OTÁVIO NERY CIPRIANI

REPLICAÇÃO DE BASES DE DADOS POSTGRESQL UTILIZANDO PGCLUSTER

Trabalho de conclusão apresentado ao Curso de Administração em Redes Linux (ARL) da Universidade Federal de Lavras, como parte das exigências do curso de Pós-Graduação Lato Sensu em Administração em Redes Linux, para obtenção do título de Especialização.

LAVRAS
MINAS GERAIS – BRASIL
2009

OTÁVIO NERY CIPRIANI

REPLICAÇÃO DE BASES DE DADOS POSTGRESQL UTILIZANDO PGCLUSTER

Trabalho de conclusão apresentado ao Curso de Administração em Redes Linux (ARL) da Universidade Federal de Lavras, como parte das exigências do curso de Pós-Graduação Lato Sensu em Administração em Redes Linux, para obtenção do título de Especialização.

Orientador:

Prof. Joaquim Quinteiro Uchôa

LAVRAS
MINAS GERAIS – BRASIL
2009

OTÁVIO NERY CIPRIANI

REPLICAÇÃO DE BASES DE DADOS POSTGRESQL UTILIZANDO PGCLUSTER

Trabalho de conclusão apresentado ao Curso de Administração em Redes Linux (ARL) da Universidade Federal de Lavras, como parte das exigências do curso de Pós-Graduação Lato Sensu em Administração em Redes Linux, para obtenção do título de Especialização.

Aprovada em 21 de novembro de 2009.

_____
José Monserrat Neto

_____
Ramon Simões Abílio

_____
Joaquim Quinteiro Uchôa
(Orientador)

LAVRAS
MINAS GERAIS – BRASIL

"Se o cérebro humano fosse tão simples ao ponto em que pudéssemos entendê-lo, nós seríamos tão simples que não conseguiríamos."

(Fortune)

Agradecimentos

Agradeço ao meu orientador, Professor Dr. Joaquim Quinteiro Uchôa, que me auxiliou no desenvolvimento do presente trabalho.

Sumário

LISTA DE FIGURAS

REPLICAÇÃO DE BASES DE DADOS POSTGRESQL UTILIZANDO PGCLUSTER

RESUMO

Este projeto apresenta algumas extensões para o servidor de banco de dados PostgreSQL voltadas para a criação de *clusters* de alta disponibilidade e balanceamento de carga. Apresenta, também, um *cluster* de alta disponibilidade criado utilizando a extensão PGCluster.

Palavras-chave: Replicação, PostgreSQL, PGCluster, cluster.

POSTGRESQL DATABASE REPLICATION WITH PGCLUSTER

ABSTRACT

This paper presents a few extensions to the PostgreSQL database server designed to provide high availability and load balancing clusters. It also presents a high availability cluster based on the PGCluster extension.

Keywords: *Replication, PostgreSQL, PGCluster, cluster.*

1 INTRODUÇÃO

1.1 Considerações Iniciais

Os sistemas computacionais estão cada vez mais presentes nas organizações. Atualmente é inconcebível a existência de uma empresa de médio ou grande porte que não utilize algum tipo de sistema informatizado.

Para garantir a contínua operação dos negócios da empresa é necessário assegurar o funcionamento ininterrupto dos sistemas de informação dos quais eles dependem. Essa garantia geralmente é obtida com a utilização de *clusters* de alta disponibilidade.

Seja trabalhando em *clusters* ou isoladamente, operando em conjunto ou como parte integrante desses sistemas estão os servidores de bancos de dados. PostgreSQL é um servidor de bancos de dados objeto-relacional de código aberto muito utilizado por grande empresas como Yahoo! (COMPUTERWORLD, 2008) e Sony (COMPUTERWORLD, 2006).

1.2 Motivação

Uma vez que os sistemas de informação se tornaram essenciais ao funcionamento de uma empresa de médio ou grande porte, é de extrema importância garantir que esses sistemas funcionem continuamente. Sua indisponibilidade, seja por falha do *hardware* ou do *software*, implica direta ou indiretamente em perda de dinheiro por parte da organização.

Os prejuízos causados pela indisponibilidade serão ainda maiores se esta foi causada pelo colapso do dispositivo que armazenava os dados, o que significa perda de dados importantes. Dependendo de quais informações foram perdidas, a empresa pode até mesmo ser obrigada a encerrar suas atividades.

A garantia de funcionamento ininterrupto dos sistemas de informação e de proteção contra perda de informações por falha no equipamento pode ser implementada utilizando-se um *cluster* de alta disponibilidade.

Tais *clusters* eram privilégio exclusivo de grandes corporações, pois apenas elas podiam arcar com os elevados custos de aquisição, implantação e manutenção, tanto de equipamentos como de *software*. Atualmente, no entanto, como os equipamentos estão se tornando cada vez mais acessíveis e com a grande oferta de *software* gratuitos de qualidade, a criação de *clusters* de alta disponibilidade tornou-se acessível também às médias empresas.

Uma vez que *clusters* estão ao alcance de um número cada vez maior de empresas e que a adoção do servidor de bancos de dados PostgreSQL está aumentando cada vez mais, é necessário um estudo sobre algumas soluções para a utilização deste servidor em ambientes de alta disponibilidade.

1.3 Objetivos

Como o PostgreSQL não possui suporte nativo a *clusters*, diversas extensões foram desenvolvidas com esse objetivo. O presente trabalho preocupa-se em descrever as características de algumas dessas extensões e demonstrar a utilização da extensão PGCluster.

1.4 Estrutura do Trabalho

No Capítulo 2 é feito um levantamento dos pontos chave para o entendimento dos principais conceitos abordados no trabalho através de uma revisão da literatura. Neste capítulo, são abordadas algumas questões sobre *clusters* e seus principais benefícios. Além disso são analisadas as extensões pgpool-II, Postgres-R, Slony-I e PGCluster.

O Capítulo 3 apresenta o desenvolvimento do trabalho propriamente dito. Ele está dividido em Seções que discriminam os materiais utilizados e as fases de desenvolvimento, apontando as estratégias de implementação adotadas e as principais dificuldades encontradas.

No Capítulo 4 são apresentados os resultados obtidos no trabalho bem como uma análise crítica dos mesmos.

Finalmente, no Capítulo 5 é feita uma síntese sobre a importância do trabalho relacionando-a com os resultados obtidos. Além disso, são feitas propostas para trabalhos futuros na mesma área de concentração.

2 REFERENCIAL TEÓRICO

Neste capítulo se encontra a base teórica necessária à compreensão do trabalho. Ele está subdividido em seções que abrangem desde *clusters* até cada uma das extensões mencionadas anteriormente.

2.1 Cluster

Um *cluster* pode ser definido como "um grupo formado por dois ou mais computadores que trabalham em conjunto como se fosse um único sistema". (BATISTA, 2007)

Um dos principais benefícios de um *cluster* é a alta disponibilidade, conseqüência de se ter vários equipamentos desempenhando a mesma função (redundância), permitindo que o sistema continue em operação mesmo que haja falha em um dos computadores do *cluster*, também chamado de nó. Quando ocorre uma interrupção, que pode ou não ser programada, os serviços são migrados do nó defeituoso para os demais nós de modo transparente para o usuário final, permitindo que a aplicação continue normalmente sua execução. (BATISTA, 2007)

Além da alta disponibilidade, *clusters* podem oferecer uma melhora no desempenho das aplicações. Esta melhora no desempenho é obtida através do balanceamento de carga ou da distribuição de tarefas entre as máquinas.

Servidores de bancos de dados podem se beneficiar tanto da alta disponibilidade quanto da melhoria de performance oferecidas pelos *clusters*. A forma mais comum de explorar esses benefícios é utilizando replicação de bases de dados.

2.2 Replicação

Replicação, no contexto de bancos de dados, é cuidar para que duas ou mais instâncias de uma base de dados tenham sempre os mesmos dados (PARTIO, 2007). A replicação de uma base de dados é uma das principais formas de se garantir que os dados estejam a salvo em caso de algum desastre como, por exemplo, o colapso dos discos de uma máquina.

Existem basicamente duas formas de replicação de bases de dados: síncrona e assíncrona. A replicação síncrona é geralmente do tipo mestre/mestre, enquanto que a replicação assíncrona é geralmente do tipo mestre/escravo. Um servidor mestre é aquele que pode realizar operações de leitura e escrita na base de dados, enquanto que um servidor escravo pode apenas realizar operações de leitura.

