo". No entanto, apesar de inferiores, estes resultados são muito positivos. Obter resultados semelhantes ao modo "directo"é indicação de que é conseguida a mesma qualidade de serviço, mesmo com o peso do funcionamento do próprio Pgpool-II envolvido.

Como naturalmente ocorrem falhas em sistemas de bases de dados, pelo facto de a qualidade de serviço se manter agora com duas instâncias, este modo de funcionamento ganha vantagem a longo prazo, já que em caso de falha o sistema continua a dar resposta.

4.6 Discussão dos Resultados

Os resultados obtidos evidenciam como a construção de um sistema de bases de dados pode ficar limitada por vários factores envolventes. A arquitectura do sistema deve ser orientada à carga prevista para o mesmo, de forma a que após alguns testes os diversos componentes (CPU, memória, qualidade dos discos, qualidade da rede e placas de rede) possam ser afinados. No caso deste sistema de teste, a rede provocou alguma degradação do serviço, mas o CPU é a limitação principal para que não se consiga obter melhores resultados a partir de 8 utilizadores concorrentes.

Nos resultados obtidos na Secção 4.1 é notório no modo "pgpool replicate_select"o peso do funcionamento da replicação. Comparativamente, o modo "load balance"não desperdiça

performance do sistema em operações redundantes, e o ganho de performance face ao modo "pgpool replicate_select"é suficiente para praticamente anular os custos do funcionamento da replicação, obtendo uma performance semelhante ao modo "directo".

Capítulo 5

Conclusão e Trabalho Futuro

Este trabalho procurou identificar os impactos positivos e negativos da aplicação de replicação de dados a um sistema com um nó, de forma a obter Escalabilidade e Alta Disponibilidade. O Pgpool-II foi o software mediador escolhido pelas suas diversas capacidades.

Relativamente à Escalabilidade, o Pgpool-II é uma opção viável para que um sistema consiga adaptar-se consoante as necessidades, podendo com alguma configuração adicionar ou remover instâncias à solução. Tal como identificado na Secção 2.2, com alguns cuidados relativamente às funções que devem pertencer à black_function_list de balanceamento de carga, o sincronismo entre as várias instâncias consegue ser atingido.

Relativamente à Alta Disponibilidade, podemos ter em funcionamento diversas instâncias, sincronizadas. Em caso de falha de uma delas, o serviço continua normalmente, e o Pgpool-II identifica a instância indisponível para que o administrador responsável possa recuperar a mesma. Verificou-se este comportamento pelos testes realizados com HammerDB direcionados ao Pgpool-II. Sem "replicate_select"activo os testes apenas executam na primeira instância configurada, mas com "replicate_select"activo os testes também executam na segunda instância. Para o HammerDB, que neste caso é o cliente do sistema, não é visível a organização do sistema, pois este apenas está direcionado ao endereço e porto do Pgpool-II. Desde que haja uma instância disponível o Pgpool-II consegue dar resposta ao cliente.

Os resultados obtidos permitem concluir que há grandes vantagens a longo prazo em utilizar o Pgpool-II como solução de replicação e balanceamento de carga, obtendo a desejada Alta Disponibilidade como subproduto da Escalabilidade alcançada. No entanto, também ficou provado que em sistemas de bases de dados é necessário uma configuração cuidada dos recursos, para que o serviço não seja posto em causa.

Apesar de cumprir os requisitos, esta solução de replicação ao nível do statement SQL, exclusiva do Pgpool-II, tem alguns pontos a melhorar. Ao inserir o Pgpool-II num sistema, este torna-se o ponto único de falha de todo o sistema, pelo que é fundamental ter também redundância do próprio Pgpool-II, para o sistema ser realmente tolerante a falhas.

Um possível problema é o balanceamento de carga implementado pelo Pgpool-II. Tal como referido anteriormente, o algoritmo de balanceamneto trabalha ao nível da sessão, e não

ao nível da operação. Este comportamento poderá resultar numa má distribuição da carga efectiva, se existirem sessões com um nível de carga muito superior a outras.

Também existe um potencial problema relativamente à recuperação de uma instância. Uma instância que fique indisponível terá de voltar ao mesmo estado de consistência que as restantes para que possa voltar a ser utilizada. Como não existe controlo sobre a informação que entra nas bases de dados, a única solução para recuperar uma instância é restaurar as bases de dados com base num backup de uma das restantes instâncias. No entanto, se o sistema continuar a funcionar, nunca se conseguirá fazer um backup que represente um estado actual, pelo que existe a necessidade de parar o serviço para que não entrem nas bases de dados mais modificações. Assim sendo, a perda de uma instância implica a paragem de todo o serviço pelo tempo necessário recuperar a mesma.
Para contornar este problema, o Pgpool-II tem a funcionalidade de Recuperação Online, que minimiza o tempo de indisponibilidade do serviço durante a recuperação de uma instância.

Relativamente a trabalho futuro, devido à fraca infraestrutura disponível, fica por analisar o comportamento e os resultados obtidos com balanceamento de carga por cada novo nó adicionado. Apesar de bem documentada, a funcionalidade de Recuperação Online não foi validada, pelo que seria interessante verificar a sua eficiência e testar o tempo de recuperação de uma instância, dependendo do seu volume. Também seria interessante comparar esta solução a outras existentes, nomeadamente o uso de "Streaming Replication"numa arquitectura Master/Slave, que minimiza o tempo de indisponibilidade apesar de limitar a sua aplicação a apenas 2 nós, 1 activo e o outro passivo, disponível apenas para leituras.

Bibliografia

[1] Replication, Clustering, and Connection Pooling. . https://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling. Acedido a 18-Novembro-2014.

[2] HammerDB. Introduction to Transactional (TPC-C) Testing for all Databases. http://hammerora.sourceforge.net/hammerdb_transactionintro.pdf. Acedido a 18-Novembro-2014.

[3] Era da Informação. http://en.wikipedia.org/wiki/Information_Age#Digital_Age. Acedido a 18-Novembro-2014.

[4] Escalabilidade. http://en.wikipedia.org/wiki/Scalability. Acedido a 18-Novembro-2014.

[5] Alta Disponibilidade. http://en.wikipedia.org/wiki/High_availability. Acedido a 18-Novembro-2014.

[6] Margaret Rouse. I/O contention (input/output contention). http://searchstorage.techtarget.com/definition/I-O-contention-input-output-contention. Acedido a 18-Novembro-2014.

[7] Balanceamento de Carga. http://en.wikipedia.org/wiki/Load_balancing_%28computing%29. Acedido a 18-Novembro-2014.

