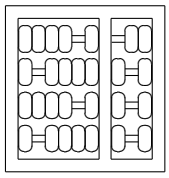

INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

MC437 - Grupo06 - Relatório 3

Gabriel Oliveira *João Fidélis* *Lucas Morais*
Matheus Figueiredo *Pedro Grijó*

Technical Report - IC-16-03 - Relatório Técnico

June - 2016 - Junho

MC437 - Grupo06

Gabriel Bueno de Oliveira João Guilherme Daros Fidélis
Lucas Henrique Morais Matheus Yokoyama Figueiredo
Pedro Rodrigues Grijó

Resumo

Este é o relatório final do projeto da disciplina MC437 (Projeto de Sistemas de Informação), projeto cujo objetivo foi a resolução de um problema de replicação de banco de dados. Foram utilizados dois bancos de dados diferentes, um atuando como primário e outro como secundário. O primário é replicado para um secundário em hot-standby. Ao inserirmos uma falha, promovemos o secundário para primário e fazemos o antigo primário voltar como secundário, sendo que o objetivo foi obter o menor tempo de indisponibilidade possível no sistema. Utilizamos o benchmark TPC-W, que modela uma livraria online, através de um ambiente controlado, para simular atividades num servidor WEB. Em conjunto com o simulador RBE, que gera três diferentes perfis de carga (Shopping, Ordering e Browsing), pudemos checar o desempenho do servidor instalado num cluster no IC, ao verificarmos o número de WIPS (WEB Interactions per Second) e WIRT (WEB Interaction Response Time) gerados por diferentes cargas.

1 Introdução

Este é o relatório final do projeto da disciplina MC437 - Projeto de Sistemas de Informação. O objetivo do projeto foi a definição e implementação de uma solução para o problema de replicação de banco de dados. As cargas de trabalho utilizadas para testar a solução de replicação foram geradas por uma implementação do Transaction Processing Council-Web Benchmark (TPC-W) [1]. A meta final foi obter um sistema com a maior disponibilidade (1) e desempenho (2) possíveis.

$$a = \frac{tempo\_em\_operacao}{tempo\_em\_operacao + tempo\_inoperante} \tag{1}$$

$$d = web\_interactions\_per\_second(WIPS) \tag{2}$$

Onde tempo_em_operação é o período de tempo em que o sistema está apto a responder requisições de seus usuários, tempo_inoperante é o período de tempo em que o sistema é incapaz de produzir respostas corretas para as requisições realizadas pelos usuários e, finalmente, web_interactions_per_second é o número de requisições web por segundo que ocorrem no sistema. Num sistema ideal, $a = 1$ e $d \to \infty$.

Durante a elaboração do projeto, foram implementadas e testadas maneiras de recuperação em caso de falhas num banco de dados. A disponibilidade, dada pela equação (1),

indica que o tempo em que o sistema fica inoperante deve ser o menor possível para que essa seja alta.

Para isso foi aplicada a técnica de Streaming Replication [2], na qual um banco de dados é mantido como primário, recebendo operações de leitura e escrita. Para garantir alta disponibilidade, foi utilizada redundância, mantendo outro banco de dados (em outra máquina) como secundário em hot-standby. Um banco em hot-standby nada mais é do que uma cópia "quente"do banco primário que só faz operações de leitura, de forma que caso o primário seja desligado, o secundário possa ser promovido a primário (passando a permitir também operações de escrita) imediatamente, mantendo o sistema num estado consistente. O Streaming Replication garante que toda operação de escrita feita no banco primário seja repassada e feita também no secundário, para que este possa ser promovido imediatamente, caso necessário. Esta técnica permite que muitos bancos fiquem em standby, garantindo alta disponibilidade e confiablidade ao sistema. No caso deste projeto, foram empregados apenas um banco primário e um banco secundário nos experimentos.

O projeto foi dividido em 3 arcos de desenvolvimento, explicados brevemente a seguir.

1.1 Arco de desenvolvimento 01

No primeiro arco o objetivo foi a configuração do sistema inicial para testar o desempenho e a disponibilidade do mesmo. Não ocorreu injeção de falhas no sistema, apenas foi variada a quantidade de clientes para que fosse obtido um número ideal de WIPS (alto, que não deixe o sistema indisponível) para os próximos testes.

1.2 Arco de desenvolvimento 02

O segundo arco parte dos resultados obtidos no arco 01. A diferença está na injeção de falha. Aqui foram feitos 3 experimentos para medir desempenho e disponibilidade, caracterizados a seguir:

- Apenas com o banco primário funcionando.
- Bancos primário e secundário funcionando.
- Bancos primário e secundário funcionando, injeção de falha no primário e promoção do banco secundário para primário automaticamente.

1.3 Arco de desenvolvimento 03

O terceiro e último arco segue dos resultados obtidos no arco 02. Aqui o diferencial são a detecção de falhas e a promoção automática do banco secundário para primário (e vice-versa) quando necessário. A seguir, este relatório discorre sobre as condições experimentais, metodologia de pesquisa e resultados obtidos neste arco final.

2 Condições Experimentais

Nesta seção serão descritas as configurações de hardware e software utilizadas nos experimentos.

2.1 Plataforma de Testes

Nesta fase do trabalho contamos com quatro máquinas fornecidas pelo Instituto de Computação. As máquinas são iguais do ponto de vista de configuração de hardware, e esta pode ser observada na Tabela 1.

Sistema Operacional	Ubuntu 14.04
CPU	Intel(R) Core(TM)2 Quad CPU Q8400 2.66GHz
Memória RAM	4GB e 1333 MHz de frequência

Tabela 1: Configuração das máquinas remotas

2.2 Arquitetura de Software da Solução

A distribuição dos programas utilizados na solução foi feita conforme consta na Tabela 2.

Máquinas			
CBN6	**CBN7**	**dbmaster2**	**dbslave2**
HAProxy 1.6	RBE	PostgreSQL 9.5.1	PostgreSQL 9.5.1
Apache Tomcat 7	-	-	-
TPC-W	-	-	-

Tabela 2: Softwares em cada máquina do cluster

A **CBN6** atua como servidor web. Nela foi instalado um servidor Apache Tomcat [**?**] e o aplicativo TPC-W, um benchmark de transações web que implementa uma livraria digital.

Para fazer uso do TPC-W foi utilizado o RBE (Remote Browser Emulator) [3], que foi colocado separadamente na **CBN7**. O RBE é um simulador, escrito completamente em Java, que simula o tráfego HTTP gerado por um usuário que estivesse acessando o site através de um navegador. Neste trabalho, sua função foi de emular os conjuntos de clientes que acessam o lado servidor do TPC-W. O motivo que levou à instalação do RBE em uma máquina separada do servidor foi tentar obter um melhor desempenho nos testes, pois desse modo o trabalho ficou dividido entre 4 máquinas ao invés de apenas 3.

