Nesta seção você encontra artigos sobre banco de dados, SQL ou persistência

Replicação no PostgreSQL

Descrição, prós e contras de algumas técnicas disponíveis

De que se trata o artigo?
Demonstração de algumas técnicas de replicação disponíveis no PostgreSQL. Para tanto, serão feitas apresentações rápidas de cada método, focando os prós e contras de cada utilização. Serão mostradas as técnicas de replicação com PGCluster, Slony, entre outros.

Para que serve?
Fornecer um ponto de partida para o conhecimento mais detalhado das técnicas de replicação no PostgreSQL, levando os administradores de bancos de dados a conhecerem um pouco mais sobre cada ferramenta, bem como ajudar na melhor escolha voltada para suas necessidades particulares.

Em que situação o tema é útil?
A replicação é bastante importante em grandes bases de dados que necessitam ter seus dados distribuídos. Esta replicação se dá por questões de segurança, estruturação de uma arquitetura de alta disponibilidade ou prevenção a falhas. Podendo ajudar também na distribuição/particionamento da base de dados para facilidade de acesso entre os diversos setores de um ambiente corporativo.

Clailson de Almeida
clailson@infox.com.br
É Graduado em Tecnologia em Processamento de Dados e pós-graduado em Administração de Banco de Dados pela Universidade Tiradentes (UNIT), no estado de Sergipe. Já atuou como desenvolvedor WEB, com forte utilização de Java. Já trabalhou 2 anos como Analista de Sistemas e possue experiência no desenvolvimento com bancos de dados Oracle, SQL Server e PostgreSQL, sendo que, deste último, tem 6 anos de utilização, dos quais 3 anos e meio como DBA. Atualmente trabalha na empresa INFOX Tecnologia da Informação Ltda., na cidade de Aracaju, Sergipe, como DBA PostgreSQL de sistemas de grande porte, principalmente para o TRF da 5ª. Região. Possui certificação PostgreSQL, pela Pearson Vue, com o título de PostgreSQL CE (PostgreSQL Certified Engineer Open Source Software) 8 Silver".

Com o crescimento dos sistemas de informações e do aumento das bases de dados, torna-se cada vez mais necessária a utilização de técnicas de replicação de dados. Tais técnicas vão desde uma simples cópia de um servidor a outro, até uma estrutura que provê capacidade de alta disponibilidade, recuperação de falhas, balanceamento de carga e etc. Cada uma com sua particularidade de utilização e gerenciamento, com pontos negativos e positivos.

Este artigo demonstrará algumas técnicas de replicação de bases de dados disponíveis no PostgreSQL. Para tanto, serão feitas apresentações rápidas de cada método, focando os prós e contras de cada utilização. Serão mostradas as técnicas de replicação com PGCluster, Slony, replicação dos arquivos de transações (*log shipping*), Sequoia e PGPool.

SQL60.indb 54 17.11.08 10:44:12

PGCluster

O PGCluster é uma ferramenta de replicação do PostgreSQL que permite a montagem de um sistema de replicação síncrono *multi-master*, onde existe uma replicação em tempo real, com a utilização de vários servidores primários. Ele pode ser composto por três tipos de servidores: o de balanceamento de carga, o de armazenamento e o de replicação (ver **Figura 1**).

Pode haver mais de um servidor responsável pelo balanceamento de carga, recebendo as consultas e as encaminhando para os nós de armazenamento com menor carga. O fator de carga de um nó é calculado pelo número de sessões ativas (conexões feitas pelo sistema ou diretamente por usuários que estão realizando alguma tarefa). Ele também tem a função de verificar se há problemas com o nó de armazenamento durante esta comunicação, separando-o se houver falhas.

O servidor de armazenamento, podendo haver mais de um, faz parte do agrupamento do banco de dados (chamado *cluster*). É o servidor que mantém o *back-end* do PostgreSQL, onde os dados podem ser armazenados (caso não se utilize uma unidade externa de armazenamento como um *storage*) e processados.

O servidor de replicação, podendo também haver mais de um, cuida da sincronia dos dados entre os diversos *hosts* de armazenamento, e quando o servidor de replicação principal falha, outro assume o seu lugar.

A replicação pode ser feita de dois modos: modo normal e modo confiável. No modo normal, uma resposta é enviada ao usuário assim que a atualização for executada no nó de armazenamento que a recebe. No modo confiável, a resposta só é enviada após a execução do comando de atualização ter sido realizada em todos os nós de armazenamento.

