

UNIVERSIDADE DO MINHO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA
ADMINISTRAÇÃO DE DADOS

Benchmark TPC-C

João Linhares (a86618)
Joel Ferreira (a89982)
Rui Azevedo (a80789)

16 de janeiro de 2021

Conteúdo

1	Introdução	9
2	Configuração do <i>Benchmark TCP-C</i>	10
2.1	Automação de Testes e da Configuração Inicial	10
2.2	Ambiente de Teste	11
2.2.1	Número de <i>Warehouses</i>	12
2.2.2	Número de <i>Clientes</i>	13
3	Otimização do desempenho da carga transacional no <i>PostgreSQL</i>	17
3.1	Problemas encontrados	17
3.2	<i>Settings</i>	18
3.2.1	<i>wal_level</i>	18
3.2.2	<i>fsync</i>	19
3.2.3	<i>synchronous_commit</i>	20
3.2.4	<i>wal_sync_method</i>	22
3.2.5	<i>full_pages_writes</i>	24
3.2.6	<i>wal_buffers</i>	25
3.2.7	<i>wal_writer_delay</i>	26
3.2.8	<i>wal_writer_flush_after</i>	28
3.2.9	<i>commit_delay</i>	29
3.2.10	<i>commit_siblings</i>	31
3.3	<i>Checkpoints</i>	32
3.3.1	<i>checkpoint_timeout</i>	32
3.3.2	<i>max_wal_size</i>	34
3.3.3	<i>min_wal_size</i>	36
3.3.4	<i>checkpoint_completion_target</i>	37
3.3.5	<i>checkpoint_warning</i>	39
3.4	Archiving	40

3.4.1	<i>archive_mode</i>	40
3.5	Isolamento	42
3.6	Análise de resultados	43
3.6.1	<i>Settings</i>	43
3.6.2	<i>Checkpoints</i>	44
3.6.3	Configuração final	45
4	Optimização de Interrogações analíticas	48
4.1	Query Analítica 1	49
4.1.1	Índices	50
4.1.2	Vistas Materializadas	51
4.2	Query Analítica 2	52
4.2.1	Índices	57
4.2.2	Vistas Materializadas	60
4.3	Query Analítica 3	65
4.3.1	Índices	67
4.3.2	Vistas Materializadas	69
5	Conclusão	73
6	Anexos	75

Listas de Figuras

1	Gráfico que representa a evolução do tamanho da BD consoante o número de <i>warehouses</i>	13
2	Gráfico que representa a evolução do <i>response time (s)</i> consoante o número de clientes	14
3	Gráfico que representa a evolução do <i>throughput (tx/s)</i> consoante o número de clientes	15
4	Utilização do CPU com 100 clientes obtida através da ferramenta de monitoramento da <i>Google Cloud</i>	15
5	Métricas obtidas com 80 <i>warehouses</i> e 100 clientes	16
6	Comparação <i>wal_level</i>	19
7	Métricas com a opção <i>fsync = off</i>	20
8	Comparação <i>synchronous_commit</i>	21
9	Métricas com a opção <i>synchronous_commit = off</i>	22
10	Comparação <i>wal_sync_method</i>	23
11	Métricas com a opção <i>wal_sync_method = fsync</i>	23
12	Métricas com a opção <i>full_page_writes = off</i>	24
13	Comparação <i>wal_buffers</i>	25
14	Métricas com a opção <i>wal_buffers = 32MB</i>	26
15	Comparação <i>wal_writer_delay</i>	27
16	Métricas com a opção <i>wal_writer_delay = 2000ms</i>	27
17	Comparação <i>wal_writer_flush_after</i>	28
18	Métricas com a opção <i>wal_writer_flush_after = 32MB</i>	29
19	Comparação <i>commit_delay</i>	30
20	Métricas com a opção <i>commit_delay = 10ms</i>	30
21	Comparação <i>commit_siblings</i>	31
22	Métricas com a opção <i>commit_siblings = 16</i>	32
23	Comparação <i>checkpoint_timeout</i>	33
24	Métricas com a opção <i>checkpoint_timeout = 15</i>	34