Também na **CBN6** foi colocado o HAProxy [4], que atua como um proxy para aplicações baseadas em TCP e HTTP. Ele oferece alta disponibilidade e balanceamento de carga para servidores web.

Nas outras duas máquinas remotas foram instaladas instâncias de bancos de dados PostgreSQL [5], uma com um banco atuando como primário (**dbmaster2**) e outra com um banco secundário (**dslave2**) inicialmente em hot-standby. Assim, o HAProxy foi utilizado para detectar falha do banco primário e redirecionar as requisições feitas para o banco secundário, que deve ser promovido a banco primário.

Ao integrarmos essas funcionalidades, conseguimos emular um site de compras com dados gerados aleatoriamente.

O TPC-W gera as duas métricas em que nos baseamos para avaliar o desempenho do sistema: WIPS (Web Interactions per Second) e WIRT (Web Interaction Response Time). A carga de trabalho gerada pelo RBE pode ser de três perfis: *shopping*(WIPS), *browsing*(WIPSb) ou *ordering*(WIPSo). Cada perfil corresponde a uma porcentagem diferente de operações de leitura e escrita no banco de dados, como exemplificado na Tabela 3.

Perfil	Leitura	Escrita
WIPSb	95%	5%
WIPS	80%	20%
WIPSo	50%	50%

Tabela 3: Descrição das operações dos perfis do RBE

Rodamos o RBE com os três perfis variando o parâmetro Think-Time. O Think-Time é o tempo que um usuário simulado pelo RBE gasta "pensando", ou seja, quanto menor o Think-Time, mais requisições são enviadas ao servidor pelos usuários no mesmo período de tempo de experimento.

3 Metodologia de Pesquisa

Após todo o sistema estar rodando de maneira estável, objetivo atingido no primeiro arco, foram realizados experimentos no segundo arco para determinar uma carga que o sistema conseguisse atender. No primeiro experimento, o RBE foi utilizado com apenas o banco primário ligado. No segundo, estavam ligados o banco primário e o banco secundário em hot-standby.

Esses dois experimentos foram realizados para caracterizar o servidor. O objetivo foi medir a maior carga para a qual ele se comportasse de forma consistente. Para isso, os valores de carga foram variados entre 2000 e 4000 clientes, variando de 1000 em 1000 para cada iteração para cada perfil de usuário descrito na Tabela 3. Assim, gráficos foram gerados para demonstrar até qual carga o servidor se manteve num bom nível de WIPS ao longo de todo o teste (que dura 100s no total).

O valor encontrado foi utilizado no RBE para simular essa "carga ótima"no servidor, causando uma falha manualmente (isto é, "matando"o banco de dados primário), enquanto o banco de dados secundário era promovido. O objetivo foi checar em quanto tempo o servidor conseguiria se recuperar e voltar a ficar ativo.

Os parâmetros utilizados no RBE foram:

Ramp-Up	5s
Ramp-down	5s
Measurement Interval	90s
Número máximo de erros	0
Think-Time	1s

Tabela 4: Parâmetros dos experimentos do RBE

Após o estudo do comportamento do servidor perante variações de carga, foi observado que a diminuiição do Think-Time levava à instabilidade do sistema, mesmo com a "carga ótima"encontrada anteriormente.

Devido a isso, no terceiro arco foi decidido que seria aplicada metade da "carga ótima"para os próximos passos, pois partindo de um valor de Think-Time igual a metade do valor utilizado anteriormente (de 1s para 0.5s), o número de requisições feitas seria o mesmo.

O último experimento, o completo, consistiu na inserção de uma falha na **dbmaster2**. Essa máquina, inicialmente contendo o banco primário, era então substituída pela **dbslave2**, ou seja, **dbslave2** era promovida a banco primário. Após a **dbmaster2** se recuperar, esta voltava como banco secundário. Depois de alguns segundos, uma outra falha era inserida, agora na **dbslave2**, que após sua queda era substituída pela **dbmaster2** (promovida à banco primário) e, finalmente, após se recuperar a **dbslave2** voltava como banco secundário, como no início do experimeno.

Para inserir as falhas, foi utilizado o comando stop do PostgreSQL para parar o banco de dados. Quando a falha ocorre, o HAProxy começa a redirecionar as requisições do banco primário para o banco secundário, que por sua vez está sendo promovido à primário por um script. Ao mesmo tempo, um outro script foi utilizado para checar o status dos bancos de dados. Ao detectar que um deles falhou, o script muda o arquivo de configuração do HAProxy para informar ao programa qual o novo banco de dados primário e qual o novo secundário. O banco de dados que foi derrubado é religado, agora como banco secundário. Isso pode acontecer quantas vezes forem necessárias. No experimento realizado, cada banco é pelo menos uma vez o banco primário e é derrubado para retornar como secundário.

Assim, com carga de 1000 usuários, o Think-Time foi variado de 0.5s a 0.1s. Isso quer dizer que seriam feitas muito mais requisições que a mesma carga sendo aplicada com 2000 usuários e Think-Time de 1s.

4 Análise e Resultados

Seguem os gráficos gerados pela execução do RBE para o experimento 1.

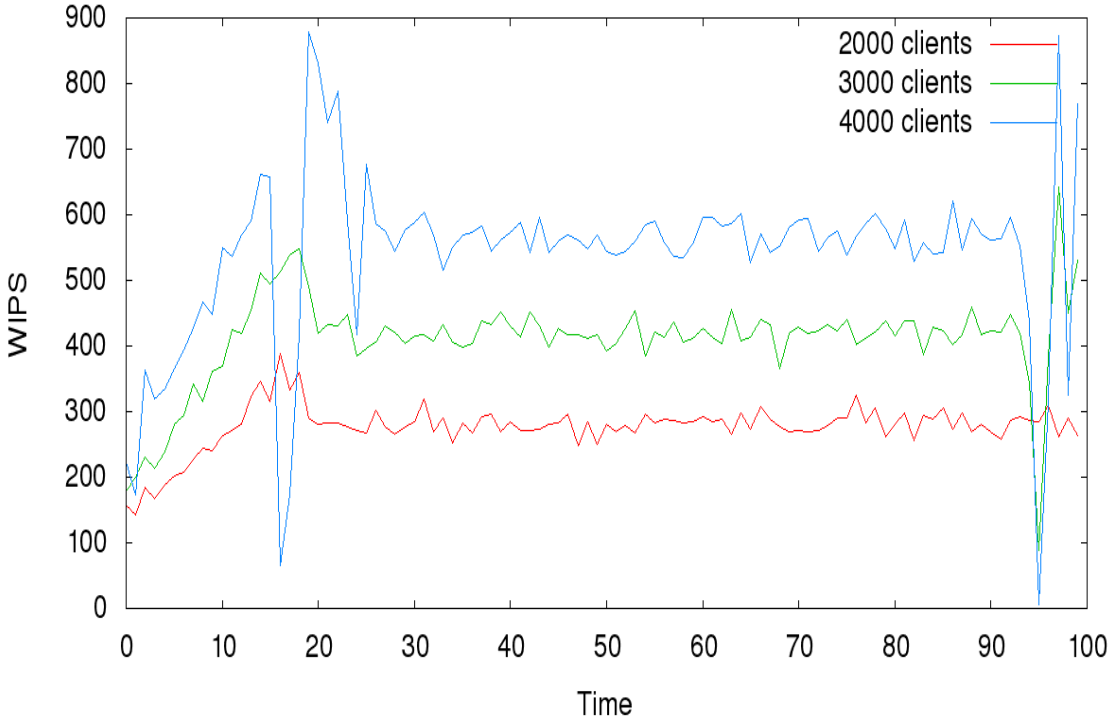

Figura 1: WIPSb por tempo(s) para cada carga simulada pelo RBE no perfil Browsing do experimento 1