2.2.1 Replicação Síncrona

Na replicação síncrona os dados presentes na base de dados são sempre exatamente os mesmos em todos os servidores. Neste modo, operações de escrita são distribuídas a todos os nós e são consideradas permanentes apenas depois que todos eles finalizarem a operação. (PARTIO, 2007)

Um dos principais problemas da replicação síncrona é que, para realizar uma operação de escrita, é necessário bloquear todas as tabelas sendo modificadas. Tal operação pode levar muito tempo, dependendo do número de máquinas envolvidas. Como nenhum nó pode realizar qualquer operação nas tabelas bloqueadas, o sistema pode ficar muito tempo sem conseguir responder às consultas dos usuários. (PARTIO, 2007)

Os problemas da replicação síncrona se tornam especialmente evidentes quando os nós que fazem parte do *cluster* se encontram dispersos geograficamente. O tempo gasto com a sincronização dos nós, nestes casos, é tão elevado

que torna soluções típicas impraticáveis, sendo necessária a utilização de técnicas específicas para tais cenários. (AMIR, 2007)

A Figura 2.1 ilustra a troca de mensagem entre elementos de um *cluster* que utiliza replicação síncrona. O cliente envia um comando SQL ao servidor que recebe as solicitações dos clientes e este, por sua vez, envia o comando a um dos nós do *cluster*. O cliente é informado que o comando SQL foi bem sucedido apenas depois que todos os nós do *cluster* processaram com sucesso a solicitação.

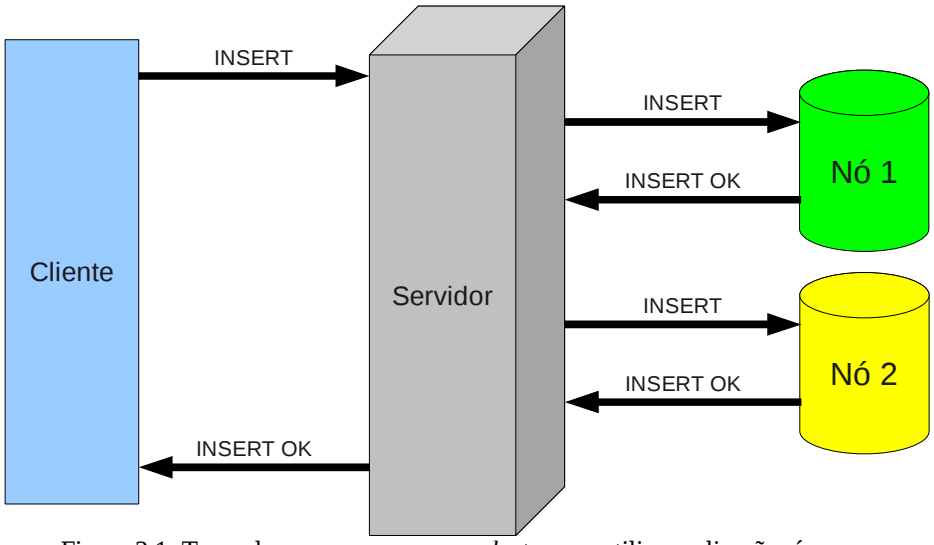


Figura 2.1: Troca de mensagens em um *cluster* que utiliza replicação síncrona

Como a replicação síncrona pode se tornar muito complicada à medida em que mais nós são adicionados ou da distância entre eles, uma técnica alternativa de replicação foi desenvolvida: a replicação assíncrona.

2.2.2 Replicação Assíncrona

A replicação assíncrona geralmente é formada por um nó mestre e um ou mais nós escravos. Todas as operações de escrita são enviadas ao mestre, que as processa e envia periodicamente as modificações realizadas na base de dados, ou

mesmo a base de dados inteira, para os escravos, dependendo da técnica adotada. (PARTIO, 2007)

O tempo de propagação das mudanças realizadas na base de dados do mestre para os escravos pode variar de alguns segundos a várias horas. Durante esse tempo, as bases de dados nos diferentes nós não possuem os mesmos dados, e encontram-se, portanto, em um estado inconsistente. (PARTIO, 2007)

As principais limitações da replicação assíncrona são inerentes ao próprio processo utilizado na replicação. Em soluções com vários mestres, a garantia de consistência da base de dados geralmente fica a cargo do próprio administrador, enquanto que em soluções com um único mestre cria-se um gargalo na performance global do sistema e um ponto único de falha: o próprio mestre. (KEMME, 2000)

A replicação assíncrona apresenta também outros problemas relacionados à consistência das bases de dados. Como as bases de dados são atualizadas utilizando transações separadas, é possível que essas transações sejam executadas em ordem diferente daquela em que foram emitidas pelo cliente, a não ser que seja utilizado algum sistema que garanta essa ordem. Tais sistemas, no entanto são raramente implementados, pois são muito custosos e acabam por neutralizar muitas das vantagens da replicação assíncrona em relação à replicação síncrona. (DAUDJEE, 2004)

A Figura 2.2 ilustra a troca de mensagem entre elementos de um *cluster* que utiliza replicação assíncrona. O cliente envia um comando SQL ao servidor que recebe as solicitações dos clientes e este, por sua vez, envia o comando a um dos nós do *cluster*. O cliente é informado que o comando SQL foi bem sucedido assim que o nó que recebeu a solicitação termina de processá-la com sucesso. Só então o servidor que recebeu a solicitação do cliente envia o comando SQL aos demais nós do *cluster*.

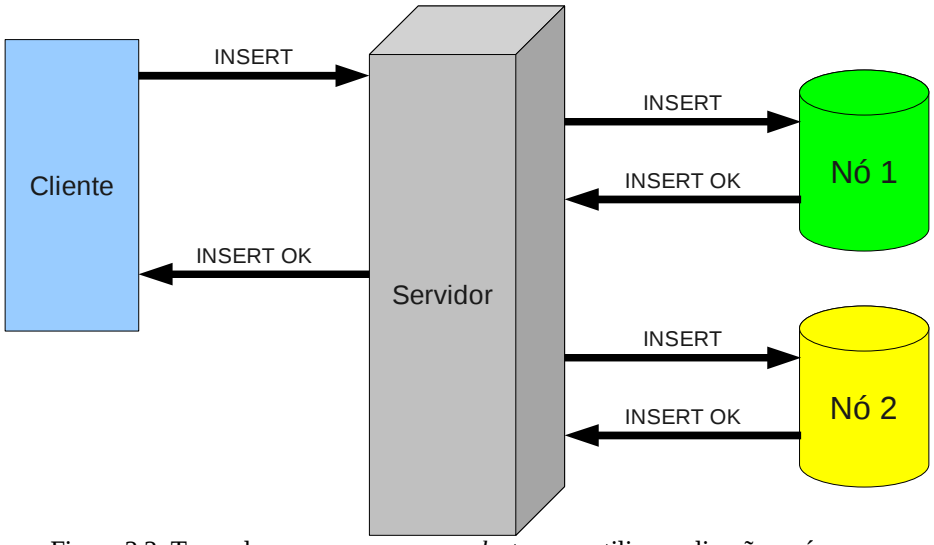


Figura 2.2: Troca de mensagens em um *cluster* que utiliza replicação assíncrona

O servidor de banco de dados PostgreSQL não possui suporte nativo à replicação de dados. Entretanto, diversas extensões ao servidor foram desenvolvidas com esse objetivo, tais como pgpool-II, Postgre-R, Slony-I e PGCluster.

2.3 pgpool-II

Ao contrário das demais extensões mencionadas no presente trabalho, pgpool-II é na realidade um *middleware* entre o servidor de banco de dados PostgreSQL e seus clientes, atuando de forma transparente tanto para o cliente como para o servidor. Isso significa que o cliente enxerga o pgpool-II como se fosse o servidor PostgreSQL, enquanto que o servidor o enxerga como um de seus clientes. Essa abordagem permite que aplicações utilizem o pgpool-II praticamente sem nenhuma modificação em seus códigos. (PGPOOL, 2009)

De acordo com a documentação oficial do pgpool-II (PGPOOL, 2009), os principais benefícios oferecidos por ele são: melhor tratamento de conexões excedentes, pois ao invés de descartar as conexões que ultrapassam o limite máximo

como o servidor PostgreSQL faz, o pgpool-II as coloca em uma fila para processamento posterior; *pool* de conexões, que permite que conexões com as mesmas características (usuário, banco de dados e versão do protocolo utilizado) sejam reutilizadas, reduzindo assim a carga sobre o servidor e melhorando a performance geral; replicação através da execução simultânea de uma operação em diversos servidores; balanceamento de carga através da distribuição de consultas entre os diversos servidores gerenciados pelo pgpool-II; e consultas paralelas, implementadas através da execução simultânea de partes diferentes de uma mesma consulta em servidores diferentes.

Ainda de acordo com a documentação oficial, o pgpool-II pode operar em cinco modos distintos: *raw*, *connection cool*, *replication*, *parallel* e *master/slave*. Uma breve descrição de cada um dos modos de operação, do mais básico ao mais avançado, será dada a seguir:

Em sua configuração mais básica (*Raw Mode*), os clientes simplesmente se conectam ao servidor PostgreSQL através do pgpool-II. Esta configuração é útil quando se deseja apenas limitar o número máximo de conexões simultâneas ao PostgreSQL, ou para permitir que um servidor alternativo assuma em caso de falha do servidor principal.