O servidor de replicação também verifica problemas com o nó de armazenamento durante a comunicação. Se um problema for detectado, o servidor de replicação separa o nó para posterior replicação e uma entrada no arquivo de log é criada. Quando um nó é recolocado ou adicionado ao agrupamento, o servidor de replicação cuida da sincronização dos dados.

Resumo DevMan

Neste artigo, serão demonstradas algumas técnicas de replicação de bases de dados disponíveis no PostgreSQL, focando os prós e contras de técnicas como o PGCluster, ferramenta de replicação que permite a montagem de um sistema de replicação síncrono *multi-master*; o Slony, sistema de replicação *master* para múltiplos *slaves*; a replicação com log shipping, nativa do PostgreSQL; o Sequoia, que utiliza a tecnologia Java e o PGPool.

Veja o que você vai aprender adicionalmente neste artigo:

- O que é um *storage*?
- O que é um IPC?
- RAIDb

Storage: O *storage* (também chamado de *storage área network* - área de armazenamento em rede) é uma rede projetada para agrupar dispositivos de armazenamento (discos rígidos configurados em RAID). Os SANs são mais comuns nos armazenamentos de grande porte e são diferenciados de outras formas de armazenamento em rede pelo método de acesso em baixo nível que eles apresentam. O tráfego de dados nessas redes é bastante similar àqueles usados internamente em discos, como ATA e SCSI. Em uma rede de armazenamento, o servidor envia pedidos por blocos específicos ou segmentos de dados de específicos discos. Esse método é conhecido como *block storage* (armazenamento de blocos). O dispositivo age similarmente a um *driver* interno, acessando o bloco específico e enviando a resposta através da rede. Compartilhar o armazenamento normalmente simplifica a administração e proporciona flexibilidade, uma vez que cabos e dispositivos de armazenamento não precisam ser movidos fisicamente para mudar o armazenamento de um servidor para outro. Os SANs normalmente são construídos em uma infra-estrutura especialmente projetada para comportar grande tráfego de dados originados de armazenamento. Assim, eles proporcionam um acesso mais rápido e estável. A tecnologia mais comum para SAN é a rede de fibra ótica com o conjunto de comandos SCSI. Um canal de fibra ótica SAN padrão é feita de alguns *switches* que estão interligados, formando uma rede.

IPC: A comunicação entre processos é o grupo de mecanismos que permite aos processos em execução a transferirem informações entre si. Essas facilidades permitem aos processos e *threads* compartilharem dados, trocar informações e sincronizar o acesso a recursos compartilhados. O desenvolvimento do sistema de comunicação entre processos (IPC) iniciou nas primeiras versões do UNIX System V, da AT&T. O compartilhamento de uma região de memória entre dois ou mais processos corresponde à forma mais rápida para troca de dados. A zona de memória compartilhada (denominada segmento de memória compartilhada) é utilizada por cada um dos processos como se ela fosse um espaço de endereçamento que pertencesse a cada um dos programas. Em outras palavras, o compartilhamento de memória permite aos processos trabalharem sob um espaço de endereçamento comum em memória virtual. Em conseqüência disso, este mecanismo é dependente da forma de gerenciamento da memória; isto significa que as funcionalidades deste tipo de comunicação entre processos são fortemente ligadas ao tipo de arquitetura (máquina) sobre a qual esta implementação está realizada. A funcionalidade de memória compartilhada serve como meio eficiente de compartilhar dados entre múltiplos processos sem a necessidade de mover os dados de um processo para o outro.

RAIDb: Ao pé da letra RAIDb significa Redundant Arrays of Inexpensive Databases ou Array Redundante de Banco de Dados Baratos. É um padrão que combina várias instâncias de bancos de dados em um conjunto de banco de dados. RAIDb deriva de duas outras tecnologias: o RAID padrão (Redundant Arrays of Independent Disks – Array Redundante de Discos Independentes) e os clusters de computadores. Ele provê melhor performance e tolerância a falhas que um banco de dados isolado, com os benefícios do balanceamento de carga e cache de consultas. O RAIDb é definido em três diferentes níveis com variação de custo, performance e tolerância a falhas. Estes níveis se dividem em: RAIDb-0, para particionamento completo; RAIDb-1, para replicação completa e RAIDb-2, para uma replicação parcial, onde o usuário define o grau de replicação para cada tabela do banco de dados.

