Postgres Load Balancing Benchmark

Grupo 5
Filipe Inês, n78775, º
Pedro Mendes, n97144, pedromendesfc@tecnico.ulisboa.pt

Instituto Superior Técnico, Taguspark, Portugal

Resumo Um distribuidor de carga pode ser usado em bases de dados para oferecer alta disponibilidade e ganho de performance em pedidos de leitura apenas. Este projecto visa analisar a escalabilidade de uma base de dados PostgreSQL com o PGpool-II como distribuidor de carga. Como ferramenta de *benchmark* utilizou-se a ferramenta PGbench. Observando os resultados obtidos é possível concluir que a instalação do PGpool num *cluster* de bases de dados só será benéfica caso este seja alvo, maioritariamente, de transacções de leitura, caso contrário deverá ser dada atenção ao deterioramento da performance das transacções de escrita, tentando procurar soluções para mitigar este problema.

Keywords: Postgres · Scalability · Pgpool · Benchmark · Pgbench.

1 Parte 1

1.1 Introdução

O *paper* escolhido [1] explora a comparação de duas implementações distintas de distribuidores de carga para dois sistemas de base de dados conhecidos, MySQL e PostgreSQL. Porém, como o objectivo deste projecto era apenas analisar a escalabilidade de um sistema que respeitasse um conjunto de requisitos indicados num curto espaço de tempo e dado que o Postgres é um sistema robusto em produção, decidiu-se então focar a análise apenas no caso do PostgreSQL.

A escolha do *paper* deve-se à constante necessidade de sistemas de base de dados, nos dias de hoje, e como tal de que estes se mantenham eficientes e rápidos com o aumento da carga, fazendo com que, neste caso, a replicação e os distribuidores de carga ajudem nesse sentido. Assim, é necessário verificar a escalabilidade de tais sistemas ao introduzir réplicas e/ou distribuidores de carga.

1.2 Âmbito

Este projecto [1] foi desenvolvido no âmbito da unidade curricular de Engenharia de Sistemas de Larga Escala do Instituto Superior Técnico (Taguspark).

[1] Repositório Git - https://github.com/Mendess2526/ESLE/commit/93720d1fa1deb66a1c4b1efd8aaff5d433c3a63d

1.3 Objectivo

O projecto teve como objectivo analisar a escalabilidade de uma base de dados PostgreSQL ao adicionar réplicas da mesma e o distribuidor de carga PGpool-II. Para além disso, destinou-se também à utilização dos conceitos apresentados nas aulas teóricas da unidade curricular.

1.4 Descrição do sistema

PGpool-II é um *middleware* posicionado entre os servidores PostgreSQL e clientes, actuando como cliente para os servidores e como servidor para os clientes. Este *middleware* tem várias funcionalidades que incluem balanceador de carga, replicador de base de dados (para backups e robustez contra falhas), gestor de ligações e sistema de cache para *queries*.

Em geral o PGpool-II coordena um *cluster* de base de dados tendo o porto e o *hostname* de cada uma destas, este *cluster* é formado por um *master* e múltiplos *slaves*. O *master* pode receber transacções do PGpool-II de ambos os tipos (leitura e escrita), enquanto que os *slaves* apenas recebem as transacções de escrita do *master* e as de leitura do PGpool-II. No *cluster* a base de dados é replicada entre todos os *slaves* podendo haver uma distribuição de transacções de leitura por entre os *slaves*.

No âmbito deste trabalho vamos analisar o PGpool-II como balanceador de carga. É necessário notar que este *middleware* apenas tem vantagens de escalonamento apenas sobre transacções de leitura e não a nível de transacções de escrita. Através da figura 1 é possível observar o esquema genérico do sistema e a distribuição dos tipos de transacções.