25	Comparação <i>max_wal_size</i>	35
26	Métricas com a opção <i>max_wal_size = 5GB</i>	35
27	Comparação <i>min_wal_size</i>	36
28	Métricas com a opção <i>min_wal_size = 160MB</i>	37
29	Comparação <i>checkpoint_completion_target</i>	38
30	Métricas com a opção <i>checkpoint_completion_target = 0.2</i>	38
31	Comparação <i>checkpoint_warning</i>	39
32	Métricas com a opção <i>checkpoint_warning = 120s</i>	40
33	Comparação <i>archive_mode</i>	41
34	Métricas com a opção <i>archive_mode = off</i>	41
35	Métricas com as melhores configurações das <i>settings</i>	44
36	Métricas com as melhores configurações dos <i>checkpoints</i>	45
37	Métricas finais.	47
38	Lista de índices	48
39	Plano e Tempo de execução da Query Analítica 1	49
40	Plano e Tempo de execução da Query Analítica 1 com <i>seqcan = OFF</i>	50
41	Plano e Tempo de execução da Query Analítica 1 usando o índice <i>i1</i>	51
42	Plano e Tempo de execução da Query Analítica 1 com Vista Materializada	52
43	Plano e Tempo de execução da Query Analítica 2 - Primeira execução.	53
44	Plano e Tempo de execução da Query Analítica 2 - Segunda execução.	54
45	Plano e Tempo de execução da Query Analítica 2 - Sequential Scan Off.	55
46	Estatísticas por tipo de nodo.	56
47	Estatísticas por tabela.	56
48	Plano da Query Analítica 2 com índices.	58
49	Tempo de execução da Query Analítica 2 com índices.	59
50	Estatísticas por nodo da execução da Query 2 com índices.	59
51	Estatísticas por tabela da execução da Query 2 com índices.	60
52	Plano da Query Analítica 2 com índices e vista materializada.	62
53	Tempo de execução da Query Analítica 2 com índices e vista materializada.	62

54	Estatísticas por nodo da execução da Query 2 com índices e vista materializada.	63
55	Estatísticas por tabela da execução da Query 2 com índices e vista materializada.	63
56	Plano da Query Analítica 2 com índices, vista materializada e com sequential scan a on.	64
57	Tempo de execução da Query Analítica 2 com índices, vista materializada e com sequential scan a on.	64
58	Estatísticas da execução da Query 2 com índices, vista materializada e com sequential scan a on.	65
59	Tempo de execução da Query Analítica 3 sem qualquer alteração.	66
60	Plano e Tempo de execução da Query Analítica 3 com novos parâmetros.	67
61	Tabela sobre os Scans do plano de execução obtido através do PgAdmin.	68
62	Plano e Tempo de execução da Query Analítica 3 com novos índices.	68
63	Plano e Tempo de execução da Query Analítica 3 com SeqScan a off com índices.	69
64	Região crítica do tempo de execução da <i>query</i> 3.	70
65	Tempo e plano de execução da query 3 após criação da vista materializada.	71
66	Tempo e plano de execução da query 3 após criação da vista materializada com índices.	72

Listas de Tabelas

1	Características das Máquinas do Sistema	11
2	Warehouses x Tamanho da Base de Dados	12
3	Resultados obtidos ao correr determinado número de clientes	14
4	Diferenças entre os dois métodos de execução na configuração <i>wal_sync_method</i>	18
5	Resultados obtidos com a alteração da configuração <i>wal_level</i>	18
6	Resultados obtidos com a alteração da configuração <i>fsync</i>	19
7	Resultados obtidos com a alteração da configuração <i>synchronous_commit</i>	21
8	Resultados obtidos com a alteração da configuração <i>wal_sync_method</i>	22
9	Resultados obtidos com a alteração da configuração <i>full_pages_writes</i>	24
10	Resultados obtidos com a alteração da configuração <i>wal_buffers</i>	25
11	Resultados obtidos com a alteração da configuração <i>wal_writer_delay</i>	26
12	Resultados obtidos com a alteração da configuração <i>wal_writer_flush_after</i>	28
13	Resultados obtidos com a alteração da configuração <i>commit_delay</i>	29
14	Resultados obtidos com a alteração da configuração <i>commit_siblings</i>	31
15	Resultados obtidos com a alteração da configuração <i>checkpoint_timeout</i>	33
16	Resultados obtidos com a alteração da configuração <i>max_wal_size</i>	34
17	Resultados obtidos com a alteração da configuração <i>min_wal_size</i>	36
18	Resultados obtidos com a alteração da configuração <i>checkpoint_completion_target</i>	37
19	Resultados obtidos com a alteração da configuração <i>checkpoint_warning</i>	39
20	Resultados obtidos com a alteração da configuração <i>archive_mode</i>	40
21	Resultados obtidos com os diferentes níveis de isolamento	42

Listings

1	Query Analítica 1	49
2	Vista Materializada 1	51
3	Query Analítica 2	52
4	Índices criados	57
5	Vista Materializada	61
6	Interrogação Analítica com Vista Materializada	61
7	Query Analítica 3	65
8	Query Analítica 3 alterada	66
9	Criação de índices	67
10	Criação de índice	69
11	Vista Materializada	70
12	Nova estrutura da query	70
13	Script de Configuração do Database Server	75
14	Script de Configuração do Servidor de Benchmark	77
15	Script Limpeza Base de Dados	79
16	Script Drop Base de Dados e Criação de Nova Dado um Número de Warehouses	81
17	Script que Corre o Script Transacional Dado um Número de Clientes e Warehouses	82
18	Script Restore da Base de Dados Dado um Ficheiro de Backup, Número de Warehouses e Clientes	84
19	Script Criação e Povoamento da Base de Dados com Ficheiro de Backup ou Correndo o Script Transacional	86
20	Script Configuração de Discos Locais da Google	87
21	Script Execução Teste Para Clientes Criando Máquinas Virtuais Novas	88
22	Script Execução Teste Para Pârametros do PostgreSQL Criando Máquinas Virtuais Novas	90
23	Script Execução Testes das Combinações Criando Máquinas Virtuais Novas	92
24	Script Testes Automáticos Para Clientes	94
25	Script Testes Automáticos Para <i>Settings</i>	95