SQL60.indb 55 17.11.08 10:44:15

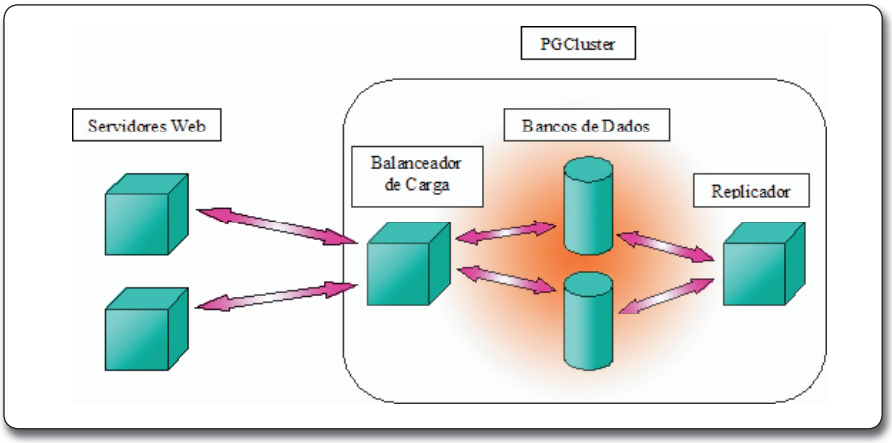

Figura 1. Estrutura do PGCluster para balanceamento de carga e replicação

O PGCluster só pode ser executado em um ambiente Linux, pois o *engine* é derivado de uma modificação no código fonte do PostgreSQL e compilado para este ambiente operacional. Além dessa restrição, existem outras que podem ser mencionadas, como o fato dos *large objects* (tipos de dados que podem guardar objetos muito grandes) que para serem replicados, devem ser colocados em um diretório que possa ser acessado por todos os servidores de armazenamento do *cluster*. O PGCluster também não consegue restaurar um banco de dados quando o mesmo utiliza *tablespaces* (organização lógica do banco de dados, que especifica um local físico no sistema de arquivo, onde os objetos serão armazenados).

Testes de performance foram divulgados comparando-se o PostgreSQL com o PGCluster configurado e o PostgreSQL isolado. Com a utilização do PGCluster, foi observado que, à medida que o número de atualizações aumentava, sendo executadas paralelamente com as consultas, havia uma queda de performance devido ao fato da atualização requerer um bloqueio para processar as atualizações que são replicadas. Esta sobrecarga anulava o efeito da carga de processamento distribuída.

Como alternativa às limitações do PGCluster foi desenvolvido o PGCluster II, que é uma tentativa de reescrever o mesmo, usando um modelo de memória compartilhada. Este projeto foi criado com o objetivo principal de fazer as escritas menos custosas, sem sacrificar a alta disponibilidade nem a performance da leitura de dados, focando um ambiente de sistemas web.

De acordo com a **Figura 2**, o PGCluster II utiliza uma área de armazenamento compartilhada (podendo ser um *storage*) e vários servidores com o *engine* do PostgreSQL (chamado de *postmaster*, que é o nome do serviço) comunicando-se entre eles através de uma memória compartilhada ou IPC (*Inter Process Comunication* – Comunicação Entre Processos).

Por causa dessa arquitetura, as escritas são realizadas aleatoriamente em um nó remoto do *cluster* e replicada para os outros nós, utilizando-se a memória compartilhada. Isto depende de um gerenciamento mais complexo por parte do PGCluster devido ao compartilhamento de semáforos e da própria memória. Já as leituras podem ser feitas diretamente no nó local em que a informação se encontra, aumentando assim a performance.

Existem problemas de sobrecarga na memória compartilhada ocasionados pelos seguintes fatores:

- Acesso demasiado durante a inicialização de cada processo do *back-end*;
- Armazenamento de partes de logs não descarregados, *caches*, *buffers* e etc;
- Divisão em várias partes, contendo vários ponteiros de memória para cada divisão, indicando o endereço de cada processo inicializado e suas estruturas.

Toda essa arquitetura e o gerenciamento da memória aumentam os custos de *hardware*, pois é necessária uma grande quantidade de memória, além da necessidade de ter uma área de armazenamento compartilhada entre os vários *engines*, como o *storage*, que não são baratos.