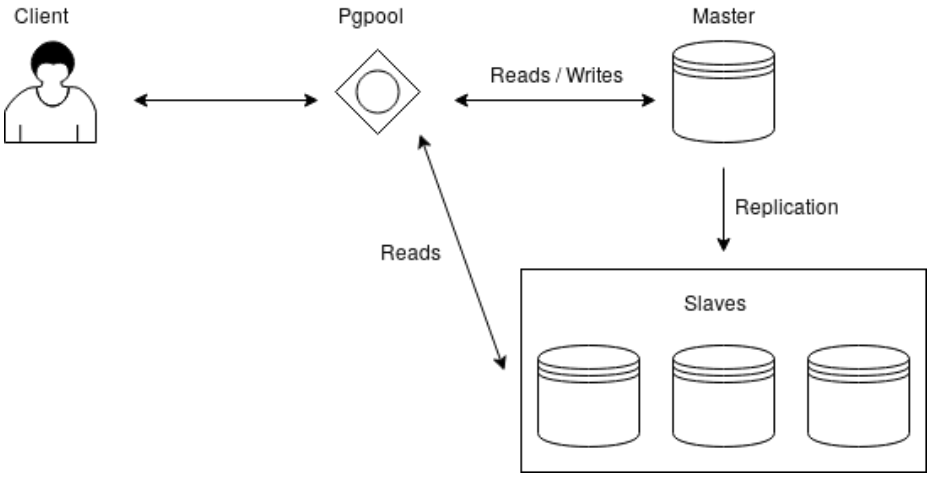

Figura 1. Esquema genérico da ligação entre cliente e servidor PostgreSQL.

1.5 Workload

O *wordload* foi feito usando a ferramenta PGbench, esta baseia-se no TPC-B *benchmark*[2]. Este pode ser visto como um *stress test* de bases de dados, caracterizado por:

- Quantidades significativas de I/O de disco
- Tempo moderado de execução do sistema
- Integridade das transacções

Cada transacção gerada pelo *benchmark* contém cinco operações *SELECT*, *UPDATE* e *INSERT* a não ser que seja seleccionado o modo apenas para leituras em que são feitos apenas operações *SELECT*. Foram realizados testes, com duração de 60 segundos, de escritas e leituras para 1, 5, 10, 15, 20, 25 e 30 clientes em simultâneo para analisar *clusters* com 0, 1, 2 e 3 *slaves*. Os testes foram realizados com uma utilização máxima de 5% do cpu por cada réplica.

Devido à arquitectura do PGPool é necessário fazer os dois tipos de testes em separado (leituras e escritas) para ser possível analisar os benefícios do sistema, uma vez que se verificam maioritariamente nas transacções de leitura onde é possível balancear os pedidos entre os vários *slaves*.

Outro aspecto a notar é a duração, de 60 segundos, já que o PGPool reutiliza as ligações estabelecidas à base de dados. Os benefícios deste aspecto do sistema só se verificam com um tempo de utilização razoável em que haja oportunidade de aplicar este reaproveitamento de ligações múltiplas vezes.

1.6 Resultados

O testes foram feitos com *queries* do tipo leitura e escrita, com duração de 60 segundos, para 1, 5, 10, 15, 20, 30 e 35 clientes em concorrência. Os resultados obtidos correspondem a testes feitos para 1, 2, 3 e 4 réplicas cada uma com 5% de utilização do cpu. A métrica analisada corresponde à latência média dos pedidos que foram executados durante um minuto para cada conjunto de clientes em concorrência.

Para *queries* de escrita não se verifica melhorias (piora até com o aumento do número de *slaves*), isto porque o PGpool não tem nenhum mecanismo de balanceamento para *queries* deste tipo visto que estas transacções têm sempre de chegar a todos as réplicas.

Já para *queries* de leitura são visíveis as melhorias na latência média dos pedidos, visto que para este tipo de pedidos o PGpool faz balanceamento de carga dividindo os pedidos pelas várias réplicas.

Num cenário com apenas um cliente a latência média piora quando são utilizados múltiplos *slaves*, isto acontece uma vez que os pedidos estarão a passar pelo PGpool tendo assim um *overhead* adicional. Ao comparar os casos de um *cluster* com 3 *slaves* e um com 2 *slaves* (Figura 3), concluí-se que a latência média só melhora com 30 clientes, verificando que o aumento de réplicas só compensa aumentando também o *workload*.