No modo *pool* de conexões (*Connection Pool Mode*), o pgpool-II atua basicamente da mesma forma como no modo básico (*Raw Mode*). A diferença é que pedidos de conexão que apresentam as mesmas características são reutilizadas. Isso significa, por exemplo, que se determinado usuário estabelecer diversas conexões simultâneas a uma mesma base de dados, elas serão vistas pelo servidor PostgreSQL como uma única conexão, aliviando a carga do sistema.

No modo de replicação (*Replication Mode*) o pgpool-II executa todas as operações em todos os servidores gerenciados por ele, criando uma réplica dos bancos de dados. Desta forma, caso algum dos servidores PostgreSQL pare de

funcionar, o pgpool-II simplesmente o remove temporariamente do *cluster* e o sistema continua funcionando normalmente. Neste modo o pgpool-II também oferece balanceamento de carga distribuindo as consultas entre servidores distintos.

O modo paralelo (*Parallel Mode*) é um modo avançado no qual os dados são distribuídos entre os servidores. Desta forma, cada servidor contém uma parte diferente do banco de dados. A vantagem deste modo sobre o modo de replicação é que, quanto maior o número de servidores no *cluster*, menor o volume de dados que cada servidor precisa armazenar.

O pgpool-II possui ainda o modo mestre/escravo (*Master/Slave Mode*), onde as operações que precisam ser replicadas são passadas ao Mestre, enquanto as outras são distribuídas entre os servidores sempre que possível. Neste modo o pgpool-II opera em conjunto com alguma outra ferramenta de replicação de dados, como o Slony-I. De acordo com [PARTIO, 2007], os ganhos de performance no modo mestre/escravo são superiores aos ganhos obtidos no modo paralelo.

A extensão pgpool-II apresenta uma série de limitações, se comparada a um servidor PostgreSQL padrão. Ela não suporta, por exemplo, um sistema de controle de acesso como o "pg_hba.conf". Caso o acesso via TCP/IP ao servidor PostgreSQL esteja habilitado, pgpool-II aceitará conexões provenientes de qualquer *host*, sendo necessária a utilização de outro método de controle para limitar o acesso (iptables, por exemplo). (PGPOOL, 2009)

Ainda de acordo com a documentação oficial do pgpool-II, existem também diversas outras limitações, dependendo do modo de operação escolhido. No modo de replicação (*replication mode*), por exemplo, não há como garantir que funções que retornam valores diferentes cada vez que são executadas (CURRENT_TIMESTAMP e SERIAL, por exemplo) serão replicadas corretamente em todos os nós do *cluster*. Além disso, tabelas criadas com "CREATE

TEMP TABLE" não são destruídas após o término da sessão, pois, para o nó que atendeu a solicitação, a sessão ainda permanece ativa, graças ao *pool* de conexões. Para corrigir esse problema é necessário destruir explicitamente as tabelas temporárias criadas dentro do bloco de transação.

2.4 Slony-I

O Slony-I é uma extensão ao PostgreSQL que realiza replicação assíncrona entre um mestre e um ou mais escravos, indicado principalmente para uso em *data centers* e para realização de *backups* em tempo real das bases de dados de um servidor. É a solução apontada pelo *site* oficial do PostgreSQL[1] como a extensão mais popular disponível livremente para replicação assíncrona.

Slony-I utiliza *triggers* para realizar a replicação das bases de dados do servidor. Quando uma modificação ocorre, um *trigger* é acionado e grava em um arquivo de *log* as modificações realizadas. Ao detectar as mudanças, através do arquivo de *log*, o mestre notifica todos os escravos e grava o evento de notificação em uma tabela específica (sl_event). Quando um escravo vê a notificação, ele atualiza sua base de dados e notifica o mestre para que este possa remover o evento da tabela. (PARTIO, 2007)

As principais vantagens de Slony-I são a sua fácil implantação e configuração, sólida funcionalidade, técnicas baseadas em padrões SQL existentes e uma ativa comunidade de usuários. Além disso, Slony-I é a única solução para replicação assíncrona em PostgreSQL sendo desenvolvida e mantida atualmente. (PARTIO, 2007)

Dentre as principais desvantagens do Slony-I estão o fato de que ele não realiza automaticamente a detecção de falhas nos nós e nem promove automatica-

1 http://www.postgresql.org/

mente um nó escravo para mestre em caso de falha do mestre original. Uma explicação para isso pode ser obtida no próprio *site* oficial do Slony-I[2]:

> "Determinar quando um nó falha é primariamente responsabilidade de um *software* gerenciador de rede, não de Slony-I. A configuração, caminhos alternativos, e políticas preferenciais serão diferentes em cada local. [...]
>
> Além disso, determinar o que fazer baseado no estado do *cluster* representa política de negócios do local, particularmente em vista do fato de que uma política do tipo FAIL OVER requer que o nó com falha seja descartado. Se Slony-I forçasse essa política, isso poderia conflitar com os requisitos do negócio, tornando Slony-I uma escolha inaceitável." (SLONY-I, 2007)

Slony-I também apresenta desvantagens que estão diretamente relacionadas à sua técnica de replicação baseada em *triggers*. Qualquer limitação imposta pelo servidor aos *triggers* também é uma limitação imposta ao Slony-I. Como PostgreSQL permite *triggers* apenas nas operações INSERT, UPDATE e DELETE, qualquer outra operação, como TRUNCATE, CREATE, DROP ou ALTER, não pode ser replicada. (POSTGRESQL, 2008)

2.5 PGCluster

PGCluster é uma extensão para PostgreSQL que oferece replicação síncrona entre dois ou mais mestres. PGCluster é a solução indicada pelo *site* oficial do PostgreSQL como a solução mais popular disponível livremente para esse tipo de replicação.

2 http://main.slony.info/

PGCluster é composto por três tipos de servidores distintos: o servidor de replicação (*Replication Server*), o balanceador de carga (*Load Balance Server*) e o servidor PostgreSQL em si. O balanceador de carga não é necessário apenas para realizar o balanceamento de carga entre os servidores, mas também para criar um *cluster* de alta disponibilidade. (PGCLUSTER, 2005)

De acordo com a documentação oficial do PGCluster, o servidor de replicação envia uma operação de um servidor para os demais servidores do *cluster*, ao mesmo tempo em que verifica o estado de cada servidor. Caso seja detectado algum nó defeituoso, este é separado do *cluster* até que seja reparado. Quando um novo nó é inserido no *cluster* ou quando um nó é recolocado, o servidor de replicação se encarrega de atualizá-lo.

Ainda de acordo com a documentação oficial, o balanceador de carga recebe uma operação de um cliente e a envia ao servidor que estiver menos carregado. O servidor menos carregado é aquele com o menor número de conexões ativas. De forma semelhante ao servidor de replicação, o balanceador de carga detecta e remove do *cluster* servidores defeituosos.

Um *cluster* PGCluster pode operar em dois modos distintos: normal (*normal mode*) e confiável (*reliable mode*). No modo normal o balanceador de carga retorna o resultado de uma operação para o cliente assim que ele receber a resposta do servidor PostgreSQL que a executou. No modo confiável o balanceador de carga devolve a resposta ao cliente somente depois que a operação foi executada em todos os servidores PostgreSQL do *cluster*. (PGCLUSTER, 2005)

Ao contrário do Slony-I e do pgpool-II, PGCluster é na verdade um *patch* para o código fonte do PostgreSQL. A vantagem desta abordagem é que, ao modificar o código fonte do PostgreSQL, o PGCluster torna-se parte integrante de seu núcleo, permitindo um elevado controle sobre as operações de replicação e balanceamento. Isso evita limitações às suas funcionalidades como as sofridas, por

exemplo, pelo Slony-I. A desvantagem é que o PGCluster precisa ser atualizado sempre que uma nova versão do PostgreSQL é disponibilizada, o que impede que seus usuários atualizem seus servidores até que uma nova versão do PGCluster, baseada na nova versão do PostgreSQL, também seja lançada.

2.6 Postgres-R

Postgres-R é uma extensão ao servidor de banco de dados PostgreSQL que fornece replicação síncrona (vários mestres) e foi projetada para ser o mais transparente possível para o cliente. Comparado a um sistema de banco de dados de um único nó, um *cluster* Postgres-R é mais confiável e pode ser ampliado facilmente, além de ser mais barato e flexível. (POSTGRES-R, 2008)

De acordo com seu *site* oficial, o principal objetivo da extensão Postgres-R é a implantação de um servidor de banco de dados PostgreSQL de alta disponibilidade e com balanceamento de carga sem o uso de qualquer equipamento especial, ou seja, utilizando equipamentos que estão amplamente disponíveis e a um custo acessível.

Assim como a extensão PGCluster, a Postgres-R é um *patch* para o código fonte do PostgreSQL. Postgres-R é disponibilizada sob a mesma licença do servidor PostgreSQL, eliminando completamente quaisquer problemas relacionados a incompatibilidade de licenças entre os códigos. (POSTGRES-R, 2008)

2.6.1 Componentes de um *Cluster* Postgres-R

De acordo com seu *site* oficial, o principal componente do Postgres-R é o gerenciador de replicação. A principal função do gerenciador de replicação é coordenar mensagens entre os nós. Ele organiza e mantém a conexão com o sistema de comunicação de grupo, bem como as conexões aos servidores que processam as transações.