Outro problema é que, apesar da tentativa de se criar uma nova versão do PGCluster para melhorar a escrita, ainda existem problemas relacionados a este fator devido a ser uma tecnologia nova e ao complexo gerenciamento de memória, segundo estudos divulgados. Além disso, a adição de nós extras no *cluster* não melhora a performance da escrita.

Slony

O Slony é um sistema de replicação *master* para múltiplos *slaves*, suportando a promoção dos *slaves* à *masters* e transferência das informações em cascata. O grande objetivo do desenvolvimento do Slony é construir um sistema *master-slave* que incluísse todas as características e capacidades necessárias para replicar um grande banco de dados para um número razoavelmente limitado de *slaves*. A estrutura do Slony pode ser vista na **Figura 3**.

O Slony foi projetado como um sistema para *data centers* e *sites* de *backup*, onde o modo normal de operação é que todos os nós estejam disponíveis. Períodos prolongados de parada são exigidos para remover ou desativar um nó dentro desta configuração, como uma parada programada de manutenção de algum servidor. Também não é suportada uma configuração em que nós *offline* fiquem disponíveis esporadicamente para sincronização, nem replicação *multi-master* ou replicação síncrona.

A estrutura básica dos sistemas combinados na instalação do Slony é um servidor *master* com um ou mais nós *slaves*. Nem todos os nós *slaves* receberão os dados replicados diretamente do nó *master* (ver **Figura 3**). Os nós *slaves* podem receber de outro *slave*. Isto acontece porque todo nó que recebe os dados de uma fonte válida pode ser configurado para remeter os dados recebidos para outros nós na estrutura.

Os motivos de se ter esta configuração é primeiramente a escalabilidade, onde um servidor, especialmente o *master*, que recebe todas as transações de atualizações das aplicações clientes, tem uma capacidade limitada de processamento

SQL60.indb 56 17.11.08 10:44:19

para satisfazer todos os *slaves* durante o processo de replicação. O segundo motivo é limitar a largura de banda de comunicação na rede necessária para um *site* de *backup*, pois pode-se ter múltiplos *slaves* em um *site* remoto e a replicação é feita para apenas um *slave*, que se encarrega de transferir as mudanças para os outros servidores do *cluster* de *backup*.

O Slony é um sistema de replicação baseado em *triggers* de linha, do tipo AFTER ROW, ou seja, *triggers* disparadas depois que cada linha é modificada. O sistema do Slony analisa as linhas novas e antigas (NEW e OLD respectivamente) para reconstruir o comando SQL executado que representa a mudança na linha modificada.

Para ajudar na identificação de uma linha modificada, a tabela precisa ter algum tipo de coluna chave de registro (uma *primary key* – chave primária, ou uma *unique key* – chave única), podendo ser uma chave combinada de qualquer tipo de dado. Se não existir nenhuma chave, o processo de instalação do Slony irá acrescentar uma coluna do tipo int8 à tabela. Colunas inalteradas em um comando UPDATE não serão incluídas na replicação.

O Slony não é um sistema de gerenciamento de rede, pois o mesmo não tem nenhuma habilidade para descobrir a falha de um nó, nem promover um nó automaticamente à *master*. Determinar se um nó falhou é responsabilidade de um *software* de administração de rede, não do Slony. A configuração de quais caminhos seguir em uma falha e as políticas de preferência de qual nó será promovido a *master* serão diferentes para cada sistema.

Apesar disto, é possível ter uma configuração para desempenhar tanto a tarefa de replicação quanto as tarefas de gerencimento de rede. Para tanto, será necessário a combinação de algumas ferramentas de monitoramento de rede (como o Heartbeat, por exemplo), que avaliem automaticamente os nós que são considerados vivos e quais nós são considerados mortos, junto com alguma política que determine o que fazer quando acontece uma falha.

Há um projeto em desenvolvimento para um sistema subseqüente chamado Slony-II, que pode prover capacidades *multi-master*. Mas isso representa um projeto diferente e separado, sendo implementado em uma perspectiva bastante diferente do que o Slony atualmente é.

A replicação de mudanças de estrutura é outro problema apresentado. O PostgreSQL não possui a habilidade para definir *triggers* disparadas para alterações de estrutura, não existindo assim um modo transparente de reproduzir tais mudanças. Para isto ocorrer, é necessário um trabalho significativo no *engine* do PostgreSQL, alterando o código fonte e recompilando o mesmo. Além das mudanças de estrutura, o Slony também não propaga automaticamente os *large objects*.