[8] Emmanuel Cecchet, George Candea, and Anastasia Ailamaki. Middleware-based database replication: the gaps between theory and practice. *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 739–752, 2008.

[9] Anja Bog, Hasso Plattner, and Alexander Zeier. A mixed transaction processing and operational reporting benchmark. *Information Systems Frontiers*, 13(3):321–335, October 2010.

[10] Vaidė Zuikevičiūtė and Fernando Pedone. Conflict-aware load-balancing techniques for database replication. *Proceedings of the 2008 ACM symposium on Applied computing - SAC '08*, pages 2169–2173, 2008.

[11] R Garcia, Rodrigo Rodrigues, and N Preguiça. Efficient middleware for byzantine fault tolerant database replication. *Proceedings of the sixth conference on Computer systems*, pages 107–122, 2011.

[12] Takeshi Mishima and Hiroshi Nakamura. Pangea: An eager database replication middleware guaranteeing snapshot isolation without modification of database servers. *Proceedings of the VLDB Endowment*, 2(1):1066–1077, 2009.

[13] Sherif Sakr. Cloud-hosted databases: technologies, challenges and opportunities. *Cluster Computing*, 17(2):487–502, July 2013.

[14] E Sarhan, A Ghalwash, and M Khafagy. Queue weighting load-balancing technique for database replication in dynamic content web sites. *Proceedings of the 9th WSEAS international conference on Applied computer science*, pages 50–55, 2009.

[15] Pgpool-II. https://wiki.postgresql.org/wiki/Pgpool-II#General_Information. Acedido a 18-Novembro-2014.

[16] Slony. https://wiki.postgresql.org/wiki/Slony. Acedido a 18-Novembro-2014.

[17] Bucardo/FAQ. http://bucardo.org/wiki/Bucardo/FAQ. Acedido a 18-Novembro-2014.

[18] pgpool Global Development Group. Pgpool-II User Manual). http://www.pgpool.net/docs/latest/pgpool-en.html. Acedido a 18-Novembro-2014.

[19] PostgreSQL Global Development Group. PostgreSQL 9.3.5 Documentation, 24.3. Continuous Archiving and Point-in-Time Recovery (PITR). http://www.postgresql.org/docs/9.3/static/continuous-archiving.html. Acedido a 18-Novembro-2014.

[20] TPC (Transaction Processing Performance Council). Homepage. http://www.tpc.org/. Acedido a 18-Novembro-2014.

[21] TPC (Transaction Processing Performance Council). Who We Are. http://www.tpc.org/information/who/whoweare.asp. Acedido a 18-Novembro-2014.

[22] Frequently Asked Questions (FAQ)What Is the TPC? http://www.tpc.org/information/about/faq-generic.asp. Acedido a 18-Novembro-2014.

[23] TPC (Transaction Processing Performance Council). TPC Benchmarks. http://www.tpc.org/information/benchmarks.asp. Acedido a 18-Novembro-2014.

[24] Transaction Processing Performance Council. TPC BENCHMARK^{TM} C - Standard Specification. (February):1–130, 2010.

[25] HammerDB. Home Page. http://hammerora.sourceforge.net/. Acedido a 18-Novembro-2014.

[26] Pgpool-II Download). http://www.pgpool.net/mediawiki/index.php/Downloads. Acedido a 18-Novembro-2014.

[27] PostgreSQL Global Development Group. PostgreSQL 9.3.5 Documentation, Chapter 18.3. Connections and Authentication. http://www.postgresql.org/docs/9.3/static/runtime-config-connection.html. Acedido a 18-Novembro-2014.

[28] PostgreSQL Global Development Group. PostgreSQL 9.3.5 Documentation, Chapter 18.4. Resource Consumption. http://www.postgresql.org/docs/9.3/static/runtime-config-resource.html. Acedido a 18-Novembro-2014.

[29] Pgpool Global Development Group. Installing pgpool Administration Tool. http://pgpool.projects.pgfoundry.org/pgpoolAdmin/doc/en/install.html. Acedido a 18-Novembro-2014.