[2] http://www.tpc.org/tpcb/

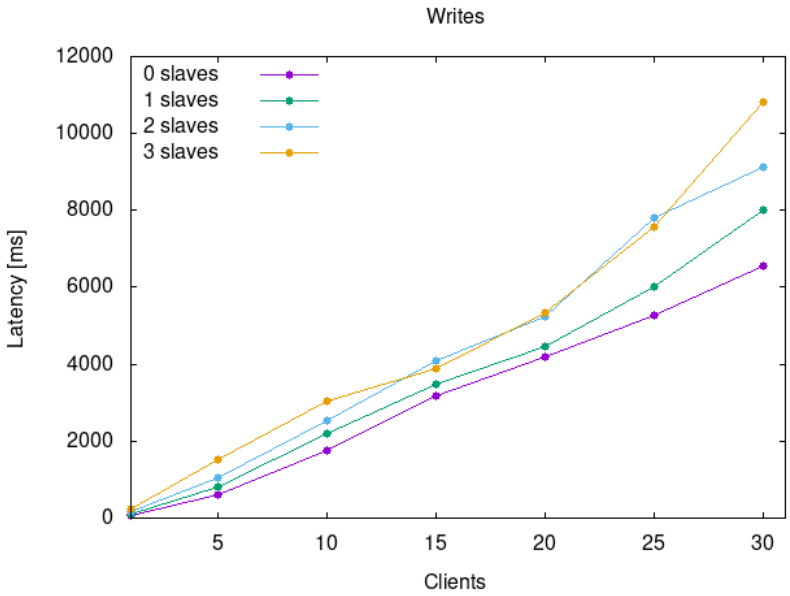

Figura 2. Escritas

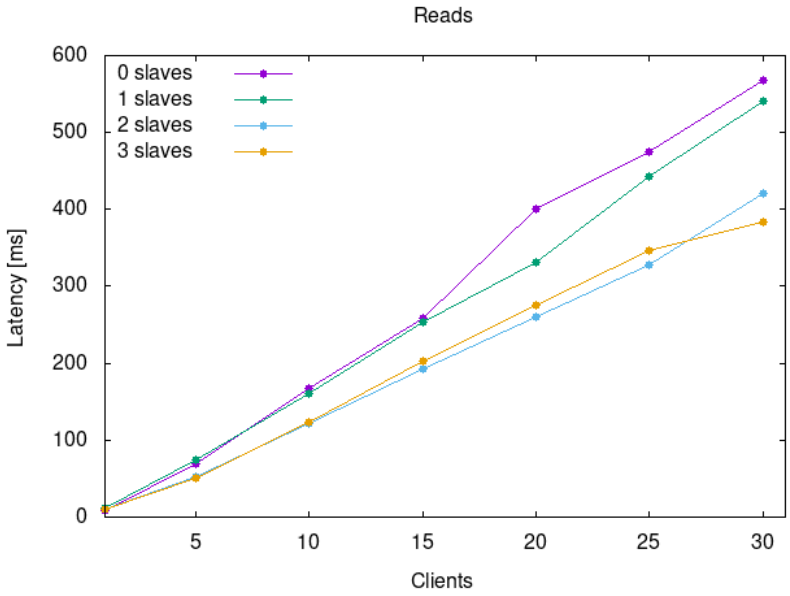

Figura 3. Leituras

1.7 Conclusão

Com os resultados obtidos podemos verificar que para um *cluster* de base de dados, usando o PGpool, o paralelismo obtido para vários clientes concorrentes melhora a latência média das transacções de leitura, verificando que até certo ponto o impacto do *cluster* com o aumento do número de réplicas só melhora a latência para um aumento do *workload*. Esta situação foi verificada para o caso em que a latência de um *cluster* com 3 réplicas só melhorou quando o número de clientes atingiu os 30, sendo pior para os outros casos.

O uso do sistema PGpool melhora apenas a latência média das transacções de leitura, piorando significativamente as transacções de escrita.