A única razão comum para estas limitações é que o Slony coleta informações de atualizações através das *triggers*, e nem as mudanças de estrutura, as operações com *large objects* e as requisições de TRUNCATE (truncar a base de dados) podem ser vinculadas às *triggers*. Como resultado, as únicas mudanças que podem ser replicadas automaticamente são as modificações DML nas tabelas e seqüências (utilizando-se um processo diferente para a replicação das mesmas).

Existe a possibilidade do Slony propagar tais mudanças se elas forem submetidas via *script*, utilizando-se o método interno EXECUTE SCRIPT. Porém, esta operação não é controlada automaticamente e o DBA terá que construir um *script* SQL com os comandos DDL e submetê-lo a este método. Apesar desta possibilidade, existem vários problemas adicionais relacionados a esta operação, que não fazem parte do escopo deste artigo.

Outra limitação é que, à medida que o *cluster* aumenta, ou seja, à medida que mais nós *slaves* são colocados, o custo de comunicação aumenta numa relação de 1:4, pois é necessário ter várias entradas em cada nó que permitam a conexão com todos os outros. É claro que nem todo nó *slave* precisa replicar para o outro, mas a possibilidade existe e depende da necessidade do administrador e da política de prevenção a falhas adotada.

Por esta razão, é uma péssima idéia ter uma grande rede de comunicação resultante de um grande número de nós. Estudos em laboratório comprovam que até meia dúzia de nós pode ser um

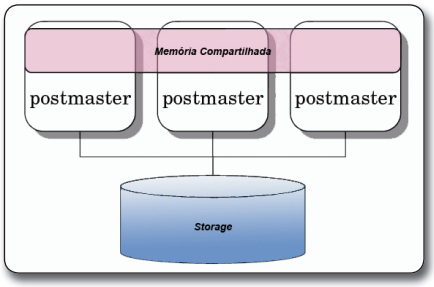

Figura 2. Arquitetura básica do PGCluster II

número bem razoável. Toda vez que o número de nós dobra, é esperado um *overhead* quatro vezes maior nas comunicações entre os servidores.

Replicação dos arquivos de transações (Log Shipping)

Os arquivos de transações no PostgreSQL são chamados de WAL (*write ahead logging* – registro prévio da escrita) e funcionam da mesma forma que qualquer outro SGBD. O conceito central do WAL é que as alterações nos arquivos de dados (onde as tabelas e os índices residem) devem ser escritas somente após estas alterações terem sido registradas, ou seja, quando os registros que descrevem as alterações tiverem sido descarregados em um meio de armazenamento permanente.

Se este procedimento for seguido, não será necessário descarregar as páginas de dados no disco a cada efetivação de transação, porque se sabe que no evento de uma queda será possível recuperar o banco de dados utilizando o registro. Ou seja, todas as alterações que não foram aplicadas às páginas de dados são refeitas a partir dos WAL, com a operação *roll-forward*, também conhecida como REDO.

O primeiro grande benefício da utilização do WAL é a redução significativa do número de escritas em disco, uma vez que, na hora em que a transação é efetivada, somente precisa ser descarregado em disco o arquivo de registro, no lugar de todos os arquivos de dados modificados pela transação. Em ambiente multiusuário, a efetivação de várias transações pode ser feita através de uma única sincronização do arquivo de registro.