Apêndice A

Script de Configuração do PostgreSQL

```
# -----------------------------
# PostgreSQL configuration file
# -----------------------------
#
# This file consists of lines of the form:
#
#   name = value
#
# (The "=" is optional.)  Whitespace may be used.  Comments are introduced with
# "#" anywhere on a line.  The complete list of parameter names and allowed
# values can be found in the PostgreSQL documentation.
#
# The commented-out settings shown in this file represent the default values.
# Re-commenting a setting is NOT sufficient to revert it to the default value;
# you need to reload the server.
#
# This file is read on server startup and when the server receives a SIGHUP
# signal.  If you edit the file on a running system, you have to SIGHUP the
# server for the changes to take effect, or use "pg_ctl reload".  Some
# parameters, which are marked below, require a server shutdown and restart to
# take effect.
#
# Any parameter can also be given as a command-line option to the server, e.g.,
# "postgres -c log_connections=on".  Some parameters can be changed at run time
# with the "SET" SQL command.
#
# Memory units:  kB = kilobytes        Time units:  ms  = milliseconds
#                MB = megabytes                     s   = seconds
#                GB = gigabytes                     min = minutes
#                                                   h   = hours
#                                                   d   = days
```

```
#------------------------------------------------------------------------------
# FILE LOCATIONS
#------------------------------------------------------------------------------

# The default values of these variables are driven from the -D command-line
# option or PGDATA environment variable, represented here as ConfigDir.

#data_directory = 'ConfigDir'# use data in another directory
# (change requires restart)
#hba_file = 'ConfigDir/pg_hba.conf'# host-based authentication file
# (change requires restart)
#ident_file = 'ConfigDir/pg_ident.conf'# ident configuration file
# (change requires restart)

# If external_pid_file is not explicitly set, no extra PID file is written.
#external_pid_file = ''# write an extra PID file
# (change requires restart)


#------------------------------------------------------------------------------
# CONNECTIONS AND AUTHENTICATION
#------------------------------------------------------------------------------

# - Connection Settings -


listen_addresses = '*'
                    # what IP address(es) to listen on;
# comma-separated list of addresses;
# defaults to 'localhost'; use '*' for all
# (change requires restart)
port = 5432 # (change requires restart)
max_connections = 1000 # (change requires restart)


# Note:  Increasing max_connections costs ~400 bytes of shared memory per
# connection slot, plus lock space (see max_locks_per_transaction).
#superuser_reserved_connections = 3 # (change requires restart)
#unix_socket_directories = ''# comma-separated list of directories
# (change requires restart)
#unix_socket_group = ''# (change requires restart)
#unix_socket_permissions = 0777 # begin with 0 to use octal notation
# (change requires restart)
#bonjour = off # advertise server via Bonjour
# (change requires restart)
#bonjour_name = ''# defaults to the computer name
```

```
# (change requires restart)

# - Security and Authentication -

#authentication_timeout = 1min # 1s-600s
#ssl = off # (change requires restart)
#ssl_ciphers = 'DEFAULT:!LOW:!EXP:!MD5:@STRENGTH'# allowed SSL ciphers
# (change requires restart)
#ssl_renegotiation_limit = 512MB # amount of data between renegotiations
#ssl_cert_file = 'server.crt'# (change requires restart)
#ssl_key_file = 'server.key'# (change requires restart)
#ssl_ca_file = ''# (change requires restart)
#ssl_crl_file = ''# (change requires restart)
#password_encryption = on
#db_user_namespace = off

# Kerberos and GSSAPI
#krb_server_keyfile = ''
#krb_srvname = 'postgres'# (Kerberos only)
#krb_caseins_users = off

# - TCP Keepalives -
# see "man 7 tcp" for details

#tcp_keepalives_idle = 0 # TCP_KEEPIDLE, in seconds;
# 0 selects the system default
#tcp_keepalives_interval = 0 # TCP_KEEPINTVL, in seconds;
# 0 selects the system default
#tcp_keepalives_count = 0 # TCP_KEEPCNT;
# 0 selects the system default


#------------------------------------------------------------------------------
# RESOURCE USAGE (except WAL)
#------------------------------------------------------------------------------

# - Memory -


shared_buffers = 32MB # min 128kB


# (change requires restart)
#temp_buffers = 8MB # min 800kB
#max_prepared_transactions = 0 # zero disables the feature
# (change requires restart)
# Note:  Increasing max_prepared_transactions costs ~600 bytes of shared memory
# per transaction slot, plus lock space (see max_locks_per_transaction).
```

```
# It is not advisable to set max_prepared_transactions nonzero unless you
# actively intend to use prepared transactions.
#work_mem = 1MB # min 64kB
#maintenance_work_mem = 16MB # min 1MB
#max_stack_depth = 2MB # min 100kB

# - Disk -

#temp_file_limit = -1 # limits per-session temp file space
# in kB, or -1 for no limit

# - Kernel Resource Usage -

#max_files_per_process = 1000 # min 25
# (change requires restart)
#shared_preload_libraries = ''# (change requires restart)

# - Cost-Based Vacuum Delay -

#vacuum_cost_delay = 0 # 0-100 milliseconds
#vacuum_cost_page_hit = 1 # 0-10000 credits
#vacuum_cost_page_miss = 10 # 0-10000 credits
#vacuum_cost_page_dirty = 20 # 0-10000 credits
#vacuum_cost_limit = 200 # 1-10000 credits

# - Background Writer -

#bgwriter_delay = 200ms # 10-10000ms between rounds
#bgwriter_lru_maxpages = 100 # 0-1000 max buffers written/round
#bgwriter_lru_multiplier = 2.0 # 0-10.0 multipler on buffers scanned/round

# - Asynchronous Behavior -

#effective_io_concurrency = 1 # 1-1000; 0 disables prefetching


#------------------------------------------------------------------------------
# WRITE AHEAD LOG
#------------------------------------------------------------------------------

# - Settings -

#wal_level = minimal # minimal, archive, or hot_standby
# (change requires restart)
#fsync = on # turns forced synchronization on or off
#synchronous_commit = on # synchronization level;
# off, local, remote_write, or on
```

```
#wal_sync_method = fsync # the default is the first option
# supported by the operating system:
#   open_datasync
#   fdatasync (default on Linux)
#   fsync
#   fsync_writethrough
#   open_sync
#full_page_writes = on # recover from partial page writes
#wal_buffers = -1 # min 32kB, -1 sets based on shared_buffers
# (change requires restart)
#wal_writer_delay = 200ms # 1-10000 milliseconds

#commit_delay = 0 # range 0-100000, in microseconds
#commit_siblings = 5 # range 1-1000

# - Checkpoints -

#checkpoint_segments = 3 # in logfile segments, min 1, 16MB each
#checkpoint_timeout = 5min # range 30s-1h
#checkpoint_completion_target = 0.5 # checkpoint target duration, 0.0 - 1.0
#checkpoint_warning = 30s # 0 disables

# - Archiving -

#archive_mode = off # allows archiving to be done
# (change requires restart)
#archive_command = ''# command to use to archive a logfile segment
# placeholders: %p = path of file to archive
#               %f = file name only
# e.g. 'test ! -f /mnt/server/archivedir/%f && cp %p /mnt/server/archivedir/%f'
#archive_timeout = 0 # force a logfile segment switch after this
# number of seconds; 0 disables


#------------------------------------------------------------------------------
# REPLICATION
#------------------------------------------------------------------------------

# - Sending Server(s) -

# Set these on the master and on any standby that will send replication data.

#max_wal_senders = 0 # max number of walsender processes
# (change requires restart)
#wal_keep_segments = 0 # in logfile segments, 16MB each; 0 disables
#wal_sender_timeout = 60s # in milliseconds; 0 disables
```

```
# - Master Server -

# These settings are ignored on a standby server.

#synchronous_standby_names = ''# standby servers that provide sync rep
# comma-separated list of application_name
# from standby(s); '*' = all
#vacuum_defer_cleanup_age = 0 # number of xacts by which cleanup is delayed

# - Standby Servers -

# These settings are ignored on a master server.

#hot_standby = off # "on" allows queries during recovery
# (change requires restart)
#max_standby_archive_delay = 30s # max delay before canceling queries
# when reading WAL from archive;
# -1 allows indefinite delay
#max_standby_streaming_delay = 30s # max delay before canceling queries
# when reading streaming WAL;
# -1 allows indefinite delay
#wal_receiver_status_interval = 10s # send replies at least this often
# 0 disables
#hot_standby_feedback = off # send info from standby to prevent
# query conflicts
#wal_receiver_timeout = 60s # time that receiver waits for
# communication from master
# in milliseconds; 0 disables


#------------------------------------------------------------------------------
# QUERY TUNING
#------------------------------------------------------------------------------

# - Planner Method Configuration -

#enable_bitmapscan = on
#enable_hashagg = on
#enable_hashjoin = on
#enable_indexscan = on
#enable_indexonlyscan = on
#enable_material = on
#enable_mergejoin = on
#enable_nestloop = on
#enable_seqscan = on
#enable_sort = on
#enable_tidscan = on
```

```
# - Planner Cost Constants -

#seq_page_cost = 1.0 # measured on an arbitrary scale
#random_page_cost = 4.0 # same scale as above
#cpu_tuple_cost = 0.01 # same scale as above
#cpu_index_tuple_cost = 0.005 # same scale as above
#cpu_operator_cost = 0.0025 # same scale as above
#effective_cache_size = 128MB

# - Genetic Query Optimizer -

#geqo = on
#geqo_threshold = 12
#geqo_effort = 5 # range 1-10
#geqo_pool_size = 0 # selects default based on effort
#geqo_generations = 0 # selects default based on effort