Assim, a instalação do PGpool num *cluster* de base de dados só será benéfica caso este seja alvo, maioritariamente, de transacções de leitura, caso contrário deverá ser dada atenção ao deterioramento da performance das transacções de escrita, tentando procurar soluções para mitigar este problema.

2 Parte 2

2.1 Introdução

Nesta segunda parte o PgPool foi analisado em instâncias no aws. Devido a dificuldades técnicas não foi possível correr de forma distribuída em vários nós, ainda assim é possível analisar a *performance* em diferentes instâncias com diferentes recursos e retirar de que modo estes afetam o PgPool. Para tentar simular a distribuição entre várias instâncias foi limitada a utilização do CPU para 10% para os vários containers, esta abordagem tem várias desvantagens em relação à análise distribuída sendo a principal o facto de se ignorar a latência da rede.

2.2 Características das Maquinas

Quatro maquinas diferentes foram utilizadas para realizar as experiências, todas estas a correr linux (kernel 4.15.0-1051-aws).

Maquina	CPU	Disco
1	Intel Xeon E5-2686 v4 octa-core 2.3GHz	EBS, General Purpose SSD
2	Intel Xeon E5-2686 v4 octa-core 2.3GHz	EBS, Magnetic (standard)
3	Intel Xeon E5-2676 v3 dual-core 2.4GHz	EBS, General Purpose SSD
4	Intel Xeon E5-2676 v3 dual-core 2.4GHz	EBS, Magnetic (standard)

Tabela 1. Maquinas utilizadas

2.3 Workload

O *workload* utilizado para as benchmarks foi o seguinte [2]:

Escritas

```
BEGIN;
UPDATE pgbench_accounts SET abalance = abalance + :delta
    WHERE aid = :aid;

SELECT abalance FROM pgbench_accounts
    WHERE aid = :aid;

UPDATE pgbench_tellers SET tbalance = tbalance + :delta
    WHERE tid = :tid;

UPDATE pgbench_branches SET bbalance = bbalance + :delta
    WHERE bid = :bid;

INSERT INTO pgbench_history (tid, bid, aid, delta, mtime)
    VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);

END;
```

Leituras

```
BEGIN;
SELECT abalance FROM pgbench_accounts
    WHERE aid = :aid;
END;
```

Estes *workloads* são repetidos durante 60 segundos tentando realizar o máximo de transacções possíveis.

2.4 Benchmark

Parâmetros Iram ser analisados 5 parâmetros do sistema com dois níveis cada um. Dois parâmetros são intrínsecos ao PgPool enquanto os outros 3 serão recursos das instâncias em que o sistema irá ser analisado.
Parâmetros intrínsecos ao sistema:

- A - Número de nós (2/5)
- D - Connection cache (On / Off)

Parâmetros das máquinas:

- B - CPU Cores (2/8)
- C - Disco (HDD/SSD)
- E - Ram (521Mb / 1Gb)

Fazer um benchmark completo tendo 5 factores com 2 níveis cada implica realizar um total de $2^5 = 32$ experiências. Para simplificar e facilitar a análise reduziu-se o número de experiências para $2^{(5-2)} = 8$. De modo a reduzir o número de experiências 2 dos 5 parâmetros vão ser definidos como combinação de outros dois parâmetros da seguinte da maneira:

Experiência	A	B	C	D ($A.B$)	E ($B.C$)
1	-1	-1	-1	1	1
2	-1	-1	1	1	-1
3	-1	1	-1	-1	-1
4	-1	1	1	-1	1
5	1	-1	-1	-1	1
6	1	-1	1	-1	-1
7	1	1	-1	1	-1
8	1	1	1	1	1

Tabela 2. Sign tables

Métricas: A métricas medidas foram as mesmas analisadas na primeira parte, latência de operações de escrita e leitura. Na primeira parte do projeto chegamos a conclusão dos factores que influenciam positivamente e negativamente a escalabilidade do sistema.
Para operações de leitura a latência melhorava com mais nós e para operações de escrita a latência piorava bastante em relação ao benefícios que o sistema trazia para operações de leitura. Assim o PgPool apenas era benéfico se o sistema realizasse maioritariamente operações de leitura a não ser que fosse possível mitigar o aumento da latência de operações de escrita com o aumento do número de nós, foi possível nesta parte analisar e quantificar de que forma era possível mitigar este problema.