Outro benefício é a consistência das páginas de dados. Sem a utilização do

SQL60.indb 57 17.11.08 10:44:22

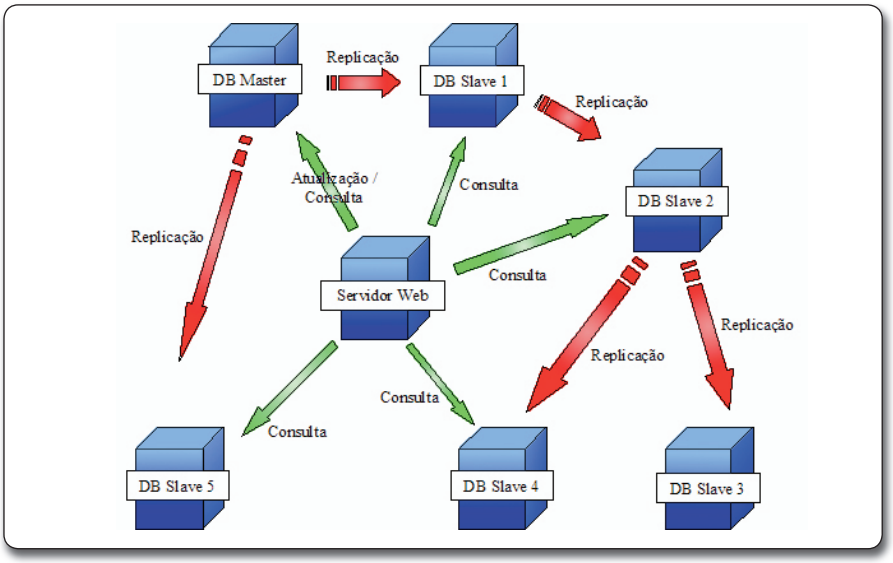

Figura 3. Arquitetura do Slony

WAL, o PostgreSQL não seria capaz de garantir a consistência no caso de uma queda, ou seja, qualquer problema durante a escrita poderia resultar em:

- Linhas de índice apontando para linhas inexistentes da tabela;
- Perda de linhas de índice nas operações de quebra de página (*split*);
- Conteúdo da página da tabela ou do índice totalmente corrompido, por causa das páginas de dados parcialmente escritas.

A partir da versão 8.3.x, o arquivamento contínuo (conhecido como *log shipping*) também pode ser usado para criar uma configuração de agrupamento de servidores (*cluster*) com replicação e alta disponibilidade, com um ou mais servidores *slaves* prontos para assumirem as operações, caso ocorram falhas no servidor *master*. Esta capacidade é chamada de *warm standby*.

Os servidores *master* e *slave* trabalham juntos para prover estas capacidades. O servidor *master* opera em modo normal, com o arquivamento contínuo dos arquivos de transações, enquanto cada servidor *slave* opera em modo de recuperação contínua, recebendo os arquivos WAL e aplicando as alterações contidas neles. Nenhuma mudança nas tabelas ou estrutura do banco de dados é necessária para habilitar esta operação, somente configurações no próprio SGBD. Isto oferece uma baixa complexidade de administração comparada com as outras soluções de replicação existentes. Esta configuração também tem um impacto muito baixo no desempenho do servidor *master*.

Esta replicação de arquivos de transações é feita de maneira assíncrona. Isso quer dizer que os registros de WAL são transferidos depois que a transação é confirmada. Como conseqüência, pode existir uma perda de dados se o servidor *master* sofrer algum problema e as transações registradas nos arquivos WAL ainda não tiverem sido replicadas.

Existem duas soluções para este problema. Uma delas é limitar o tamanho de um parâmetro chamado "archive_timeout". Este parâmetro diz quanto tempo um arquivo WAL pode ficar em aberto, ou seja, depois de quanto tempo um novo arquivo de transação pode ser gerado e o antigo replicado, caso o tamanho do mesmo (que é de 16MB) não seja atingido durante este tempo. Pode ser fixado um valor muito baixo, de alguns segundos, se for preciso. Entretanto, esta configuração aumentará substancialmente a exigência de trafego na rede para a transferência dos WAL, pois uma maior quantidade será replicada por unidade de tempo desejada.

Outra solução é transferir manualmente o último arquivo de transação gerado pelo *master* para o *slave*. O PostgreSQL saberá exatamente quando foi a última transação confirmada válida e aplicará as mudanças no *slave* até o momento da falha do *master*.

Esta técnica replica tanto alterações de dados em comandos UPDATE, DELETE e INSERT, como alterações de estrutura em comandos CREATE, DROP, ALTER, etc. de todas as bases de dados da instância a que se refere. Com isso, qualquer modificação nas bases de dados do *master* é replicada para os slaves, visto que tudo é registrado nos arquivos de transações.

Durante a replicação utilizando este método, o servidor *slave* não fica disponível para acesso, pois o mesmo fica executando em processo contínuo de recuperação. O tempo da recuperação é extremamente pequeno entre o momento que o *slave* termina o processo e se torna completamente *on-line* para uso. Podemos nos referir a esta capacidade como *warm standby*, que oferece alta disponibilidade.

É importante dizer que esta técnica de replicação também não gerencia qual servidor está com problema e nem promove servidores de *slave* para *master*. E neste ponto, vale as mesmas considerações feitas para o Slony, pois é necessário um *middleware* e/ou outras ferramentas de gerenciamento para fazer este chaveamento.