#geqo_selection_bias = 2.0 # range 1.5-2.0
#geqo_seed = 0.0 # range 0.0-1.0

# - Other Planner Options -

#default_statistics_target = 100 # range 1-10000
#constraint_exclusion = partition # on, off, or partition
#cursor_tuple_fraction = 0.1 # range 0.0-1.0
#from_collapse_limit = 8
#join_collapse_limit = 8 # 1 disables collapsing of explicit
# JOIN clauses


#------------------------------------------------------------------------------
# ERROR REPORTING AND LOGGING
#------------------------------------------------------------------------------

# - Where to Log -

log_destination = 'stderr'# Valid values are combinations of
# stderr, csvlog, syslog, and eventlog,
# depending on platform.  csvlog
# requires logging_collector to be on.

# This is used when logging to stderr:
logging_collector = on # Enable capturing of stderr and csvlog
# into log files. Required to be on for
# csvlogs.
# (change requires restart)
```

```
# These are only used if logging_collector is on:
#log_directory = 'pg_log'# directory where log files are written,
# can be absolute or relative to PGDATA
#log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'# log file name pattern,
# can include strftime() escapes
#log_file_mode = 0600 # creation mode for log files,
# begin with 0 to use octal notation
#log_truncate_on_rotation = off # If on, an existing log file with the
# same name as the new log file will be
# truncated rather than appended to.
# But such truncation only occurs on
# time-driven rotation, not on restarts
# or size-driven rotation.  Default is
# off, meaning append to existing files
# in all cases.
#log_rotation_age = 1d # Automatic rotation of logfiles will
# happen after that time.  0 disables.
#log_rotation_size = 10MB # Automatic rotation of logfiles will
# happen after that much log output.
# 0 disables.

# These are relevant when logging to syslog:
#syslog_facility = 'LOCAL0'
#syslog_ident = 'postgres'

# This is only relevant when logging to eventlog (win32):
#event_source = 'PostgreSQL'

# - When to Log -

#client_min_messages = notice # values in order of decreasing detail:
#   debug5
#   debug4
#   debug3
#   debug2
#   debug1
#   log
#   notice
#   warning
#   error

#log_min_messages = warning # values in order of decreasing detail:
#   debug5
#   debug4
#   debug3
#   debug2
#   debug1
```

```
#   info
#   notice
#   warning
#   error
#   log
#   fatal
#   panic

#log_min_error_statement = error # values in order of decreasing detail:
#   debug5
#   debug4
#   debug3
#   debug2
#   debug1
#   info
#   notice
#   warning
#   error
#   log
#   fatal
#   panic (effectively off)

#log_min_duration_statement = -1 # -1 is disabled, 0 logs all statements
# and their durations, > 0 logs only
# statements running at least this number
# of milliseconds


# - What to Log -

#debug_print_parse = off
#debug_print_rewritten = off
#debug_print_plan = off
#debug_pretty_print = on
#log_checkpoints = off
#log_connections = off
#log_disconnections = off
#log_duration = off
#log_error_verbosity = default # terse, default, or verbose messages
#log_hostname = off
log_line_prefix = '%t '# special values:
#   %a = application name
#   %u = user name
#   %d = database name
#   %r = remote host and port
#   %h = remote host
#   %p = process ID
```

```
#   %t = timestamp without milliseconds
#   %m = timestamp with milliseconds
#   %i = command tag
#   %e = SQL state
#   %c = session ID
#   %l = session line number
#   %s = session start timestamp
#   %v = virtual transaction ID
#   %x = transaction ID (0 if none)
#   %q = stop here in non-session
#        processes
#   %% = '%'
# e.g. '<%u%%%d> '
#log_lock_waits = off # log lock waits >= deadlock_timeout
#log_statement = 'none'# none, ddl, mod, all
#log_temp_files = -1 # log temporary files equal or larger
# than the specified size in kilobytes;
# -1 disables, 0 logs all temp files
log_timezone = 'Europe/London'


#------------------------------------------------------------------------------
# RUNTIME STATISTICS
#------------------------------------------------------------------------------
# - Query/Index Statistics Collector -

#track_activities = on
#track_counts = on
#track_io_timing = off
#track_functions = none # none, pl, all
#track_activity_query_size = 1024 # (change requires restart)
#update_process_title = on
#stats_temp_directory = 'pg_stat_tmp'


# - Statistics Monitoring -

#log_parser_stats = off
#log_planner_stats = off
#log_executor_stats = off
#log_statement_stats = off


#------------------------------------------------------------------------------
# AUTOVACUUM PARAMETERS
#------------------------------------------------------------------------------
```

```
#autovacuum = on # Enable autovacuum subprocess?  'on'
# requires track_counts to also be on.
#log_autovacuum_min_duration = -1 # -1 disables, 0 logs all actions and
# their durations, > 0 logs only
# actions running at least this number
# of milliseconds.
#autovacuum_max_workers = 3 # max number of autovacuum subprocesses
# (change requires restart)
#autovacuum_naptime = 1min # time between autovacuum runs
#autovacuum_vacuum_threshold = 50 # min number of row updates before
# vacuum
#autovacuum_analyze_threshold = 50 # min number of row updates before
# analyze
#autovacuum_vacuum_scale_factor = 0.2 # fraction of table size before vacuum
#autovacuum_analyze_scale_factor = 0.1 # fraction of table size before analyze
#autovacuum_freeze_max_age = 200000000 # maximum XID age before forced vacuum
# (change requires restart)
#autovacuum_multixact_freeze_max_age = 400000000 # maximum Multixact age
# before forced vacuum
# (change requires restart)
#autovacuum_vacuum_cost_delay = 20ms # default vacuum cost delay for
# autovacuum, in milliseconds;
# -1 means use vacuum_cost_delay
#autovacuum_vacuum_cost_limit = -1 # default vacuum cost limit for
# autovacuum, -1 means use
# vacuum_cost_limit


#------------------------------------------------------------------------------
# CLIENT CONNECTION DEFAULTS
#------------------------------------------------------------------------------

# - Statement Behavior -

#search_path = '"$user",public'# schema names
#default_tablespace = ''# a tablespace name, '' uses the default
#temp_tablespaces = ''# a list of tablespace names, '' uses
# only default tablespace
#check_function_bodies = on
#default_transaction_isolation = 'read committed'
#default_transaction_read_only = off
#default_transaction_deferrable = off
#session_replication_role = 'origin'
#statement_timeout = 0 # in milliseconds, 0 is disabled
#lock_timeout = 0 # in milliseconds, 0 is disabled
#vacuum_freeze_min_age = 50000000
#vacuum_freeze_table_age = 150000000
```

```
#vacuum_multixact_freeze_min_age = 5000000
#vacuum_multixact_freeze_table_age = 150000000
#bytea_output = 'hex'# hex, escape
#xmlbinary = 'base64'
#xmloption = 'content'

# - Locale and Formatting -

datestyle = 'iso, dmy'
#intervalstyle = 'postgres'
timezone = 'Europe/London'
#timezone_abbreviations = 'Default'     # Select the set of available time zone
# abbreviations.  Currently, there are
#   Default
#   Australia
#   India
# You can create your own file in
# share/timezonesets/.
#extra_float_digits = 0 # min -15, max 3
#client_encoding = sql_ascii # actually, defaults to database
# encoding

# These settings are initialized by initdb, but they can be changed.
lc_messages = 'Portuguese_Portugal.1252'# locale for system error message
# strings
lc_monetary = 'Portuguese_Portugal.1252'# locale for monetary formatting
lc_numeric = 'Portuguese_Portugal.1252'# locale for number formatting
lc_time = 'Portuguese_Portugal.1252'# locale for time formatting

# default configuration for text search
default_text_search_config = 'pg_catalog.portuguese'

# - Other Defaults -

#dynamic_library_path = '$libdir'
#local_preload_libraries = ''


#------------------------------------------------------------------------------
# LOCK MANAGEMENT
#------------------------------------------------------------------------------

#deadlock_timeout = 1s
#max_locks_per_transaction = 64 # min 10
# (change requires restart)
# Note:  Each lock table slot uses ~270 bytes of shared memory, and there are
# max_locks_per_transaction * (max_connections + max_prepared_transactions)
```

```
# lock table slots.
#max_pred_locks_per_transaction = 64 # min 10
# (change requires restart)


#------------------------------------------------------------------------------
# VERSION/PLATFORM COMPATIBILITY
#------------------------------------------------------------------------------

# - Previous PostgreSQL Versions -

#array_nulls = on
#backslash_quote = safe_encoding # on, off, or safe_encoding
#default_with_oids = off
#escape_string_warning = on
#lo_compat_privileges = off
#quote_all_identifiers = off
#sql_inheritance = on
#standard_conforming_strings = on
#synchronize_seqscans = on

# - Other Platforms and Clients -

#transform_null_equals = off


#------------------------------------------------------------------------------
# ERROR HANDLING
#------------------------------------------------------------------------------

#exit_on_error = off # terminate session on any error?
#restart_after_crash = on # reinitialize after backend crash?


#------------------------------------------------------------------------------
# CONFIG FILE INCLUDES
#------------------------------------------------------------------------------

# These options allow settings to be loaded from files other than the
# default postgresql.conf.

#include_dir = 'conf.d'# include files ending in '.conf' from
# directory 'conf.d'
#include_if_exists = 'exists.conf'# include file only if it exists
#include = 'special.conf'# include file
```

```
#------------------------------------------------------------------------------
# CUSTOMIZED OPTIONS
#------------------------------------------------------------------------------

# Add settings for extensions here
```