Nota-se que o experimento com 4000 clientes teve uma boa consistência entre 30 e 90 segundos, porém houve uma grande queda por volta dos 15s e a partir dos 90s. O mesmo ocorreu no caso para 3000 clientes a partir de 90s.

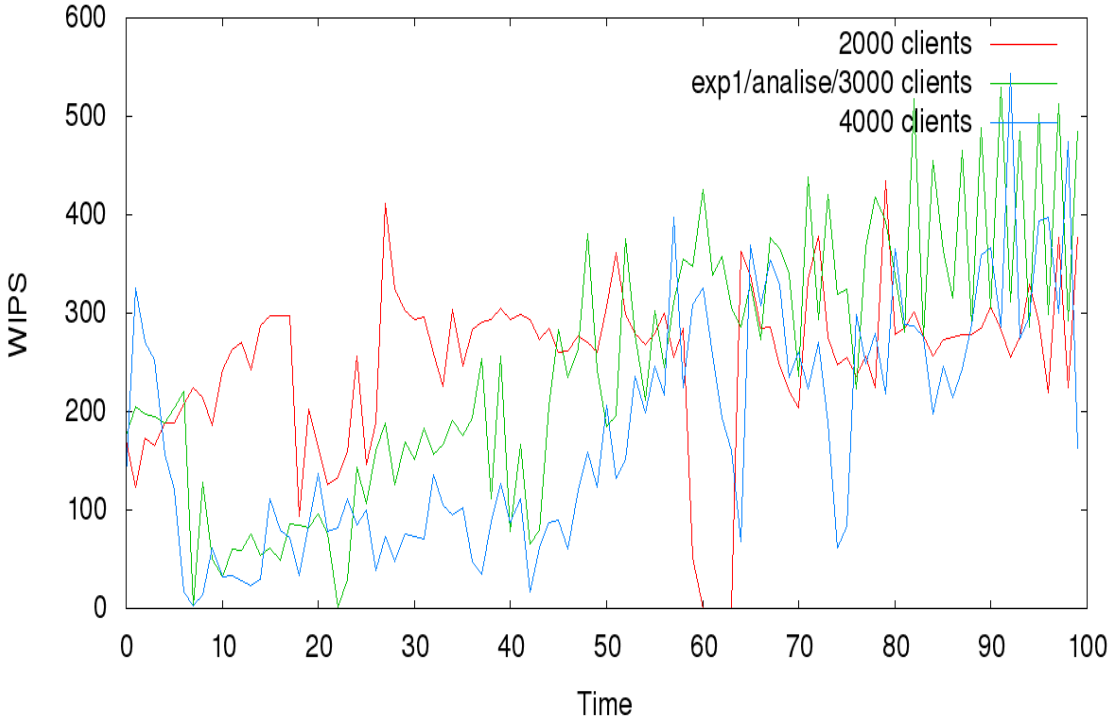

Figura 2: WIPSo por tempo(s) para cada carga simulada pelo RBE no perfil Ordering do experimento 1

Pelo gráfico, é visível que o teste com 3000 clientes obteve o melhor desempenho no geral. Até 50s, o desempenho era melhor com 2000 clientes, porém após esse tempo o teste com 3000 clientes obteve uma quantidade maior de WIPSo e se manteve consistentemente melhor que os testes com 2000 e 4000 clientes.

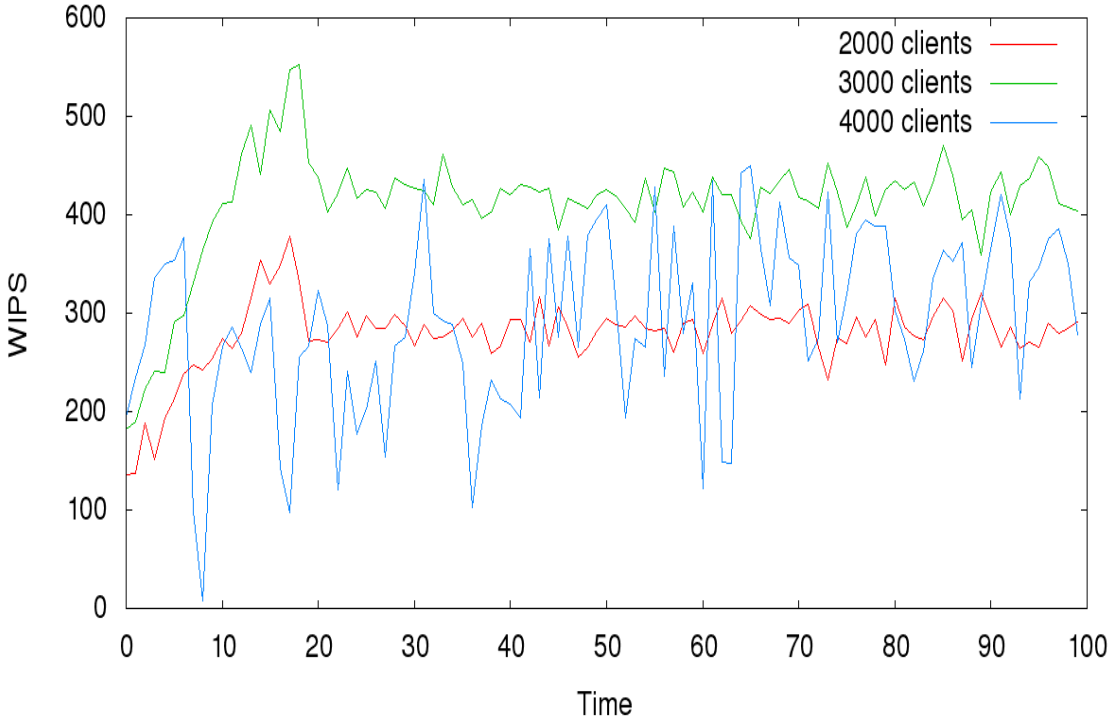

Figura 3: WIPS por tempo(s) para cada carga simulada pelo RBE no perfil Shopping do experimento 1

Neste gráfico, ficou evidente o melhor desempenho da simulação com 3000 clientes sobre as duas outras. Uma melhor taxa de WIPS foi obtida durante praticamente todo o experimento.