Os servidores que processam as transações, chamados *back-ends*, são capazes de processar apenas uma única transação por vez. Para reproduzir as transações dos nós remotos o gerenciador de replicação inicia *back-ends* remotos. Estes *back-ends* não possuem conexão com um cliente, mas estão sob o controle do gerenciador de replicação. Analogamente, um *back-end* local processa as transações locais e está conectado diretamente a um cliente. (POSTGRES-R, 2008)

2.6.2 Ciclo de Vida de uma Transação Replicada

Transações que realizam apenas leitura na base de dados são processadas localmente e são tratadas da mesma forma que uma transação executada por um servidor PostgreSQL sem replicação. A partir do momento em que uma transação modifica dados na base de dados, os novos dados são coletados como um *writeset*. O *back-end* local continua a coletar os dados e a armazenar as modificações realizadas no *writeset* até receber o *commit* do cliente. (POSTGRES-R, 2008)

Antes de confirmar o *commit* ao cliente, o *back-end* local envia o *writeset* ao gerenciador de replicação, que por sua vez o envia a todos os outros nós do *cluster* utilizando um canal totalmente ordenado do sistema de comunicação do grupo. O *back-end* local que iniciou a transação pode confirmar o *commit* ao cliente assim que receber de volta o *writeset* enviado ao gerenciador de replicação. (POSTGRES-R, 2008)

2.6.3 Resolução de Conflitos

Na Postgres-R, transações de escrita são serializadas e distribuídas na mesma ordem para todos os elementos do *cluster*. Isso garante o sincronismo e a consistência dos nós, pois uma transação que foi bem sucedida em um nó será igualmente bem sucedida em todos os outros nós. Esta técnica evita que um nó

precise esperar pelos demais, permitindo que cada nó trabalhe na maior velocidade possível. (POSTGRES-R, 2008)

3 METODOLOGIA

Este capítulo apresenta, inicialmente, as ferramentas utilizadas durante a execução do projeto. Em seguida são apresentadas a metodologia e as etapas de desenvolvimento do trabalho.

3.1 Objetivo dos Testes

No presente trabalho não serão realizados testes de laboratório com as extensões pgpool-II e Slony-I, pois são soluções baseadas em replicação assíncrona, mais indicada para replicação entre servidores geograficamente dispersos.

O foco deste trabalho são soluções baseadas em replicação síncrona, mais indicadas para replicação entre servidores em rede local, um cenário mais comum entre as empresas brasileiras.

Sendo assim, serão realizados testes apenas com a extensão PGCluster, por ser a solução de replicação síncrona multi-mestre tida como a mais popular. A extensão Postgres-R, por sua vez, é um *software* experimental, não sendo indicada, portanto, para uso em ambientes de produção. Houve uma tentativa de testá-la nesse trabalho, sem sucesso, que será detalhada mais à frente.

O objetivo dos testes será o de avaliar o comportamento da extensão PGCluster em caso de falha do servidor de replicação ou de um nó do *cluster*. Serão avaliados o impacto para o cliente e a consistência da base de dados do nó defeituoso após a falha de cada componente e seu reingresso ao *cluster*.

3.2 Ambiente de Testes

Os testes com a extensão PGCluster foram realizados utilizando quatro máquinas virtuais. Essas máquinas virtuais foram criadas especificamente para a realização dos testes e possuem 256 MB de memória RAM e disco rígido de 8

GB. O sistema operacional instalado nas máquinas virtuais foi o Gentoo Linux (64 *bits*), completamente atualizado com os pacotes estáveis da distribuição até quinze de agosto de 2009.

O aplicativo de virtualização utilizado foi o VirtualBox[3] (Sun Microsystems) versão 3.0.4. A máquina utilizada como hospedeira das máquinas virtuais foi um *notebook* Acer Aspire 5920, com 2 GB de memória RAM e processador Intel Core 2 Duo operando constantemente a 2 GHz (o sistema de otimização de freqüência da CPU de acordo com sua utilização foi desabilitado durante os testes).

A seguir serão apresentados os procedimentos para instalação da extensão Postgres-R e os procedimentos realizados na tentativa, sem sucesso, de colocar o *cluster* em operação para a realização de testes. Na seção 3.4 serão descritos os procedimentos para instalação da extensão PGCluster.

3.3 Instalação do Postgres-R

Como mencionado anteriormente, Postgres-R é disponibilizado como um *patch* para o código fonte do servidor de banco de dados PostgreSQL. O código fonte do servidor foi obtido via CVS, conforme as instruções do Apêndice H do manual oficial do PostgreSQL. (POSTGRESQL, 2008)

A data de referência utilizada para download via CVS foi 2008-11-04, a mesma data da última versão disponível do Postgres-R até o presente momento. Desta forma foi possível garantir que o *patch* do Postgres-R seria aplicado sem problemas, conforme as instruções de instalação do Postgres-R, disponíveis em seu site oficial. (POSTGRES-R, 2008)

3 Site oficial do VirtualBox: http://www.virtualbox.org/

O código fonte do servidor PostgreSQL foi baixado via CVS diretamente do servidor oficial utilizando o comando abaixo:

```
$ cvs -z9 -d
:pserver:anoncvs@anoncvs.postgresql.org:/projects/cvsroot
export -D 2008-11-04 pgsql
```

Após o download do código fonte do servidor PostgreSQL e do *patch* do Postgres-R, aplicou-se o mesmo utilizando o comando abaixo, conforme as instruções no site oficial do Postgres-R:

```
$ patch -p0 < postgres-r-20081104.diff
```

Uma vez aplicado o *patch* do Postgres-R, a compilação do servidor PostgreSQL, agora modificado, foi realizada através dos comandos abaixo[4]:

```
$ autoconf
$ ./configure --enable-replication
$ make
```

Para que o servidor pudesse ser compilado sem erros, foi necessário modificar a linha 600 do arquivo 'src/backend/replication/recovery.c', de forma que a versão do catálogo fornecida pelo PostgreSQL fosse a mesma esperada pelo Postgres-R. A versão fornecida pelo PostgreSQL, neste caso, era a 200811031.

O processo de compilação demorou cerca de 15 minutos e após concluído o servidor recém-compilado foi instalado em '/usr/local/pgsql' com o comando abaixo:

```
$ make install
```

4 A raiz de todos os caminhos relativos é o diretório do código fonte do servidor PostgreSQL.

Após a execução do comando acima, as instruções de instalação descritas no capítulo 15 do manual oficial do PostgreSQL (POSTGRESQL, 2008) foram seguidas, de forma que o servidor pudesse ser iniciado.

3.3.1 Iniciando o *Cluster* de Servidores PostgreSQL com Postgres-R

Apenas as instruções do capítulo 15 do manual oficial do PostgreSQL não são suficientes para permitir a utilização da funcionalidade de replicação. Para utilizá-la é necessário iniciar um sistema de comunicação distribuída, de forma que um servidor PostgreSQL possa se comunicar com os demais.

Inicialmente o servidor de comunicação distribuída que acompanha a Postgres-R, o *Emulated Group Communication System* (EGCS), foi utilizado. O EGCS é executado sobre o Twisted, um sistema de rede orientado a eventos escrito em Python e licenciado sob a licença do MIT (TWISTED, 2009). O comando abaixo instrui o *daemon* do Twisted (twistd) a executar o EGCS (opção "-y src/tools/egcs/egcs.tac") em primeiro plano (opção "-n"), de forma que as mensagens enviadas e recebidas sejam exibidas na console:

```
$ twistd -n -y src/tools/egcs/egcs.tac
```

Uma vez iniciados o servidor de comunicação distribuída e o servidor PostgreSQL, é necessário habilitar a replicação enviando ao principal servidor o comando abaixo, conforme instruções na página oficial do Postgres-R:

```
$ pgsql -c 'ALTER DATABASE shop START REPLICATION IN
GROUP postgresr USING egcs;'
```

Ao executar o comando acima, no entanto, o servidor EGCS terminou inesperadamente com o seguinte erro:

```
global name 'struct' not defined
```

Para corrigir o erro foi necessário inserir a linha "import struct" logo acima da linha 32 do arquivo "src/tools/egcs/egcs.tac" e reiniciar o servidor de comunicação distribuída.

A modificação acima foi suficiente para evitar que o erro anterior ocorresse, mas ainda assim o EGCS não funcionou corretamente. As mensagens não eram encaminhadas aos demais servidores conforme esperado.

Uma vez que o EGCS não funcionou, tentou-se utilizar um servidor de comunicação distribuída alternativo: o Ensemble (ENSEMBLE, 2009). O Ensemble é o único servidor de comunicação distribuída suportado pelo Postgres-R além daquele que o acompanha (EGCS).

O Ensemble, no entanto, não é mantido desde 2005 (a última versão estável data de 26 de julho de 2005) e a última versão pré-compilada disponível data de 28 de agosto de 2002.