Apêndice B

Scripts de Instalação de Software

B.1 Instalação do Pgpool-II

```
sudo yum install pgpool-II-pg93-3.3.3-1.pgdg.x86_64.rpm
cd /var/run/
sudo mkdir pgpool
sudo chmod 777 pgpool/
```

B.2 Instalação do PgpoolAdmin

```
sudo yum install pgpoolAdmin-3.3.1-1.pgdg.noarch.rpm
cd /usr/bin
sudo chmod 755 pgpool
sudo chmod 755 pcp_*
cd /etc/pgpool-II
pg_md5 <password escolhida>
-> adicionar "hugo:<resultado do md5>" ao ficheiro pcp.conf
cd /var/www/html/pgpoolAdmin
sudo chmod 777 templates_c
cd /var/www/html/pgpoolAdmin/conf
sudo chown apache pgmgt.conf.php
sudo chmod 644 pgmgt.conf.php
cd /etc/pgpool-II/
sudo chown apache pgpool.conf
sudo chmod 644 pgpool.conf
sudo chown apache pcp.conf
sudo chmod 644 pcp.conf
sudo apachectl stop
sudo apachectl start
```

Apêndice C

HammerDB

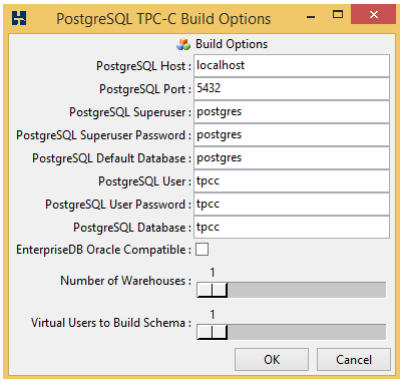

Figura C.1: Criação de Schema de Teste

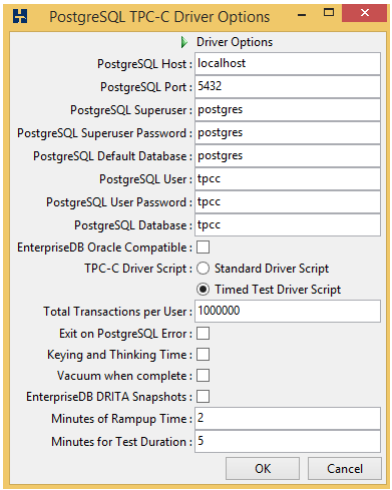

Figura C.2: Definições de Execução de Teste

C.1 Script de Testes HammerDB Original

```
#!/usr/local/bin/tclsh8.6
if [catch {package require Pgtcl} ]
{ error "Failed to load Pgtcl - Postgres Library Error" }
#EDITABLE OPTIONS##################################################
set total_iterations 1000000 ;# Number of transactions before logging off
set RAISEERROR "false" ;# Exit script on PostgreSQL (true or false)
set KEYANDTHINK "false" ;# Time for user thinking and keying (true or false)
set rampup 5;  # Rampup time in minutes before first Transaction Count is taken
set duration 30;  # Duration in minutes before second Transaction Count is taken
set mode "Local" ;# HammerDB operational mode
set VACUUM "false" ;# Perform checkpoint and vacuuum when complete (true or false)
set DRITA_SNAPSHOTS "false";#Take DRITA Snapshots
set ora_compatible "false" ;#Postgres Plus Oracle Compatible Schema
set host "localhost" ;# Address of the server hosting PostgreSQL
set port "9999" ;# Port of the PostgreSQL server
set superuser "postgres" ;# Superuser privilege user
set superuser_password "postgres" ;# Password for Superuser
set default_database "postgres" ;# Default Database for Superuser
set user "tpcc" ;# PostgreSQL user
set password "tpcc" ;# Password for the PostgreSQL user
set db "tpcc" ;# Database containing the TPC Schema
#EDITABLE OPTIONS##################################################
#CHECK THREAD STATUS
proc chk_thread {} {
set chk [package provide Thread]
if {[string length $chk]} {
    return "TRUE"
    } else {
    return "FALSE"
}
    }
if { [ chk_thread ] eq "FALSE" } {
error "PostgreSQL Timed Test Script must be run in Thread Enabled Interpreter"
}

proc ConnectToPostgres { host port user password dbname } {
global tcl_platform
if {[catch {set lda [pg_connect -conninfo [list host = $host port = $port
user = $user password = $password dbname = $dbname ]]}]} {
puts stderr "Error, the database connection to $host could not be established"
set lda "Failed"
 } else {
if {$tcl_platform(platform) == "windows"} {
#Workaround for Bug #95 where first connection fails on Windows
catch {pg_disconnect $lda}
```

```
set lda [pg_connect -conninfo [list host = $host port = $port
user = $user password = $password dbname = $dbname ]]
        }
pg_notice_handler $lda puts
set result [ pg_exec $lda "set CLIENT_MIN_MESSAGES TO 'ERROR'" ]
pg_result $result -clear
        }
return $lda
}

set mythread [thread::id]
set allthreads [split [thread::names]]
set totalvirtualusers [expr [llength $allthreads] - 1]
set myposition [expr $totalvirtualusers - [lsearch -exact $allthreads $mythread]]
if {![catch {set timeout [tsv::get application timeout]}]} {
if { $timeout eq 0 } {
set totalvirtualusers [ expr $totalvirtualusers - 1 ]
set myposition [ expr $myposition - 1 ]
}
}
switch $myposition {
1 {
if { $mode eq "Local" || $mode eq "Master" } {
if { ($DRITA_SNAPSHOTS eq "true") || ($VACUUM eq "true") } {
set lda [ ConnectToPostgres $host $port $superuser $superuser_password
$default_database ]
if { $lda eq "Failed" } {
error "error, the database connection to $host could not be established"
  }
}
set lda1 [ ConnectToPostgres $host $port $user $password $db ]
if { $lda1 eq "Failed" } {
error "error, the database connection to $host could not be established"
  }
set ramptime 0
puts "Beginning rampup time of $rampup minutes"
set rampup [ expr $rampup*60000 ]
while {$ramptime != $rampup} {
if { [ tsv::get application abort ] } { break } else { after 6000 }
set ramptime [ expr $ramptime+6000 ]
if { ![ expr {$ramptime % 60000} ] } {
puts "Rampup [ expr $ramptime / 60000 ] minutes complete ..."
}
}
if { [ tsv::get application abort ] } { break }
if { $DRITA_SNAPSHOTS eq "true" } {
puts "Rampup complete, Taking start DRITA snapshot."
```

```
set result [pg_exec $lda "select * from edbsnap()" ]
if {[pg_result $result -status] ni {"PGRES_TUPLES_OK" "PGRES_COMMAND_OK"}} {
if { $RAISEERROR } {
error "[pg_result $result -error]"
} else {
puts "DRITA Snapshot Error set RAISEERROR for Details"
}
} else {
pg_result $result -clear
pg_select $lda {select edb_id,snap_tm from edb$snap order by edb_id desc limit 1}
snap_arr {
set firstsnap $snap_arr(edb_id)
set first_snaptime $snap_arr(snap_tm)
}
puts "Start Snapshot $firstsnap taken at $first_snaptime"