Sequoia

O Sequoia é um *middleware* de *cluster* que permite qualquer aplicação Java (*standalone*, *servlet* ou *container* EJB) acessar, transparentemente, um *cluster* de banco de dados através de JDBC. Para isso, não é necessário realizar modificações nas aplicações clientes, nas aplicações servidoras ou no servidor de banco de dados.

O Sequoia foi desenvolvido para ser usado com aplicações cliente em Java ou aplicações de servidor baseada em Java que acessam um ou mais bancos de dados. Para utilizá-lo, é necessária uma aplicação cliente que acesse um banco de dados através de JDBC, uma máquina virtual Java compilada para JDK 1.4 (ou mais recente), um banco de dados com *driver* JDBC e comunicação de rede com suporte a TCP/IP entre os nós do *cluster*, como pode ser visto na **Figura 4**.

O Sequoia provê escalabilidade de performance pela adição de nós de banco de dados, balanceamento de carga, alta

SQL60.indb 58 17.11.08 10:44:25

disponibilidade, *pool* de conexão transparente, além de dar suporte para *cluster* de bancos heterogêneos, onde podemos utilizar o PostgreSQL com outro tipo de SGBD, como o Oracle e o SQL Server.

O Sequoia utiliza o conceito de RAIDb (*Redundant Array of Inexpensive Databases*). O banco de dados é distribuído e replicado entre vários nós e o Sequoia é responsável por distribuir a carga das consultas entre eles.

A arquitetura é aberta, permitindo que qualquer um adicione customizações como agendadores de requisições, balanceadores de carga, gerenciadores de conexões, políticas de *caching*, entre outras.

De acordo com a arquitetura do Sequoia, é necessária a utilização de uma máquina intermediária para servir como um *middleware* de comunicação entre os clientes e o RAIDb. Outro fator relacionado à arquitetura é a utilização do *driver* JDBC como uma camada intermediária, aumentando assim o consumo de memória em um ambiente que há grande carga de requisições. Pode-se citar também a maior complexidade na administração do mesmo, dependendo da configuração utilizada e das variações do RAIDb.

PGPool

O PGPool é um *middleware* para PostgreSQL, distribuído sob licença BSD, provendo alta disponibilidade, replicação e balanceamento de carga. Além destas características em comum com outros sistemas similares, o PGPool salva as conexões com os servidores PostgreSQL por ele coordenados (trabalhando apenas com dois servidores PostgreSQL), reutilizando-as quando uma nova conexão com as mesmas propriedades (nome de usuário, banco de dados, protocolo e etc.) chega, reduzindo a sobrecarga de conexão e aumentando a taxa de transferência de todo o sistema.

Como sistema de replicação de dados, PGPool permite *backup* em tempo real de bancos de dados, enviando as mesmas declarações SQL para ambos os servidores, podendo ser considerado um sistema de replicação síncrono. Apesar disto, algumas instruções SQL

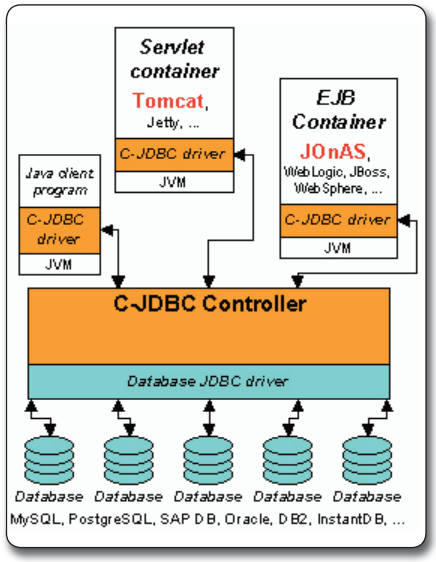

Figura 4. Arquitetura do Sequoiav

são dependentes do servidor no qual são executadas, como funções aleatórias, OID, XID e *timestamp*, não sendo, por isso, replicadas com o mesmo valor para ambos servidores.

Se algum problema torna um dos servidores PostgreSQL indisponível, o

SQL60.indb 59 17.11.08 10:44:32

PGPool tenta continuar o serviço com o servidor ainda ativo, este modo é chamado "modo degenerado". Inconvenientemente, o PGPool não oferece nenhum método de retorno ao *cluster* para um servidor com problemas, sendo necessário que os bancos sejam sincronizados novamente. A melhor maneira é desativar o servidor ativo, sincronizar os arquivos do PostgreSQL (utilizando o comando "rsync" no Linux, por exemplo) e reiniciar os bancos e o PGPool.