Com a combinação dos resultados desses gráficos, temos indícios de que 3000 usuários talvez seja o melhor número a ser utilizado, pois tem taxas de WIPS mais altas e estáveis que as outras quantidades de usuários testadas.

Agora, analisaremos os dados do experimento 2.

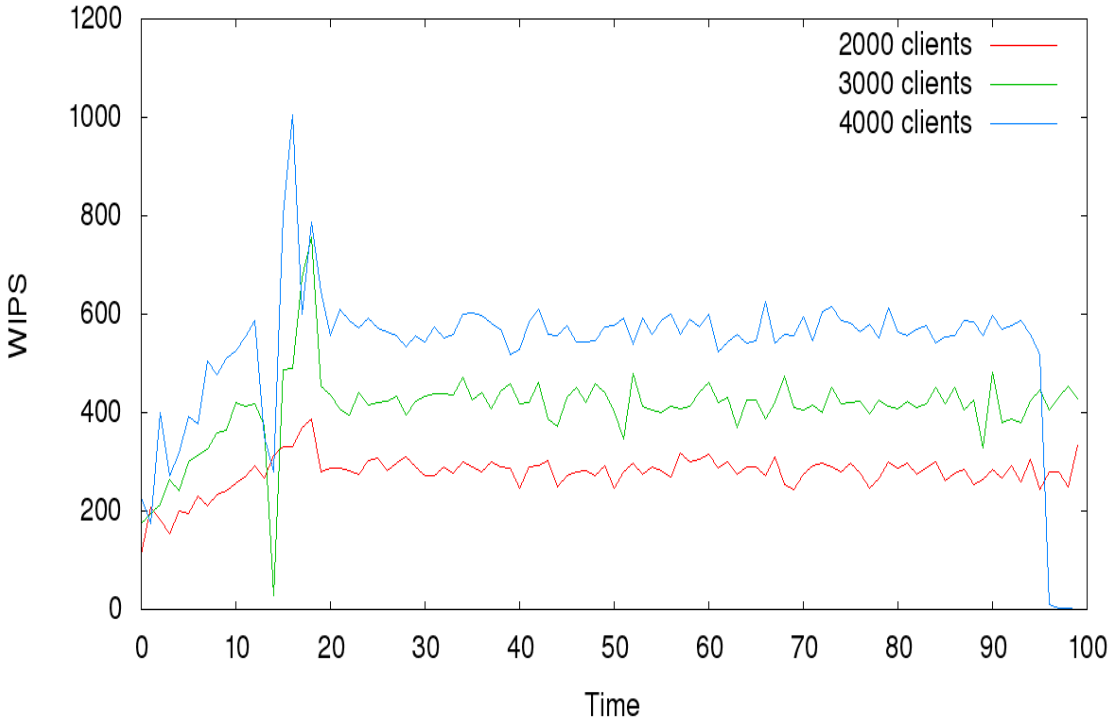

Figura 4: WIPSb por tempo(s) para cada carga simulada pelo RBE no perfil Browsing do experimento 2

O gráfico indica um melhor desempenho geral para 4000 clientes. Entretanto, no final do experimento há uma grande queda, que indica problemas devido a alta carga. Nesse caso, 3000 clientes apresenta melhor estabilidade em relação a 4000 e maior quantidade de WIPSb em relação a 2000 clientes

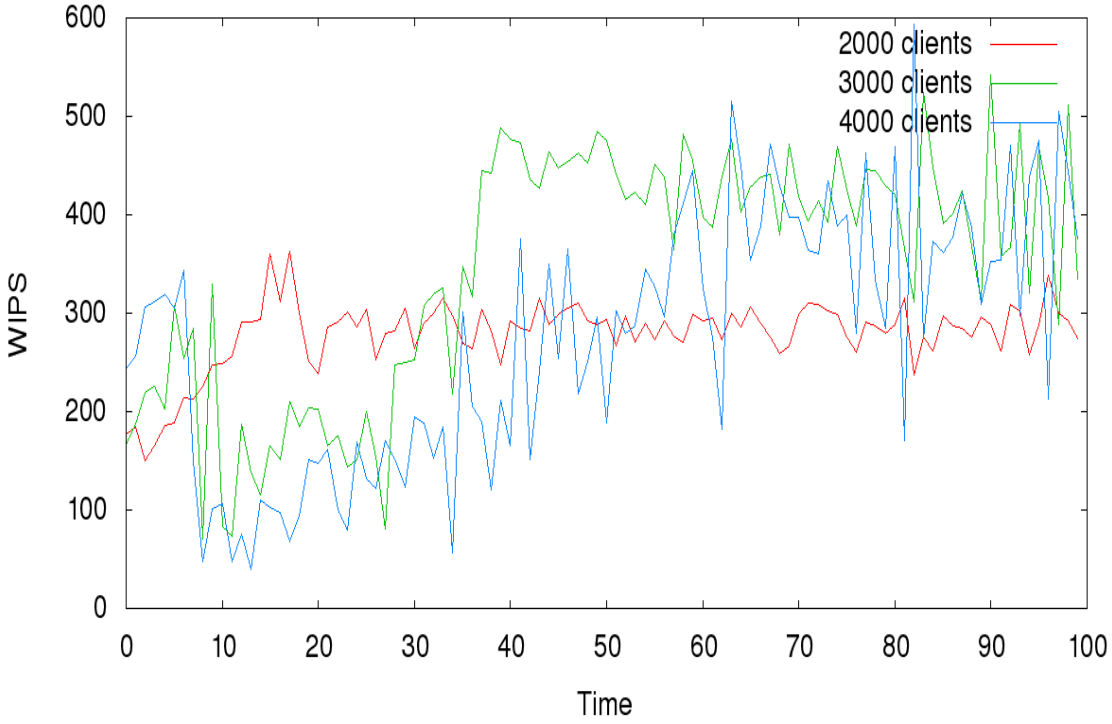

Figura 5: WIPSo por tempo(s) para cada carga simulada pelo RBE no perfil Ordering do experimento 2

O melhor desempenho geral foi para 3000 clientes, pois essa carga manteve uma boa média durante todo o experimento e se manteve no mesmo nível de WIPSo para 4000 clientes na segunda metade do experimento.