}
   } else {
puts "Rampup complete, Taking start Transaction Count."
}
pg_select $lda1 "select sum(xact_commit + xact_rollback) from pg_stat_database"
tx_arr {
set start_trans $tx_arr(sum)
}
pg_select $lda1 "select sum(d_next_o_id) from district" o_id_arr {
set start_nopm $o_id_arr(sum)
}
puts "Timing test period of $duration in minutes"
set testtime 0
set durmin $duration
set duration [ expr $duration*60000 ]
while {$testtime != $duration} {
if { [ tsv::get application abort ] } { break } else { after 6000 }
set testtime [ expr $testtime+6000 ]
if { ![ expr {$testtime % 60000} ] } {
puts -nonewline  "[ expr $testtime / 60000 ]  ...,"
}
}
if { [ tsv::get application abort ] } { break }
if { $DRITA_SNAPSHOTS eq "true" } {
puts "Test complete, Taking end DRITA snapshot."
set result [pg_exec $lda "select * from edbsnap()" ]
if {[pg_result $result -status] ni {"PGRES_TUPLES_OK" "PGRES_COMMAND_OK"}} {
if { $RAISEERROR } {
error "[pg_result $result -error]"
} else {
puts "Snapshot Error set RAISEERROR for Details"
}
```

```
} else {
pg_result $result -clear
pg_select $lda {select edb_id,snap_tm from edb$snap order by edb_id desc limit 1}
snap_arr  {
set endsnap $snap_arr(edb_id)
set end_snaptime $snap_arr(snap_tm)
}
puts "End Snapshot $endsnap taken at $end_snaptime"
puts "Test complete: view DRITA report from SNAPID $firstsnap to $endsnap"
}
   } else {
puts "Test complete, Taking end Transaction Count."
}
pg_select $lda1 "select sum(xact_commit + xact_rollback) from pg_stat_database"
tx_arr {
set end_trans $tx_arr(sum)
}
pg_select $lda1 "select sum(d_next_o_id) from district" o_id_arr {
set end_nopm $o_id_arr(sum)
}
set tpm [ expr {($end_trans - $start_trans)/$durmin} ]
set nopm [ expr {($end_nopm - $start_nopm)/$durmin} ]
puts "$totalvirtualusers Virtual Users configured"
puts "TEST RESULT : System achieved $tpm PostgreSQL TPM at $nopm NOPM"
tsv::set application abort 1
if { $VACUUM } {
set RAISEERROR "true"
puts "Checkpoint and Vacuum"
set result [pg_exec $lda "checkpoint" ]
if {[pg_result $result -status] ni {"PGRES_TUPLES_OK" "PGRES_COMMAND_OK"}} {
if { $RAISEERROR } {
error "[pg_result $result -error]"
} else {
puts "Checkpoint Error set RAISEERROR for Details"
}
} else {
pg_result $result -clear
}
set result [pg_exec $lda "vacuum" ]
if {[pg_result $result -status] ni {"PGRES_TUPLES_OK" "PGRES_COMMAND_OK"}} {
if { $RAISEERROR } {
error "[pg_result $result -error]"
} else {
puts "Vacuum Error set RAISEERROR for Details"
}
} else {
puts "Checkpoint and Vacuum Complete"
```

```
pg_result $result -clear
}
}
if { ($DRITA_SNAPSHOTS eq "true") || ($VACUUM eq "true") } {
pg_disconnect $lda
}
pg_disconnect $lda1
} else {
puts "Operating in Slave Mode, No Snapshots taken..."
}
}
default {
#RANDOM NUMBER
proc RandomNumber {m M} {return [expr {int($m+rand()*($M+1-$m))}]}
#NURand function
proc NURand { iConst x y C } {return [ expr {((([RandomNumber 0 $iConst] |
[RandomNumber $x $y]) + $C) % ($y - $x + 1)) + $x }]}
#RANDOM NAME
proc randname { num } {
array set namearr { 0 BAR 1 OUGHT 2 ABLE 3 PRI 4 PRES 5 ESE 6 ANTI 7
CALLY 8 ATION 9 EING }
set name [ concat $namearr([ expr {( $num / 100 ) % 10 }])
$namearr([ expr {( $num / 10 ) % 10 }])$namearr([ expr {( $num / 1 ) % 10 }]) ]
return $name
}
#TIMESTAMP
proc gettimestamp { } {
set tstamp [ clock format [ clock seconds ] -format %Y%m%d%H%M%S ]
return $tstamp
}
#KEYING TIME
proc keytime { keying } {
after [ expr {$keying * 1000} ]
return
}
#THINK TIME
proc thinktime { thinking } {
set thinkingtime [ expr {abs(round(log(rand()) * $thinking))} ]
after [ expr {$thinkingtime * 1000} ]
return
}
#NEW ORDER
proc neword { lda no_w_id w_id_input RAISEERROR ora_compatible } {
#2.4.1.2 select district id randomly from home warehouse where d_w_id = d_id
set no_d_id [ RandomNumber 1 10 ]
#2.4.1.2 Customer id randomly selected where c_d_id = d_id and c_w_id = w_id
set no_c_id [ RandomNumber 1 3000 ]
```

```
#2.4.1.3 Items in the order randomly selected from 5 to 15
set ol_cnt [ RandomNumber 5 15 ]
#2.4.1.6 order entry date O_ENTRY_D generated by SUT
set date [ gettimestamp ]
if { $ora_compatible eq "true" } {
set result [pg_exec $lda "exec neword($no_w_id,$w_id_input,$no_d_id,$no_c_id,
$ol_cnt,0,TO_TIMESTAMP($date,'YYYYMMDDHH24MISS'))" ]
} else {
set result [pg_exec $lda "select neword($no_w_id,$w_id_input,$no_d_id,
$no_c_id,$ol_cnt,0)" ]
}
if {[pg_result $result -status] != "PGRES_TUPLES_OK"} {
if { $RAISEERROR } {
error "[pg_result $result -error]"
} else {
puts "New Order Procedure Error set RAISEERROR for Details"
}
} else {
pg_result $result -clear
}
}
#PAYMENT
proc payment { lda p_w_id w_id_input RAISEERROR ora_compatible } {
#2.5.1.1 The home warehouse id remains the same for each terminal
#2.5.1.1 select district id randomly from home warehouse where d_w_id = d_id
set p_d_id [ RandomNumber 1 10 ]
#2.5.1.2 customer selected 60% of time by name and 40% of time by number
set x [ RandomNumber 1 100 ]
set y [ RandomNumber 1 100 ]
if { $x <= 85 } {
set p_c_d_id $p_d_id
set p_c_w_id $p_w_id
} else {
#use a remote warehouse
set p_c_d_id [ RandomNumber 1 10 ]
set p_c_w_id [ RandomNumber 1 $w_id_input ]
while { ($p_c_w_id == $p_w_id) && ($w_id_input != 1) } {
set p_c_w_id [ RandomNumber 1  $w_id_input ]