Inicialmente tentou-se compilar o Ensemble a partir de seu código-fonte, obtido diretamente do site oficial[5]. No entanto, diversos pré-requisitos para a instalação não puderam ser obtidos. Desta forma, a última versão pré-compilada foi utilizada.

Infelizmente o Ensemble também não funcionou conforme o esperado. As mensagens enviadas não eram distribuídas aos demais servidores. Após diversas tentativas infrutíferas de colocar o Ensemble em funcionamento, a solução de replicação baseada em Postgres-R foi abandonada.

5 Site oficial do Ensemble: http://www.cs.technion.ac.il/dsl/projects/Ensemble/

3.4 Instalação do PGCluster

O PGCluster foi obtido a partir de sua página no site pgFoundry[6]. A versão utilizada foi a 1.9.0rc5, baseada no PostgreSQL versão 8.3.0. A compilação e instalação foi realizada de acordo com as instruções do arquivo "INSTALL", contido no pacote do PGCluster.

Para obter um melhor desempenho do servidor e aproveitar mais adequadamente os recursos do sistema, algumas opções foram passadas ao *script* de configuração do PGCluster: "-O2 -pipe -march=core2", que instruem o compilador a, respectivamente, realizar diversas otimizações no código, utilizar *pipes* ao invés de arquivos temporários durante a compilação e gerar um executável otimizado para os processadores Core 2 da Intel (GCC, 2009); e "--with-system-tzdata='/usr/share/zoneinfo'", que faz com que as informações sobre fusos horários do sistema que se encontram no diretório indicado sejam utilizadas pelo servidor PostgreSQL. O comando utilizado para compilação foi:

```
$ ./configure CFLAGS='-O2 -pipe -march=core2' \
      --with-system-tzdata='/usr/share/zoneinfo'
```

O processo de compilação foi concluído sem erros e demorou cerca de seis minutos. Após compilado, o PGCluster foi testado executando-se o comando "gmake check" como um usuário normal, ainda conforme as instruções do arquivo "INSTALL". A checagem do PGCluster falhou cerca de dois minutos após o início da execução do comando anterior, no momento da criação do banco de dados "regression", com o comando SQL abaixo:

```
CREATE DATABASE "regression" TEMPLATE=template0 \
    ENCODING='SQL_ASCII'
```

6 Site do PGCluster no pgFoundry: http://pgfoundry.org/projects/pgcluster/

O erro apresentado ocorreu porque a configuração inicial do PGCluster não é compatível com o sistema de testes do PostgreSQL, sendo assim a instalação foi realizada, mesmo diante do erro apresentado, com o comando “gmake install-strip”, que instala o PGCluster no diretório padrão (/usr/local/pgsql) removendo dos executáveis gerados quaisquer símbolos necessários apenas para depuração, reduzindo o espaço em disco necessário e aumentando a performance. Após a execução do comando anterior, o diretório de bases de dados foi inicializado com o comando abaixo sendo executado como usuário “postgres”:

```
$ initdb -D /usr/local/pgsql/data
```

Antes de iniciar o *cluster*, este foi configurado editando-se diversos arquivos de configuração, conforme as instruções do arquivo “INSTALL_PGCLUSTER” (Anexo A). Todos os arquivos foram editados de forma a colocar em funcionamento um *cluster*, conforme a Figura 3.1:

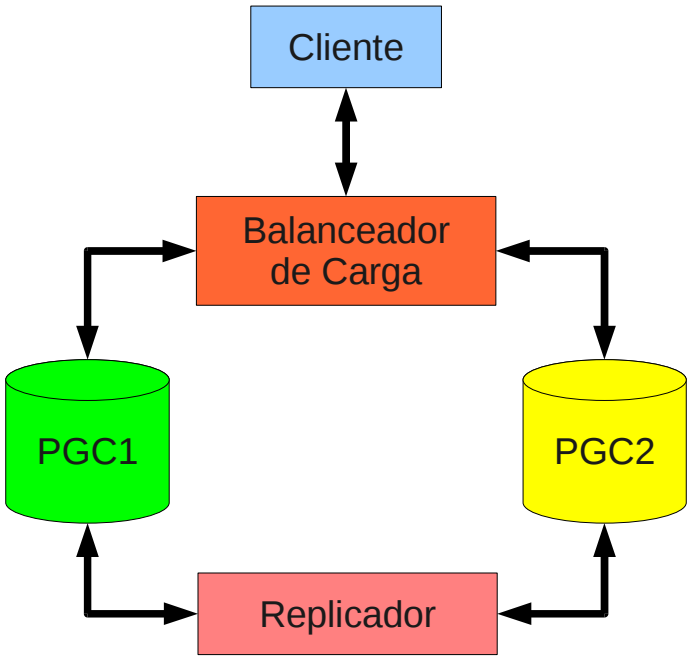


Figura 3.1: Estrutura de um *cluster* PGCluster

Até este momento, apenas uma máquina virtual foi criada e configurada. Embora o processo de instalação e configuração do PGCluster permita que uma única máquina execute todos os componentes do *cluster* (balanceador de carga, replicador e o PostgreSQL), isso anularia todos os benefícios de sua utilização (alta disponibilidade e balanceamento de carga). Sendo assim, após a configuração, a máquina virtual foi clonada três vezes, de forma que cada componente do *cluster* fosse executado em uma máquina separada.

Após os ajustes necessários em cada uma das máquinas, como nome e endereço IP, um único componente do *cluster* foi iniciado em cada uma das máquinas, conforme as instruções do arquivo "INSTALL_PGCLUSTER", tornando-o pronto para uso. Os comandos utilizados para iniciar os dois servidores PostgreSQL, o balanceador de carga e o replicador nos servidores pgc1, pgc2, pglb e pgr, respectivamente, foram:

```
$ pg_ctl start -D /usr/local/pgsql/data -l /tmp/pgc1.log
$ pg_ctl start -D /usr/local/pgsql/data -l /tmp/pgc2.log
$ pglb -D /usr/local/pgsql/data -l
$ pgreplicate -D /usr/local/pgsql/data -l
```

4 RESULTADOS E DISCUSSÃO

Este capítulo apresenta os resultados obtidos nos testes realizados com o *cluster* criado no capítulo anterior.

4.1 Criando e Populando uma Base de Dados

O primeiro teste realizado foi a criação de uma base de dados para testes. Essa base de dados, chamada “test_db”, foi criada utilizando o próprio cliente do PostgreSQL, o “psql”. O comando foi executado no console da máquina virtual utilizada para balanceamento de carga (pglb) como usuário “postgres”, pois, em condições normais de operação do *cluster*, ela é a responsável por receber as solicitações dos clientes. O comando executado foi:

```
$ psql -c “CREATE DATABASE test_db ENCODING='UTF-8';”
```

A base de dados foi criada com sucesso em ambos os servidores PostgreSQL (pgc1 e pgc2).

Uma vez que a base de dados para teste foi criada, o programa Benerator (Databene) versão 0.5.9, um programa gerador de dados pata testes (BENERATOR, 2009) foi utilizado para populá-la. Os dados inseridos na base de dados para teste foram os dados do estágio “perftest” da base de dados “shop”, que acompanha o produto.

O processo de criação da base de dados foi executado duas vezes. Na primeira vez o Benerator foi executado a partir de uma máquina remota que acessava o servidor balanceador de carga (pglb). O processo de criação da base de dados, contendo um total de 175.046 entidades, levou cerca de 74 minutos. Em seguida a base criada foi destruída e gerada novamente executando-se o Benerator diretamente a partir do console do servidor de balanceamento de carga. Neste caso, o processo de criação da base, contendo o mesmo número de itens demorou

cerca de 66 minutos, indicando que a velocidade da rede entre o cliente e o servidor de replicação não exerce influência significativa na velocidade da replicação.

4.2 A Importância do Servidor de Replicação

Para verificar a importância do servidor de replicação (pgr) para o funcionamento do *cluster*, este foi desativado com o comando abaixo:

```
$ pgreplicate -D /usr/local/pgsql/data stop
```

Em seguida, um comando SQL simples foi enviado ao balanceador de carga (pglb) com o comando abaixo, executado a partir do console da própria máquina:

```
$ psql -c “SELECT * FROM db_user;” test_db
```

O comando acima deveria retornar todas as linhas e colunas da tabela “db_user” do banco de dados “test_db”, criado na seção anterior. No entanto, como o servidor de replicação estava desativado, não houve nenhum retorno e a console ficou bloqueada.

Em uma primeira tentativa de desbloquear a console, tentou-se interromper o balanceador de carga com o comando abaixo (executado em outra console da máquina “pgr”):

```
$ pglb -D /usr/local/pgsql/data stop
```

O comando acima não surtiu efeito algum e foi necessário matar o processo responsável pela execução do comando para que a console pudesse ser desbloqueada.