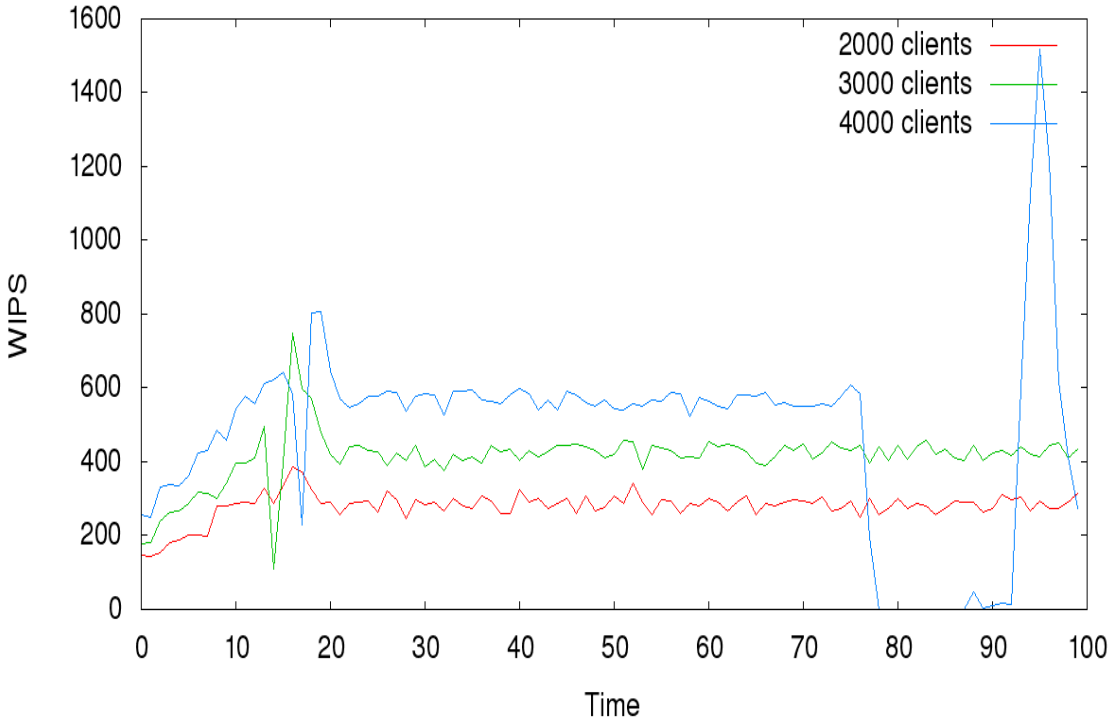

Figura 6: WIPS por tempo(s) para cada carga simulada pelo RBE no perfil Shopping do experimento 2

Este resultado se assemelha ao do perfil Browsing. 4000 clientes possui bom desempenho até aproximadamente 70s do experimento, quando sofre queda de WIPS, o que indica que está alta carga sobrecarrega o servidor. Como no perfil Browsing, a carga mais estável é de 3000 clientes.

Definimos então rodar o experimento três (com failover) utilizando uma carga de 3000 clientes.

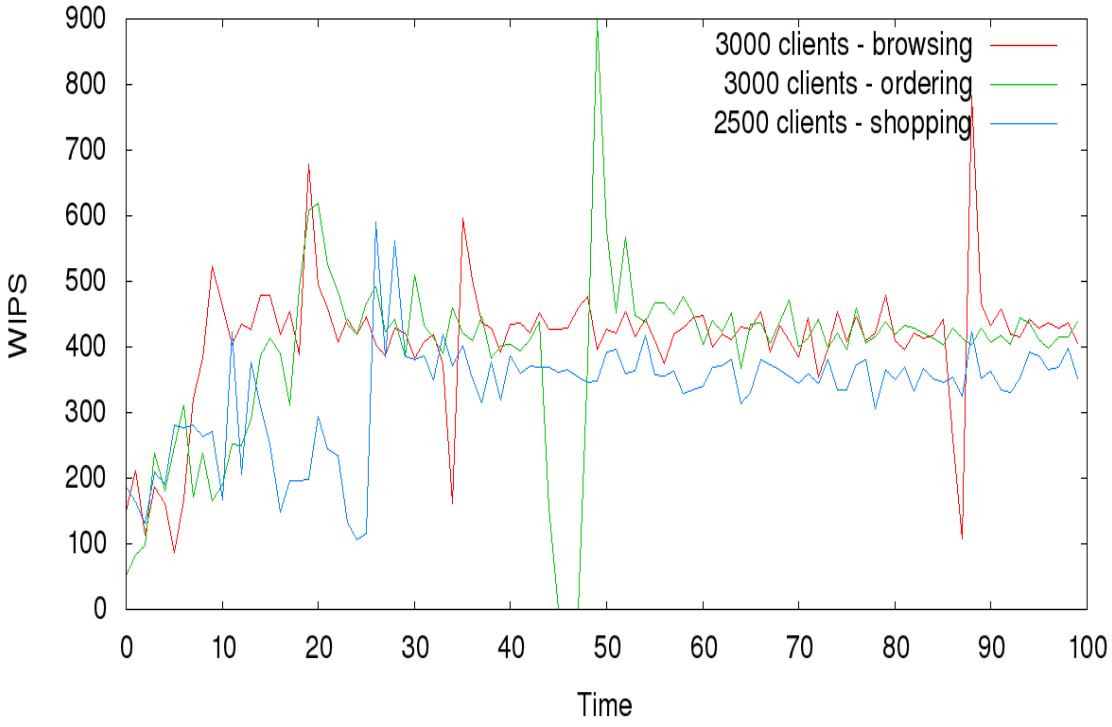

Figura 7: WIPS por tempo(s) para cada perfil simulado pelo RBE no experimento 3

Este gráfico indicas alguns resultados interessantes. Podemos ver que não houve queda de desempenho nos perfis de browsing e shopping, mesmo após a falha. Isso ocorreu provavelmente pois nosso script finaliza o processo do banco de dados primário (causando a falha) e logo em seguida já promove o secundário que rapidamente assume como primário, pois o HAProxy é rápido em notar que o primário falhou e já redireciona as requisições para o secundário. Porém, notamos que no perfil Ordering, onde tem muito mais instruções de escrita, que são muito custosas, a carga de 3000 causou falhas no servidor. Portanto, utilizamos uma carga de 2500 e conseguimos resultados satisfatórios. Vemos que a falha ocorre perto da marca de 45s e que o servidor se recupera por volta de 49s, o que equivale a um pouco menos de quatro segundos de recuperação.

Com esses dados, definimos uma carga ótima de 2000 clientes (com Think-Time = 1s), pois é um pouco abaixo dos 2500 clientes testados acima, deixando uma margem de segurança. A partir disso, começamos a variar o Think-Time e a realizarmos o experimento completo, alternando os bancos de dados múltiplas vezes entre primário e secundário. Utilizamos metade dos clientes, ou seja, 1000 clientes, pois iniciamos o Think-Time com metade do seu valor inicial.