Para propósitos de balanceamento de carga, as consultas são distribuídas entre o nó *master* e o *slave* de maneira aleatória para aumentar a performance. É importante notar que mesmo instruções SELECT podem introduzir alterações em bancos, chamando funções que podem modificá-los. Em casos como este, não se deve usar o balanceamento de carga.

Existe também o PGPool-II, um projeto que herdou as características do PGPool, mas que suporta múltiplas instâncias do PostgreSQL (128 nós, expansível via recompilação do código-fonte) e processamento paralelo nos múltiplos nós, o que melhora o desempenho.

Existem alguns problemas relacionados ao uso do PGPool, como a sobrecarga de processos, pois todos os acessos ao PostgreSQL passam pelo PGPool, o que pode reduzir um pouco a performance em casos que existam um grande número de requisições (de 1 à 15%, de acordo com os desenvolvedores do projeto).

Outro problema é que existem limitações quanto aos métodos de autenticação do PostgreSQL, dependendo do modo utilizado no PGPool. Por exemplo, em modo replicação o PGPool só aceita os protocolos "trust" (qualquer conexão é aceita, com qualquer usuário e senha) e "password" (é necessário colocar um usuário e senha válidos, mas a senha é transmitida sem codificação md5).

Além disso, existe ainda o problema com a segurança de acesso ao banco de dados, pois o controle configurado no arquivo "pg_hba.conf" (arquivo de configuração de acesso adicional que diz ao SGBD quais bases podem ser acessadas, por quais usuários, de onde e qual protocolo de acesso) não é suportado, fazendo com que qualquer um possa acessar o PGPool.

Para contornar este problema, pode-se utilizar o "iptables", que é um programa utilizado para configurar regras de filtros de pacotes no Linux 2.4.x e 2.6.x com IPV4 e IPV6. Este programa possui um conceito semelhante ao arquivo "pg_hba.conf" do PostgreSQL, só que mais robusto, com maiores aplicações e podendo ser utilizado como um *firewall*. Mas, mesmo com o "iptable", perdemos uma camada de segurança do próprio SGBD.

Conclusão

Neste artigo conhecemos algumas técnicas de replicação que podem ser utilizadas com o PostgreSQL. Foram mostradas algumas características pertinentes a cada técnica, bem como discutidos alguns prós e contras.

Existem também outros projetos disponíveis para serem utilizados com o PostgreSQL, como o ParGres (destinado a execução paralela de consultas, usando instâncias em nós de *clusters*, através de uma arquitetura distribuída) e o SkyTools (projeto desenvolvido pela empresa Skype para gerenciar seus *clusters* PostgreSQL, sendo uma tecnologia baseada no Slony I).

Além desses projetos, podem ser desenvolvidas soluções genéricas de replicação, como *triggers* que são disparadas nos eventos de UPDATE, DELETE e CREATE e que se comunicam com outro servidor, utilizando a ferramenta dbLink do PostgreSQL. Esta é apenas uma, entre tantas soluções que podem ser desenvolvidas, dependendo apenas da criatividade e capacidade técnica do pessoal responsável pela administração deste SGBD.

Não era intenção deste artigo discutir detalhadamente formas e métodos de utilização, mas servir de ponto de partida para o conhecimento mais detalhado de cada técnica, levando os administradores de bancos de dados e/ou leitores que trabalham com o PostgreSQL a conhecerem um pouco mais sobre cada ferramenta de replicação, bem como ajudar na melhor escolha voltada para suas necessidades particulares. Esperamos que o artigo tenha atingido este propósito. Até a próxima. ●

Links

Site do PGCluster
http://pgcluster.projects.postgresql.org/

Site do proj eto PGCluster no PgFoundry
http://pgfoundry.org/projects/pgcluster/

Site Oficial do PostgreSQL
http://www.postgresql.org/

Site do Slony
http://www.slony.info/

Manual do Usuário de Sequoia
http://sequoia.continuent.org/doc/2.10/userGuide/index.html

Site do projeto PGPoll no PgFoundry
http://pgfoundry.org/projects/pgpool/

Dê seu feedback sobre esta edição!

A SQL Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/sqlmagazine/feedback

SQL60.indb 60 17.11.08 10:44:45