A causa do erro foi facilmente determinada ao analisar a última entrada no *log* do servidor PostgreSQL que recebeu o comando SQL a ser executado (pgc1):

```
WARNING: This query is not permitted without running
replication server
```

Pode-se perceber claramente que o servidor de replicação é de extrema importância para o funcionamento do *cluster*, uma vez que sem ele comandos SQL simples, que não alteram nenhum dado, não podem ser executados. É razoável supor que comando SQL mais complexos também não funcionarão.

Este teste demonstrou também um comportamento não desejável do balanceador de carga, que precisou ser forçosamente interrompido devido a uma falha em outro componente do *cluster*. Emitir uma mensagem de erro ao cliente seria uma ação mais apropriada.

Concluído o teste acima, o replicador e o balanceador de carga foram colocados novamente em operação para os próximos testes.

4.3 Simulando um Nó Defeituoso

De forma a simular uma falha em um dos nós do *cluster*, o procedimento de destruição e criação da base de dados "shop" foi realizado novamente. Desta vez, no entanto, o estágio utilizado foi o "test", reduzindo consideravelmente o número de itens (de 450.000.000 para 24.000) de forma a agilizar a execução dos testes.

Durante o processo de inserção de dados na base de testes, o servidor ao qual o balanceador de carga estava conectado (pgc1) foi abruptamente desligado, simulando uma falha na rede elétrica. O processo de inserção de dados, foi interrompido e uma mensagem de erro informando que houve um erro de comunicação com o servidor de banco de dados foi exibida.

O processo de inserção de dados foi novamente iniciado e uma nova mensagem de erro foi exibida, desta vez informando que o servidor de banco de dados não estava acessível.

O processo de inserção de dados foi mais uma vez iniciado. Desta vez, no entanto, o processo todo correu sem erros. Ao analisar os registros foi possível determinar que o segundo servidor PostgreSQL que fazia parte do *cluster* (pgc2) atendeu e processou com sucesso todas as transações.

Após restabelecer o funcionamento normal do *cluster*, de acordo com a seção 4.4, os testes acima foram realizados novamente. Desta vez, contudo, o nó abruptamente desligado foi aquele que não estava atendendo à solicitação do balanceador de carga, mas apenas estava sincronizando com o outro nó.

Ao desligar o nó que não estava atendendo à solicitação, o processo todo foi concluído com sucesso e a falha do nó não afetou o cliente.

4.4 Recuperando um Nó Defeituoso

Dando continuidade à situação anterior, o nó que havia sido desconectado do *cluster* foi novamente colocado em operação simplesmente reiniciando-se a máquina virtual que o continha e executando-se o comando abaixo para iniciar o servidor PostgreSQL:

```
$ pg_ctl start -D /usr/local/pgsql/data -o '-U' -l
/tmp/pgc1.log
```

O comando acima instrui o servidor PostgreSQL a inciar uma sincronização completa de todas as bases de dados com outro nó do *cluster*, utilizando como ferramenta para tal o utilitário “pg_dump”. O “pg_dump” gera *dumps*, na forma de arquivos contendo comandos SQL, de bases de dados PostgreSQL.

A sincronização com o servidor pgc2, que operava normalmente, foi realizada automaticamente e o processo todo demorou cerca de seis segundos. Logo após o término da sincronização, gerou-se um *dump* da base de dados "test_db" de ambos os servidores e os arquivos criados foram então comparados utilizando-se o comando "diff". O resultado do comando "diff" indicou que os arquivos eram idênticos, mostrando que a sincronização entre os servidores funcionou conforme esperado.

O teste anterior foi realizado novamente, desta vez reiniciando o servidor pgc1 enquanto o servidor pgc2 ainda estava recebendo os dados do balanceador de carga. Ao sincronizar com o servidor pgc2, este aparentemente perdeu comunicação com o balanceador de carga, que por sua vez interrompeu a comunicação com o cliente. Mesmo após reiniciar o balanceador de carga, ambos os servidores PostgreSQL (pgc1 e pgc2) não aceitavam mais qualquer comando que realizasse alterações em suas bases de dados (comandos que apenas selecionavam itens funcionavam normalmente). Foi necessário reiniciar todos os servidores do *cluster* para que este retornasse ao seu funcionamento normal.

4.5 Simulando Várias Conexões Simultâneas

De forma a verificar o comportamento do *cluster* sob situações de *stress*, o utilitário "pgcbench", a versão para *clusters* do "pgbench" (POSTGRESQL, 2008), foi utilizado. O "pgcbench" é capaz de simular diversos clientes conectados ao *cluster*, cada um realizando determinadas operações de inserção, atualização e seleção de dados. As operações que, por padrão, cada cliente realiza são:

```
BEGIN;
UPDATE accounts SET abalance = abalance + :delta WHERE
aid = :aid;
SELECT abalance FROM accounts WHERE aid = :aid;
```

```
UPDATE tellers SET tbalance = tbalance + :delta WHERE tid
= :tid;
UPDATE branches SET bbalance = bbalance + :delta WHERE
bid = :bid;
INSERT INTO history (tid, bid, aid, delta, mtime) VALUES
(:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);
END;
```

O banco de dados para utilização do “pgcbench” foi criado e inicializado com os comandos abaixo:

```
$ psql -c “CREATE DATABASE bmark;”
$ pgcbench -i -s 10 bmark
```

O comando acima populou a base de dados recém-criada “bmark” com as tabelas *accounts*, *branches*, *history* e *tellers*, contendo, respectivamente, 10, 100, 1.000.000 e 0 linhas. O procedimento foi concluído sem erros e o teste de *stress*, que simulava um total de 100 clientes simultâneos, cada um executando um total de 1.000 transações, foi iniciado com o comando abaixo:

```
$ pgcbench -c 100 -t 1000 bmark
```

O comando acima foi executado com êxito e demorou cerca de 35 segundos para ser concluído.

De forma a aumentar a carga sobre o balanceador de carga, o comando acima foi novamente executado, desta vez acrescentando-se a opção “-C” que o instrui a estabelecer uma nova conexão para cada transação de cada cliente, ao invés de apenas uma única conexão para cada cliente. O comando foi concluído com êxito.

O comando anterior foi executado mais uma vez. Durante sua execução, o servidor PostgreSQL pgc2 foi desligado abruptamente, fazendo com que uma série de mensagens de erro fosse exibida na console. Nenhum dos clientes conectados ao pgc1 foi afetado.

A rotina, no entanto, foi concluída com sucesso, pois, quando o utilitário percebia o erro, tentava se conectar novamente. Essa segunda tentativa de conexão era bem sucedida, uma vez que o balanceador de carga automaticamente o redirecionava para o servidor pgc1, que operava normalmente.

5 CONCLUSÃO

No presente trabalho foi feito um levantamento dos pontos chave para o entendimento dos principais conceitos abordados no trabalho através de uma revisão da literatura. Foram abordadas algumas questões sobre *clusters* e seus principais benefícios, além de analisadas algumas características das extensões pgpool-II, Postgres-R, Slony-I e PGCluster.

Através dos testes realizados conclui-se que é possível implementar, sem grandes dificuldades, um *cluster* de alta disponibilidade de servidores PostgreSQL utilizando o PGCluster, garantindo um nível de disponibilidade muito maior em relação a um ambiente composto por apenas um servidor.

Os testes mostraram que os comandos SQL em um *cluster* PGCluster são replicados com perfeição em todos os nós que o compõem. Mostraram também que um nó que apresentou problemas durante a utilização do *cluster* e foi temporariamente removido consegue se atualizar automaticamente e sem erros quando é colocado de volta em operação.

É importante ressaltar, no entanto, que o sistema não é perfeito e pode apresentar falhas sob certas condições, conforme os testes também demonstraram. Felizmente nenhuma das falhas do *cluster* resultou em perda de dados, uma característica muito importante quando se trata de servidores de bancos de dados.

Como sugestão para trabalhos futuros, as soluções de replicação não testadas no presente trabalho (pgpool-II, Postgres-R e Slony-I) poderiam ser testadas e os resultados comparados aos obtidos com a PGCluster, em especial a Postgres-R, por ser também baseada em replicação síncrona.

REFERÊNCIAS BIBLIOGRÁFICAS

AMIR, Y.; COAN, B.; KIRSCH J.; LANE, J.. *Customizable Fault Tolerance for Wide-Area Replication* In: IEEE International Symposium on Reliable Distributed Systems, 26. 2007. Pequim, China. Anais. p. 66-80.

BATISTA, A. C.. *Estudo Teórico sobre Cluster Linux*. 2007. 65 p. Monografia (Pós-Graduação em Administração em Redes Linux) - Universidade Federal de Lavras, Lavras - MG.

COMPUTERWORLD, 2006. Computerworld Inc.. Desenvolvido por: Computerworld Inc., 2006. *Sony Online opts for open-source database over Oracle*. Disponível em: <http://www.computerworld.com/databasetopics/data/software/story/0,10801, 109722,00.html>. Acesso em: 28 de julho de 2008.