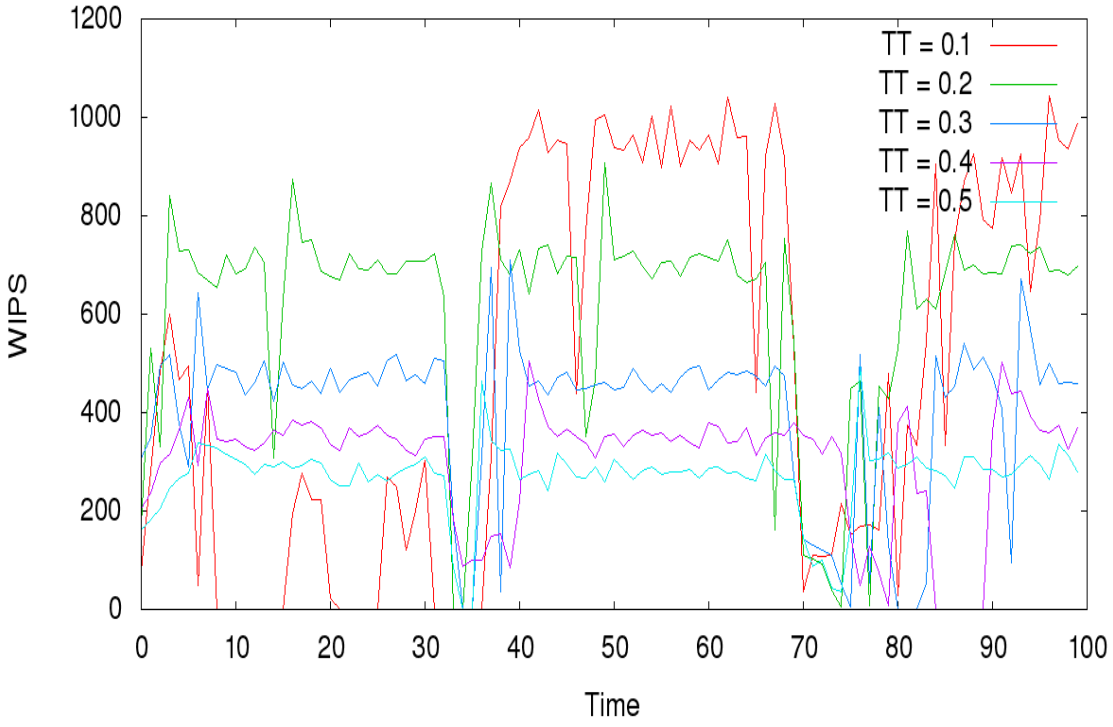

Figura 8: WIPSb por tempo(s) para cada Think-Time testado no perfil de Browsing no experimento completo

Vemos que para Think-Time muito baixos o WIPSb foi bem inconsistente, próximo de 0 nos primeiros 30s de experimento. Nota-se claramente os pontos onde as falhas nos primários foram inseridas pela queda de WIPSb para zero por um período. Os valores que mostraram melhor desempenho e estabilidade foram Think-Time de 0.2s e 0.3s.

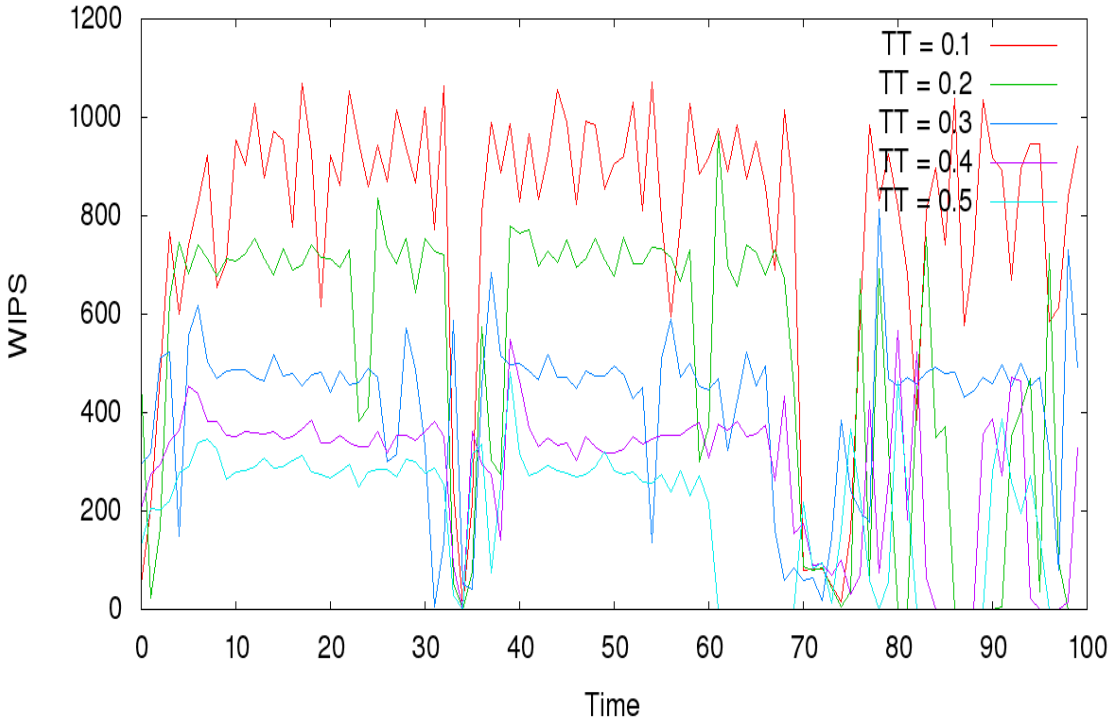

Figura 9: WIPS por tempo(s) para cada Think-Time testado no perfil de Shopping no experimento completo

O gráfico se manteve mais consistente, podendo ver a queda do banco de dados primário perto dos 30s e depois, novamente, perto dos 70s. Os melhores desempenhos foram obtidos pelos Think-times mais baixos.