}
}
set nrnd [ NURand 255 0 999 123 ]
set name [ randname $nrnd ]
set p_c_id [ RandomNumber 1 3000 ]
if { $y <= 60 } {
#use customer name
#C_LAST is generated
set byname 1
```

```
 } else {
#use customer number
set byname 0
set name {}
 }
#2.5.1.3 random amount from 1 to 5000
set p_h_amount [ RandomNumber 1 5000 ]
#2.5.1.4 date selected from SUT
set h_date [ gettimestamp ]
#2.5.2.1 Payment Transaction
#change following to correct values
if { $ora_compatible eq "true" } {
set result [pg_exec $lda "exec payment($p_w_id,$p_d_id,$p_c_w_id,$p_c_d_id,$p_c_id,
$byname,$p_h_amount,'$name','0',0,TO_TIMESTAMP($h_date,'YYYYMMDDHH24MISS'))" ]
} else {
set result [pg_exec $lda "select payment($p_w_id,$p_d_id,$p_c_w_id,$p_c_d_id,
$p_c_id,$byname,$p_h_amount,'$name','0',0)" ]
}
if {[pg_result $result -status] != "PGRES_TUPLES_OK"} {
if { $RAISEERROR } {
error "[pg_result $result -error]"
} else {
puts "Payment Procedure Error set RAISEERROR for Details"
}
} else {
pg_result $result -clear
}
}
#ORDER_STATUS
proc ostat { lda w_id RAISEERROR ora_compatible } {
#2.5.1.1 select district id randomly from home warehouse where d_w_id = d_id
set d_id [ RandomNumber 1 10 ]
set nrnd [ NURand 255 0 999 123 ]
set name [ randname $nrnd ]
set c_id [ RandomNumber 1 3000 ]
set y [ RandomNumber 1 100 ]
if { $y <= 60 } {
set byname 1
 } else {
set byname 0
set name {}
}
if { $ora_compatible eq "true" } {
set result [pg_exec $lda "exec ostat($w_id,$d_id,$c_id,$byname,'$name')" ]
} else {
set result [pg_exec $lda "select * from ostat($w_id,$d_id,$c_id,$byname,'$name')
as (ol_i_id NUMERIC,  ol_supply_w_id NUMERIC, ol_quantity NUMERIC,
```

```
ol_amount NUMERIC,ol_delivery_d TIMESTAMP,  out_os_c_id INTEGER,
out_os_c_last VARCHAR, os_c_first VARCHAR, os_c_middle VARCHAR, os_c_balance NUMERIC,
os_o_id INTEGER, os_entdate TIMESTAMP,
os_o_carrier_id INTEGER)" ]
}
if {[pg_result $result -status] != "PGRES_TUPLES_OK"} {
if { $RAISEERROR } {
error "[pg_result $result -error]"
} else {
puts "Order Status Procedure Error set RAISEERROR for Details"
}
} else {
pg_result $result -clear
}
}
#DELIVERY
proc delivery { lda w_id RAISEERROR ora_compatible } {
set carrier_id [ RandomNumber 1 10 ]
set date [ gettimestamp ]
if { $ora_compatible eq "true" } {
set result [pg_exec $lda "exec delivery($w_id,$carrier_id,
TO_TIMESTAMP($date,'YYYYMMDDHH24MISS'))" ]
} else {
set result [pg_exec $lda "select delivery($w_id,$carrier_id)" ]
}
if {[pg_result $result -status] ni {"PGRES_TUPLES_OK" "PGRES_COMMAND_OK"}} {
if { $RAISEERROR } {
error "[pg_result $result -error]"
} else {
puts "Delivery Procedure Error set RAISEERROR for Details"
}
} else {
pg_result $result -clear
}
}
#STOCK LEVEL
proc slev { lda w_id stock_level_d_id RAISEERROR ora_compatible } {
set threshold [ RandomNumber 10 20 ]
if { $ora_compatible eq "true" } {
set result [pg_exec $lda "exec slev($w_id,$stock_level_d_id,$threshold)" ]
} else {
set result [pg_exec $lda "select slev($w_id,$stock_level_d_id,$threshold)" ]
}
if {[pg_result $result -status] ni {"PGRES_TUPLES_OK" "PGRES_COMMAND_OK"}} {
if { $RAISEERROR } {
error "[pg_result $result -error]"
} else {
```

```
puts "Stock Level Procedure Error set RAISEERROR for Details"
}
} else {
pg_result $result -clear
}
}
#RUN TPC-C
set lda [ ConnectToPostgres $host $port $user $password $db ]
if { $lda eq "Failed" } {
error "error, the database connection to $host could not be established"
 } else {
if { $ora_compatible eq "true" } {
set result [ pg_exec $lda "exec dbms_output.disable" ]
pg_result $result -clear
}
 }
pg_select $lda "select max(w_id) from warehouse" w_id_input_arr {
set w_id_input $w_id_input_arr(max)
}
#2.4.1.1 set warehouse_id stays constant for a given terminal
set w_id  [ RandomNumber 1 $w_id_input ]
pg_select $lda "select max(d_id) from district" d_id_input_arr {
set d_id_input $d_id_input_arr(max)
}
set stock_level_d_id  [ RandomNumber 1 $d_id_input ]
puts "Processing $total_iterations transactions without output suppressed..."
set abchk 1; set abchk_mx 1024; set hi_t [ expr {pow([ lindex
[ time {if {  [ tsv::get application abort ]  } { break }} ] 0 ],2)}]
for {set it 0} {$it < $total_iterations} {incr it} {
if { [expr {$it % $abchk}] eq 0 } { if { [ time {if {
[ tsv::get application abort ]  } { break }} ] > $hi_t }
{  set  abchk [ expr {min(($abchk * 2), $abchk_mx)}]; set hi_t [ expr {$hi_t * 2}]}}
set choice [ RandomNumber 1 23 ]
if {$choice <= 10} {
if { $KEYANDTHINK } { keytime 18 }
neword $lda $w_id $w_id_input $RAISEERROR $ora_compatible
if { $KEYANDTHINK } { thinktime 12 }
} elseif {$choice <= 20} {
if { $KEYANDTHINK } { keytime 3 }
payment $lda $w_id $w_id_input $RAISEERROR $ora_compatible
if { $KEYANDTHINK } { thinktime 12 }
} elseif {$choice <= 21} {
if { $KEYANDTHINK } { keytime 2 }
delivery $lda $w_id $RAISEERROR $ora_compatible
if { $KEYANDTHINK } { thinktime 10 }
} elseif {$choice <= 22} {
if { $KEYANDTHINK } { keytime 2 }
```

```
slev $lda $w_id $stock_level_d_id $RAISEERROR $ora_compatible
if { $KEYANDTHINK } { thinktime 5 }
} elseif {$choice <= 23} {
if { $KEYANDTHINK } { keytime 2 }
ostat $lda $w_id $RAISEERROR $ora_compatible
if { $KEYANDTHINK } { thinktime 5 }
}
}
pg_disconnect $lda
}
}
```