COMPUTERWORLD, 2008. Computerworld Inc.. Desenvolvido por: Computerworld Inc., 2008. *Size matters: Yahoo claims 2-petabyte database is world's biggest, busiest*. Disponível em: <http://www.computerworld.com/action/article.do? command=viewArticleBasic&articleId=9087918>. Acesso em: 28 de julho de 2008.

DAUDJEE, K.; SALEM, K.. *Lazy Database Replication with Ordering Guarantees* In: International Conference on Data Engineering, 20. 2004. Boston, Estados Unidos. Anais. p. 424-435.

KEMME, B.; ALONSO, G.. *Don't Be Lazy, Be Consistent: Postgres-R, A New Way to Implement Database Replication* In: VLDB Conference, 26. 2000. Cairo, Egito. Anais. p. 134-143.

PARTIO, M.. *Evaluation of PostgreSQL Replication and Load Balancing Implementations*. 2007. 59 p. Monografia (Graduação em Tecnologia da Informação e Comunicação) - Helsinki Polytechnic Stadia, Helsinque - República da Finlândia.

SLONY-I, 2007. Slony Development Group. Desenvolvido por: Slony Development Group, 2007. *Slony-I Documentation*. Disponível em: <http://main.slony.info/documentation/slonyintro.html>. Acesso em: 29 de julho de 2008.

BENERATOR, 2009. Databene. Desenvolvido por: Volker Bergmann, 2009. *Databene Benerator*. Disponível em: <http://databene.org/databene-benerator.html>. Acesso em: 15 de agosto de 2009.

ENSEMBLE, 2009. Cornell University. Desenvolvido por: Alon Kama, 2005. *The Ensemble Distributed Communication System*. Disponível em: <http://www.cs.technion.ac.il/dsl/projects/Ensemble/>. Acesso em: 29 de março de 2009.

GCC, 2009. Free Software Foundation, Inc.. Desenvolvido por: Free Software Foundation, Inc., 2009. *GCC, the GNU Compiler Collection*. Disponível em: <http://gcc.gnu.org/>. Acesso em: 25 de novembro de 2009.

PGCLUSTER, 2005. PgFoundry. Desenvolvido por: Desenvolvedores do PGCluster, 2005. *PGCluster: The multi-master and synchronous replication system for PostgreSQL*. Disponível em: <http://pgcluster.projects.postgresql.org/>. Acesso em: 30 de julho de 2008.

PGPOOL, 2009. PgFoundry. Desenvolvido por: Desenvolvedores do pgpool, 2009. *pgpool-II User Manual*. Disponível em: <http://pgpool.projects.postgresql.org/pgpool-II/doc/pgpool-en.html>. Acesso em: 12 de novembro de 2009.

POSTGRES-R, 2008. Postgres Global Development Group. Desenvolvido por: Ronja Wanner, 2008. *Postgres-R: a database replication system for PostgreSQL*. Disponível em: <http://www.postgres-r.org>. Acesso em: 12 de outubro de 2008.

POSTGRESQL, 2008. PostgreSQL Global Development Group. Desenvolvido por: PostgreSQL Global Development Group, 2008. *PostgreSQL 8.3.6 Documentation*. Disponível em: <http://www.postgresql.org/files/documentation/pdf/8.3/postgresql-8.3-A4.pdf>. Acesso em: 28 de março de 2009.

TWISTED, 2009. Twisted Matrix Labs. Desenvolvido por: Huw Wilkins, 2009. *Twisted Matrix Labs: Building the Engine of Your Internet*. Disponível em: <http://twistedmatrix.com/trac/>. Acesso em: 25 de novembro de 2009.

ANEXOS

ANEXO A: INSTALL_PGCLUSTER

PGCluster Installation Instructions

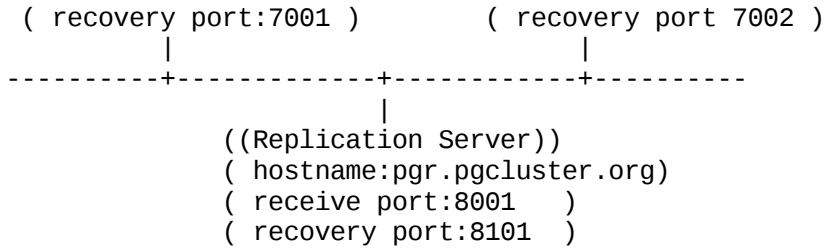
```
============================================================
1. Installation
============================================================

1-1. Install Cluster DB Server, Replication Server & Load Balancer
------------------------------------------------------------
$ cd $source_dir
$ ./configure
$ gmake
$ su
# gmake install
# chown -R postgres /usr/local/pgsql
------------------------------------------------------------

============================================================
2. Initialize DB
============================================================
$ su
# adduser postgres
# mkdir /usr/local/pgsql/data
# chown postgres /usr/local/pgsql/data
# su - postgres
$ /usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data


============================================================
3. Configuration
============================================================
(EX.System Composition)

                       |
                ((Load Balance Server))
                ( hostname: lb.pgcluster.org)
                ( receive port:5432   )
                ( recovery port:6001  )
                       |
----------+-------------+------------+----------
          |                          |
 ((Cluster DB 1))              ((Cluster DB 2))
 ( hostname:c1.pgcluster.org)  ( hostname:c2.pgcluster.org)
 ( receive port: 5432 )        ( receive port:5432  )
```



3-1. Load Balance Server

The setup file of load balance server is copied from the sample file and edited.
(the sample file is installed '/usr/local/pgsql/share' in default)

```
$cd /usr/local/pgsql/share
$cp pglb.conf.sample pglb.conf
```

In the case of the above system composition example,
the setup example of pglb.conf file is as the following

```
#=============================================================
#  Load Balance Server configuration file
#-------------------------------------------------------------
# file: pglb.conf
#-------------------------------------------------------------
# This file controls:
#   o which hosts are db cluster server
#   o which port  use connect to db cluster server
#   o how many connections are allowed on each DB server
#=============================================================
#-------------------------------------------------------------
# set cluster DB server information
#     o Host_Name : hostname
#     o Port : Connection for postmaster
#     o Max_Connection : Maximum number of connection to postmaster
#-------------------------------------------------------------
<Cluster_Server_Info>
    <Host_Name>c1.pgcluster.org</Host_Name>
    <Port>5432</Port>
    <Max_Connect>32</Max_Connect>
</Cluster_Server_Info>
<Cluster_Server_Info>
    <Host_Name>c2.pgcluster.org</Host_Name>
    <Port>5432</Port>
```

```
    <Max_Connect>32</Max_Connect>
</Cluster_Server_Info>
#------------------------------------------------------------
# set Load Balance server information
#   o Host_Name : The host name of this load balance server.
#                 -- please write a host name by FQDN or IP address.
#   o Backend_Socket_Dir : Unix domain socket path for the backend
#   o Receive_Port : Connection from client
#   o Recovery_Port : Connection for recovery process
#   o Max_Cluster_Num : Maximum number of cluster DB servers
#   o Use_Connection_Pooling : Use connection pool [yes/no]
#   o Lifecheck_Timeout : Timeout of the lifecheck response
#   o Lifecheck_Interval : Interval time of the lifecheck
#                          (range 1s - 1h)
#                          10s    -- 10 seconds
#                          10min  -- 10 minutes
#                          1h     -- 1 hours
#------------------------------------------------------------
<Host_Name>lb.pgcluster.org</Host_Name>
<Backend_Socket_Dir>/tmp</Backend_Socket_Dir>
<Receive_Port>5432</Receive_Port>
<Recovery_Port>6001</Recovery_Port>
<Max_Cluster_Num>128</Max_Cluster_Num>
<Use_Connection_Pooling>no</Use_Connection_Pooling>
<LifeCheck_Timeout>3s</LifeCheck_Timeout>
<LifeCheck_Interval>15s</LifeCheck_Interval>
#------------------------------------------------------------
# A setup of a log files
#
# o File_Name : Log file name with full path
# o File_Size : Maximum size of each log files
#               Please specify in a number and unit(K or M)
#               10   -- 10 Byte
#               10K  -- 10 KByte
#               10M  -- 10 MByte
# o Rotate : Rotation times
#            If specified 0, old versions are removed.
#------------------------------------------------------------
<Log_File_Info>
<File_Name>/tmp/pglb.log</File_Name>
<File_Size>1M</File_Size>
<Rotate>3</Rotate>
</Log_File_Info>
```

3-2. Cluster DB Server

The Cluster DB server need edit two configuration files
('pg_hba.conf' and 'cluster.conf').
These files are create under the $PG_DATA directory after 'initdb'.