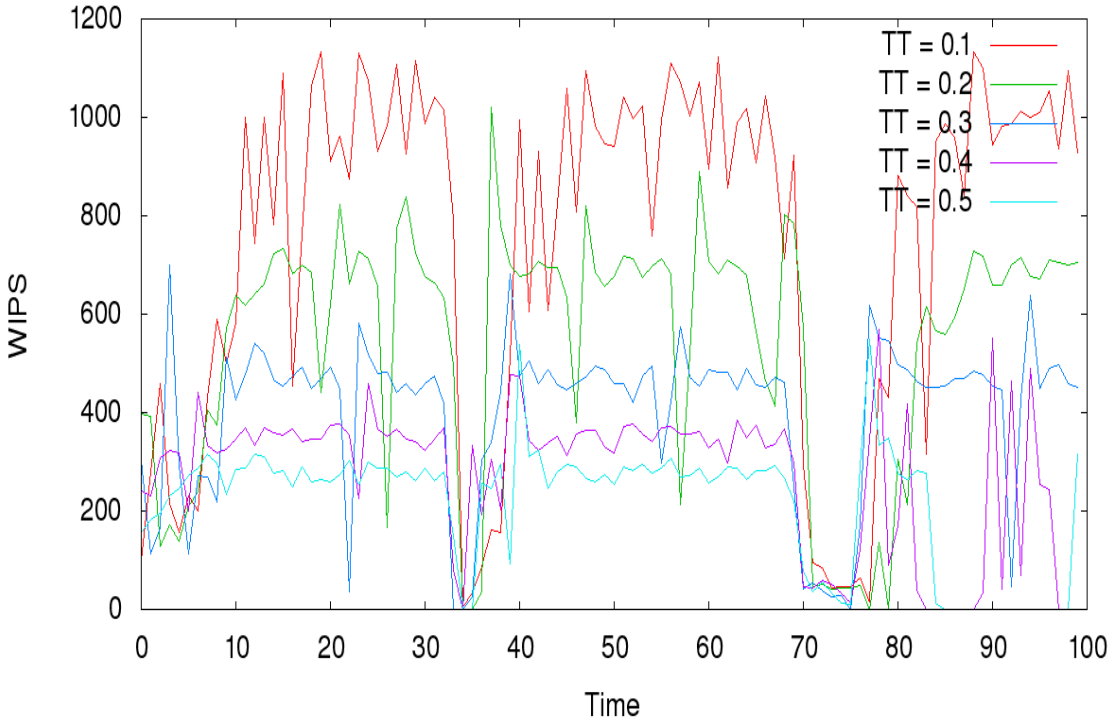

Figura 10: WIPSo por tempo(s) para cada Think-Time testado no perfil de Ordering no experimento completo

O gráfico também se manteve consistente, podendo ver a queda do banco de dados primário perto dos 30s e depois, novamente, perto dos 70s. Os melhores desempenhos foram obtidos pelos Think-Times mais baixos, novamente.

Analisando esses três gráficos, vemos que um valor estável e com bom desempenho foi o de Think-Time de 0.3s.

Para melhor ilustrar nossos resultados, fizemos gráficos com apenas os dados de Think-Time de 0.3s com 1000 clientes para os três perfis.

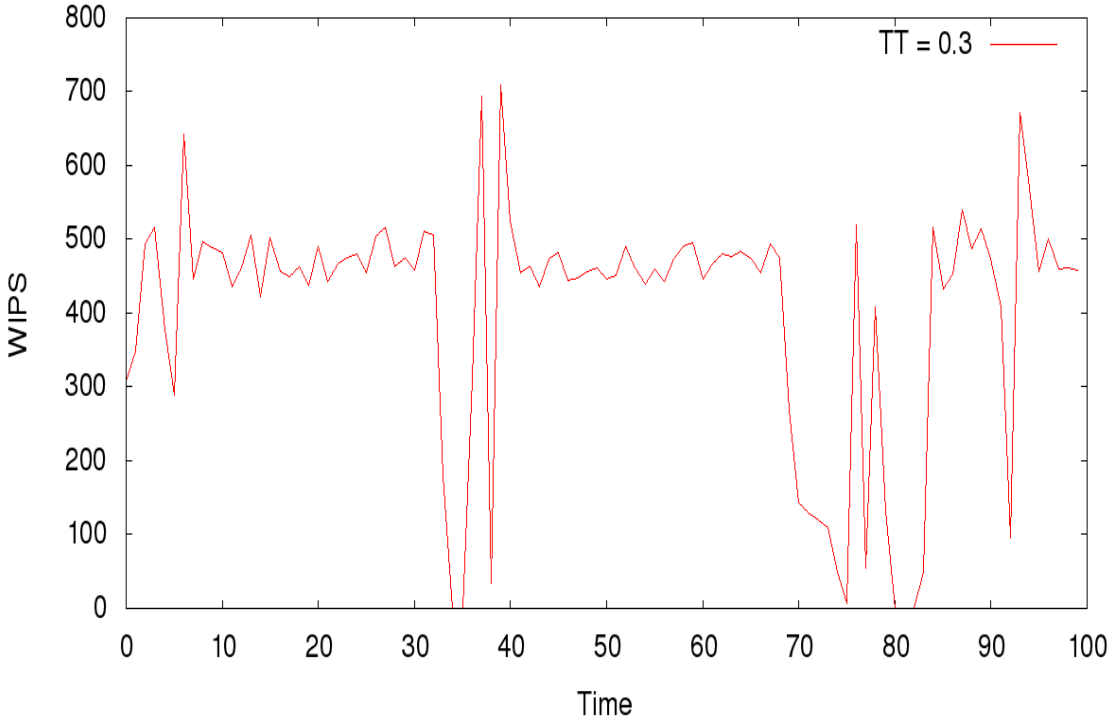

Figura 11: WIPSb por tempo(s) para Think-Time de 0.3s testado no perfil de Browsing no experimento completo

Pelo gráfico, vemos que após a primeira queda, por volta de 32s, o sistema voltou a ficar estável próximo aos 40s. A segunda queda ocorreu perto dos 68s e voltou à estabilidade perto dos 84s. Temos um tempo inoperante total de 24s e tempo de operação de 76s. Pela equação (1), nossa disponibilidade ficou por volta de 76%.

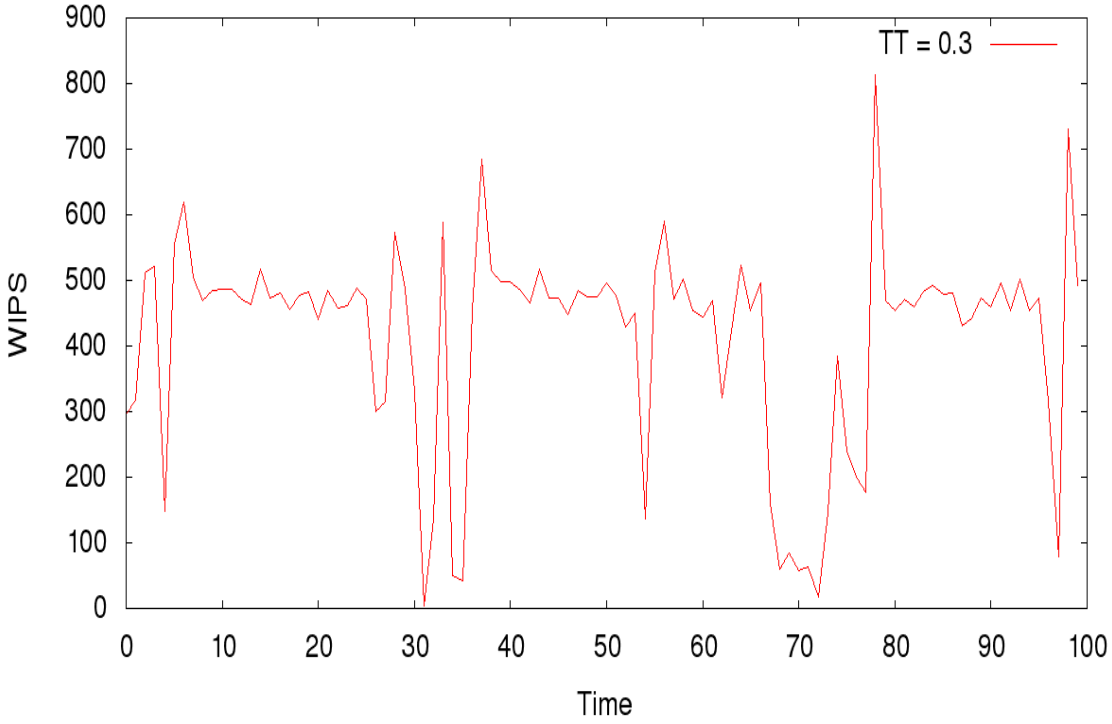

Figura 12: WIPS por tempo(s) para Think-Time de 0.3s testado no perfil de Shopping no experimento completo