Resultados: Com base nos resultados foi determinado o impacto de cada parâmetro em análise tanto para as operações de escritas como para operações de leitura.

- q0: performance base
- qA: performance associada ao número de nós
- qB: performance associada aos cores utilizados
- qC: performance associada ao tipo de disco
- qD: performance associada ao parâmetro de *connection cache*
- qE: performance associada à RAM

Experiência	Nós	CPU Cores	Disco	Connection Cache	Ram	Leituras[ms]	Escritas[ms]
1	2	2	HDD	ON	1GB	45.80	1273.01
2	2	2	SSD	ON	512Mb	44.43	842.77
3	2	8	HDD	OFF	512Mb	10.815	1248.694
4	2	8	SSD	OFF	1GB	9.878	695.192
5	5	2	HDD	OFF	1GB	48.039	3218.947
6	5	2	SSD	OFF	512Mb	45.447	1615.756
7	5	8	HDD	ON	512Mb	11.293	2156.891
8	5	8	SSD	ON	1GB	24.687	1534.96

Tabela 3. Resultados - Latência

Leituras: Cálculo do peso das *performances* para operações de leitura:

- q0: 30.048
- qA: -2.317
- qB: 15.880
- qC: -1.061
- qD: 1.503
- qE: 2.052

Variação total da latência:

$$SS_T = 8 * (qA^2 + qB^2 + qC^2 + qD^2 + qE^2) = 2121.28 \quad (1)$$

Efeito em percentagem de cada parâmetro no sistema:

- Nós (qA): 2.2%
- CPU Cores (qB): 95.12%
- Disco (qC): 0.43%
- Connection cache (qD): 0.85%
- RAM (qE): 1.50%

Escritas: Cálculo do peso das *performances* para operações de escrita:

- q0: 1573.277
- qA: -558.361
- qB: 164.343
- qC: 401.108
- qD: -121.369
- qE: 107.249

Variação total da latência:

$$SS_T = 8 * (qA^2 + qB^2 + qC^2 + qD^2 + qE^2) = 4207171.70 \quad (2)$$

Efeito em percentagem de cada parâmetro no sistema:

- Nós (qA): 59.28%
- CPU Cores (qB): 5.14%
- Disco (qC): 30.59%
- Connection cache (qD): 2.8%
- RAM (qE): 2.19%

2.5 Conclusões

Como visto já na parte 1 o comportamento do sistema varia bastante conforme a natureza do *workload*, consequentemente o impacto dos vários parâmetros também é diferente para vários tipos de workload. No geral parâmetros como o *connection cache* e RAM não tiveram grande impacto, este facto pode ser explicado pelo facto de o workload usado não ser suficientemente grande para beneficiar destes parâmetros.

Leituras: Para operações de leitura o parâmetro que mais afeta a *performance*, positivamente, é o poder de processamento (número de cores), tendo um impacto de 95%. Isto pode ser explicado pelo facto de o número de cores, tal como o número de nós, aumentar o nível de paralelismo do sistema.

Escritas: Para operações de escrita o parâmetro que mais afetou negativamente o sistema foi o número de nós, tal como explicado já na parte 1 o facto de nas escritas todos os nós terem que estar consistentes para correr as operações seguintes piora a performance do sistema em 59.2%. O tipo de disco foi o segundo parâmetro que afetou mais o sistema positivamente, tendo um impacto de 30.59% na performance. Este comportamento é esperado devido ao facto de a velocidade de leituras/escritas do disco afetar a velocidade de cada transação.

Referências

1. Breuk R. Veerman, G. Database load balancing, mysql 5.5 vs postgresql 9.1. https://www.os3.nl/_media/2011-2012/courses/lia/rory_breuk_gerrie_veerman_-_report.pdf.
2. PostgreSQL Global Development Group. Run a benchmark test on postgresql. https://www.postgresql.org/docs/10/pgbench.html.