A. pg_hba.conf
Permission to connect DB via IP connectoins is need for this system.

B. cluster.conf
In the case of the above system composition example,
the setup example of cluster.conf file is as the following

```
#============================================================
#  Cluster DB Server configuration file
#------------------------------------------------------------
# file: cluster.conf
#------------------------------------------------------------
# This file controls:
#   o which hosts & port are replication server
#   o which port use for replication request to replication server
#   o which command use for recovery function
#
#============================================================
#------------------------------------------------------------
# set cluster DB server information
#  o Host_Name : hostname
#  o Port : Connection port for postmaster
#  o Recovery_Port : Connection for recovery process
#------------------------------------------------------------
<Replicate_Server_Info>
<Host_Name>pgr.pgcluster.org</Host_Name>
<Port>8001</Port>
<Recovery_Port>8101</Recovery_Port>
</Replicate_Server_Info>
#------------------------------------------------------------
# set Cluster DB Server information
#  o Host_Name : Host name which connect with replication server
#  o Recovery_Port : Connection port for recovery
#  o Rsync_Path : Path of rsync command
#  o Rsync_Option : File transfer option for rsync
#  o Rsync_Compress : Use compression option for rsync
#                     [yes/no]. default : yes
#  o Pg_Dump_Path : path of pg_dump
#  o When_Stand_Alone : When all replication servers fell,
#                       you can set up two kinds of permittion,
#                       "real_only" or "read_write".
#  o Replication_Timeout : Timeout of each replication request
```

```
#  o Lifecheck_Timeout : Timeout of the lifecheck response
#  o Lifecheck_Interval : Interval time of the lifecheck
#                         (range 1s - 1h)
#                         10s   -- 10 seconds
#                         10min -- 10 minutes
#                         1h    -- 1 hours
#-------------------------------------------------------------
<Host_Name>c1.pgcluster.org</Host_Name>
<Recovery_Port>7001</Recovery_Port>
<Rsync_Path>/usr/bin/rsync</Rsync_Path>
<Rsync_Option>ssh -1</Rsync_Option>
<Rsync_Compress>yes</Rsync_Compress>
<Pg_Dump_Path>/usr/local/pgsql/bin/pg_dump</Pg_Dump_Path>
<When_Stand_Alone>read_only</When_Stand_Alone>
<Replication_Timeout>1min</Replication_Timeout>
<LifeCheck_Timeout>3s</LifeCheck_Timeout>
<LifeCheck_Interval>11s</LifeCheck_Interval>
#-------------------------------------------------------------
# set partitional replicate control information
# set DB name and Table name to stop repication
#   o DB_Name : DB name
#   o Table_Name : Table name
#-------------------------------------------------------------
#<Not_Replicate_Info>
#<DB_Name>test_db</DB_Name>
#<Table_Name> log_table</Table_Name>
#</Not_Replicate_Info>
```

3-3. Replication Server

The setup file of replication server is copied from the sample file and edited.
(the sample file is installed '/usr/local/pgsql/share' in default)

```
$cd /usr/local/pgsql/share
$cp pgreplicate.conf.sample pgreplicate.conf
```

In the case of the above system composition example,
the setup example of pgreplicate.conf file is as the following

```
#=============================================================
# PGReplicate configuration file
#-------------------------------------------------------------
# file: pgreplicate.conf
#-------------------------------------------------------------
# This file controls:
#   o which hosts & port are cluster server
```

```
#   o which port use for replication request from cluster server
#=============================================================
#-------------------------------------------------------------
# set cluster DB server information
#  o Host_Name : hostname
#  o Port : Connection port for postmaster
#  o Recovery_Port : Connection port for recovery
#-------------------------------------------------------------
<Cluster_Server_Info>
    <Host_Name>c1.pgcluster.org</Host_Name>
    <Port>5432</Port>
    <Recovery_Port>7001</Recovery_Port>
</Cluster_Server_Info>
<Cluster_Server_Info>
    <Host_Name>c2.pgcluster.org</Host_Name>
    <Port>5432</Port>
    <Recovery_Port>7001</Recovery_Port>
</Cluster_Server_Info>
#-------------------------------------------------------------
# set Load Balance server information
#  o Host_Name : hostname
#  o Recovery_Port : Connection port for recovery
#-------------------------------------------------------------
<LoadBalance_Server_Info>
    <Host_Name>lb.pgcluster.org</Host_Name>
    <Recovery_Port>6001</Recovery_Port>
</LoadBalance_Server_Info>
#-------------------------------------------------------------
# A setup of the cascade connection between replication servers.
# When you do not use RLOG recovery, you can skip this setup
#
#  o Host_Name : The host name of the upper replication server.
#                Please write a host name by FQDN or IP address.
#  o Port : The connection port with postmaster.
#  o Recovery_Port : The connection port at the time of
#                    a recovery sequence .
#-------------------------------------------------------------
#<Replicate_Server_Info>
#<Host_Name>upper_replicate.pgcluster.org</Host_Name>
#<Port>8002</Port>
#<Recovery_Port>8102</Recovery_Port>
#</Replicate_Server_Info>
#
#-------------------------------------------------------------
# A setup of a replication server
#
```

```
#  o Host_Name : The host name of the this replication server.
#                Please write a host name by FQDN or IP address.
#  o Replicate_Port : Connection port for replication
#  o Recovery_Port : Connection port for recovery
#  o RLOG_Port : Connection port for replication log
#  o Response_mode : Timing which returns a response
#                    - normal   -- return result of DB which received the query
#                    - reliable -- return result after waiting for response of
#                      all Cluster DBs.
#  o Use_Replication_Log : Use replication log
#                          [yes/no]. default : no
#  o Replication_Timeout : Timeout of each replication response
#  o Lifecheck_Timeout : Timeout of the lifecheck response
#  o Lifecheck_Interval : Interval time of the lifecheck
#                         (range 1s - 1h)
#                         10s -- 10 seconds
#                         10min -- 10 minutes
#                         1h -- 1 hours
#-------------------------------------------------------------
<Host_Name>pgr.pgcluster.org</Host_Name>
<Replication_Port>8001</Replication_Port>
<Recovery_Port>8101</Recovery_Port>
<RLOG_Port>8301</RLOG_Port>
<Response_Mode>normal</Response_Mode>
<Use_Replication_Log>no</Use_Replication_Log>
<Replication_Timeout>1min</Replication_Timeout>
<LifeCheck_Timeout>3s</LifeCheck_Timeout>
<LifeCheck_Interval>15s</LifeCheck_Interval>
#-------------------------------------------------------------
# A setup of a log files
#
#  o File_Name : Log file name with full path
#  o File_Size : maximum size of each log files
#                Please specify in a number and unit(K or M)
#                10 -- 10 Byte
#                10K -- 10 KByte
#                10M -- 10 MByte
#  o Rotate : Rotation times
#             If specified 0, old versions are removed.
#-------------------------------------------------------------
<Log_File_Info>
    <File_Name>/tmp/pgreplicate.log</File_Name>
    <File_Size>1M</File_Size>
    <Rotate>3</Rotate>
</Log_File_Info>
=============================================================
```

4. Start Up / Stop

4-1. replication server

A. Start replication server

```
$ /usr/local/pgsql/bin/pgreplicate -D /usr/local/pgsql/etc
```

B. Stop replication server

```
$ /usr/local/pgsql/bin/pgreplicate -D /usr/local/pgsql/etc stop
```

```
usage: pgreplicate [-D path_of_config_file] [-W path_of_work_files]
[-w wait time before fork process][-U login user][-l][-n][-v][-h][stop]
    -l: print error logs in the log file.
    -n: don't run in daemon mode.
    -v: debug mode. need '-n' flag
    -h: print this help
    stop: stop pgreplicate
    (config file default path: ./pgreplicate.conf)
```

4-2. cluster DB server

$PG_HOME = /usr/local/pgsql
$PG_DATA = /usr/local/pgsql/data

A. Start cluster DB server

```
$ /usr/local/pgsql/bin/pg_ctl start -D /usr/local/pgsql/data
```

B. Stop cluster DB server

```
$ /usr/local/pgsql/bin/pg_ctl stop -D /usr/local/pgsql/data
```

C-1. RE start (recovery) cluster DB server with backup

```
$ /usr/local/pgsql/bin/pg_ctl start -D /usr/local/pgsql/data -o "-R"
```

C-2. RE start (recovery) cluster DB server without backup

```
$ /usr/local/pgsql/bin/pg_ctl start -D /usr/local/pgsql/data -o "-r"
```

```
-----------------------------------------------------------------

D. Upgrade cluster DB server with pg_dump
-----------------------------------------------------------------
$ /usr/local/pgsql/bin/pg_ctl start -D /usr/local/pgsql/data -o "-U"
-----------------------------------------------------------------

4-3. load balance server

A. Start load balance server
-----------------------------------------------------------------
$ /usr/local/pgsql/bin/pglb -D /usr/local/pgsql/share
-----------------------------------------------------------------

B. Stop load balance server
-----------------------------------------------------------------
$ /usr/local/pgsql/bin/pglb -D /usr/local/pgsql/share stop
-----------------------------------------------------------------

usage: pglb [-D path_of_config_file] [-W path_of_work_files] [-n][-v][-h][stop]
    -l: print error logs in the log file.
    -n: don't run in daemon mode.
    -v: debug mode. need '-n' flag
    -h: print this help
    stop: stop pglb
    (config file default path: ./pglb.conf)
```