Novamente, a primeira queda ocorreu por volta de 29s e se recuperou totalmente perto dos 37s. Já a segunda queda ocorreu próxima aos 65s e a recuperação ocorreu em 78s. O tempo inoperante ficou em torno de 21s e o tempo operante em 79s. Portanto, pela equação (1), a disponibilidade ficou em torno de 79%.

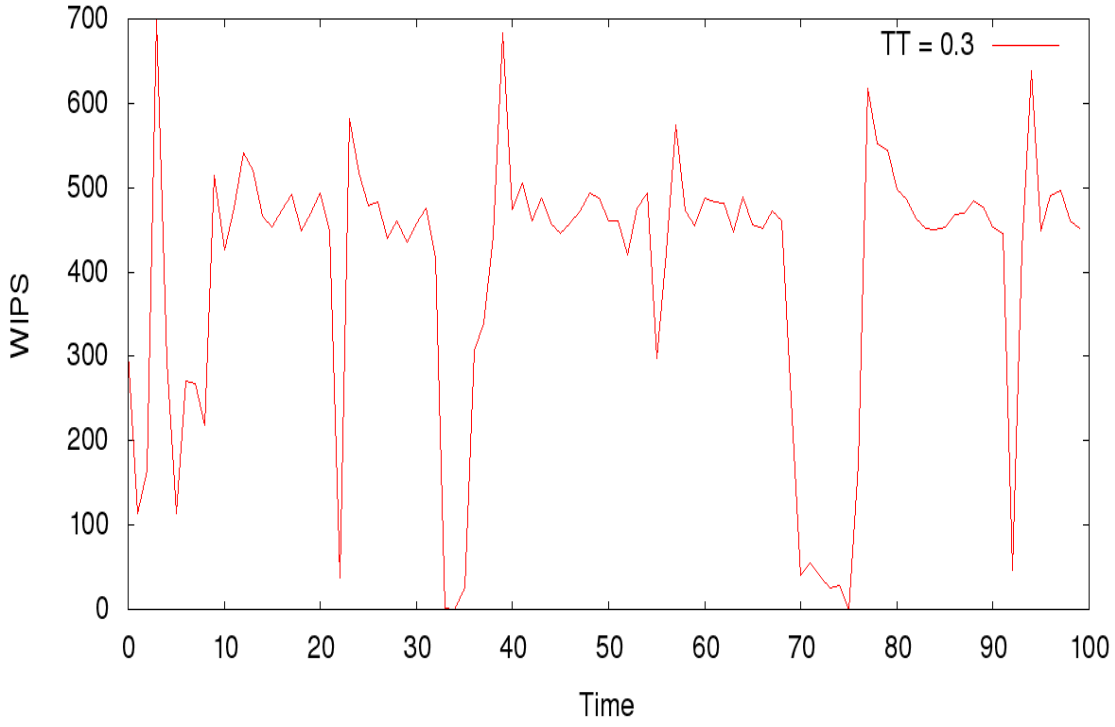

Figura 13: WIPSo por tempo(s) para Think-Time de 0.3s testado no perfil de Ordering no experimento completo

Por fim, a primeira falha ocorreu perto de 31s e a recuperação em 40s. A segunda falha ocorreu em torno de 69s e o sistema se recuperou em torno de 77s. Logo, o tempo inoperante ficou em torno de 17s e o tempo operante em 83s, o que nos da uma disponibilidade de 83%.

Segue na Tabela 5 as disponibilidades obtidas.

Perfil	Disponibilidade
WIPSb	76%
WIPS	79%
WIPSo	83%

Tabela 5: Disponibilidades do experimento final (1000 clientes, Think-Time = 0.3s)

5 Conclusão

Concluímos que os experimentos foram bem sucedidos.

Os experimentos um e dois, conseguiram prever com certa precisão a melhor carga a ser utilizada no experimento três, de cerca de 3000 clientes, que equivale aproximadamente ao uso de 1000 clientes com Think-Time de 0.3s.

O experimento três teve bastante sucesso nos perfis de browsing e shopping, onde a falha não fez cair o número de WIPS do servidor, pois a troca de banco de dados foi quase que imediata. Já no perfil de ordering, houve por volta de quatro segundos de tempo de

recuperação para o servidor voltar a responder requisições, o que consideramos um tempo bastante razoável.

Por fim, o experimento completo foi um sucesso. Nosso sistema conseguiu se recuperar adequadamente e apresentou uma disponibilidade boa, considerando que temos apenas um banco de dados em hot-standby e não tínhamos conhecimento sobre as ferramentas utilizadas ao iniciar o projeto, o que nos fez gastar muito tempo para aprendermos a utilizar as ferrementas e fazermos tudo funcionar, ao invés de gastarmos mais tempo em como melhorar a disponibilidade ou desempenho do sistema.

É claro que nenhuma empresa grande acharia disponibilidade perto de 80% um valor razoável, grandes sites como Google ou Amazon, tem disponibilidade de 99.9999...%, mas chegamos em bons valores dada as dificuldades apresentadas acima.

O desempenho do experimento completo com Think-Time de 0.3s foi em média por volta de 500 WIPS para os três perfis. Comparando com os valores de WIPS obtidos nos experimentos anteriores, vemos que 500 foi um ótimo valor. Quase nenhum gráfico do experimento 1 e 2 mostra valores melhores. Na figura 7, do experimento 3, vemos que a carga ótima ficou em torno de 500 WIPS também, mostrando que a carga de 1000 e Think-Time de 0.3s foi uma boa escolha para observarmos.

Referências

[1] "TPC benchmark W." `http://www.tpc.org/tpcw/`.

[2] "Streaming Replication." `https://wiki.postgresql.org/wiki/Streaming_Replication`.

[3] "Remote Browser Emulator." `http://www.cs.wm.edu/~esmirni/tpcw_codes/RBE.java`.

[4] "HAProxy." `http://www.haproxy.org/`.

[5] "Postgresql." `https://www.postgresql.org/`.