C.2 Script de Testes HammerDB Modificado

```
{
...

    for {set it 0} {$it < $total_iterations} {incr it}
    {

...

        set choice [ RandomNumber 1 23 ]


        set lda [ ConnectToPostgres $host $port $user $password $db ]
        if { $lda eq "Failed" } {
        error
        "error, the database connection to $host could not be established"
        } else {
        if { $ora_compatible eq "true" } {
        set result [ pg_exec $lda "exec dbms_output.disable" ]
        pg_result $result -clear
        } }


        if {$choice <= 10} {
        if { $KEYANDTHINK } { keytime 18 }
        ostat $lda $w_id $RAISEERROR $ora_compatible
        if { $KEYANDTHINK } { thinktime 12 }


        } elseif {$choice <= 20} {
        if { $KEYANDTHINK } { keytime 3 }
        slev $lda $w_id $stock_level_d_id $RAISEERROR $ora_compatible
         if { $KEYANDTHINK } { thinktime 12 }
```

```
        } elseif {$choice <= 21} {
        if { $KEYANDTHINK } { keytime 2 }
        delivery $lda $w_id $RAISEERROR $ora_compatible
        if { $KEYANDTHINK } { thinktime 10 }


        } elseif {$choice <= 22} {
        if { $KEYANDTHINK } { keytime 2 }
        payment $lda $w_id $w_id_input $RAISEERROR $ora_compatible
        if { $KEYANDTHINK } { thinktime 5 }


        } elseif {$choice <= 23} {
        if { $KEYANDTHINK } { keytime 2 }
        neword $lda $w_id $w_id_input $RAISEERROR $ora_compatible
        if { $KEYANDTHINK } { thinktime 5 }
        pg_disconnect $lda

    }
    pg_disconnect $lda
}
```