26	Script Testes Automáticos Para <i>Checkpoints</i>	100
27	Script Testes Automáticos Para <i>Archiving</i>	103
28	Script Testes Automáticos Para Combinações	104
29	Ficheiro Com Exemplos de Utilização dos Scripts	106

1 Introdução

O presente relatório é referente ao trabalho prático desenvolvido na cadeira de Administração de Base de Dados do perfil de Engenharia de Aplicações.

É pretendido com este trabalho fazer a configuração, optimização e avaliação do *benchmark* TPC-C, um sistema de base de dados *online* que simula um *benchmark* transacional (OLTP). Para além disso, é pretendido também adaptar e optimizar um conjunto de interrogações analíticas, baseadas no TPC-H, principalmente os respectivos planos e mecanismos de redundância.

Numa primeira fase, irá ser instalado o *benchmark* na plataforma *Google Cloud Platform* e, para além disso, irá ser apresentada uma configuração de referência, tendo em conta o *hardware*, o número de *warehouses* e o número de clientes a usar a aplicação.

De seguida, irão ser apresentadas as diferentes configurações feitas ao sistema de base de dados que suporta o *benchmark*, neste caso o *Postgresql*. Esta fase tem como objectivo obter um aumento no desempenho da carga transacional do sistema.

Por fim, e como já foi referido anteriormente, irão ser analisadas um conjunto de interrogações analíticas baseadas no TPC-H com o objectivo de melhorar os respectivos planos e mecanismos de redundância.

2 Configuração do *Benchmark* TCP-C

Esta secção tem como objectivo apresentar alguns detalhes sobre a instalação do *benchmark*, tendo como objectivo final expôr uma configuração base do sistema.

2.1 Automação de Testes e da Configuração Inicial

No início do trabalho prático, apercebeu-se que todas as tarefas de teste eram bastante repetitivas pelo que se decidiu automatizar ao máximo este processo para que a intervenção manual necessária fosse mínima.

A solução passou pela criação de vários *shell scripts* responsáveis pela automatização de todo o processo de teste e configuração inicial das máquinas virtuais (servidor de base de dados e *benchmark*), bem como a configuração do servidor de base de dados após um *reboot*, situação na qual todos os dados são perdidos devido ao uso de discos de trabalho *SSD* locais.

Para a configuração inicial e *reboot* da máquina existem dois *scripts*, nomeadamente, *dbserverconfiguration.sh* e *benchmarkconfiguration.sh*. O primeiro é responsável por essencialmente configurar o servidor de base de dados, ou seja, instalar o sistema de base de dados *PostgreSQL*, montar o disco de trabalho *SSD* e montar a pasta de dados do *PostgreSQL* no mesmo, alterar os ficheiros *postgresql.conf* e *pg_hba.conf* para que a máquina de *benchmark* se consiga ligar remotamente, e por fim, inicializar o servidor, deixando-o a correr em *background*. Este mesmo *script*, é usado para recuperar de um *reboot*, uma vez que usando discos locais da Google após parar o servidor é apagado todo o conteúdo desse disco, sendo depois necessário no *reboot* configurar os discos locais e instalar a base de dados novamente. O segundo *script* é usado para configurar o servidor de *benchmark*, ou seja instalar o *Java*, o *python* e o *postgresql-client-12*, descompactar os executáveis que irão correr a aplicação, e, por sua vez, correr o *script* *createdb.sh*, responsável por criar a base de dados *tpcc*, criar todas as tabelas e, por fim, executar o *script* transacional.

Na fase de decisão do ambiente de teste, foi necessário decidir qual o número de clientes que se iriam usar para correr a aplicação. O *script* responsável por este teste, *autotest_clients.sh*, calcula as métricas para valores de clientes entre 10 e 120, em intervalos de 10 clientes. Quando o *script* acaba de correr, é possível ter acesso a todos os valores obtidos sendo só necessário fazer a análise dos mesmos e a decisão de quantos usar. Este *script* invoca outro *script* auxiliar (*autorun_clients.sh*) que procede ao *restore* da base de dados, cria novas máquinas virtuais para cada teste e altera os ficheiros *database.properties* do *benchmark tpcc*.

Para a fase de testes das configurações do sistema de base de dados, foi necessário criar uma máquina nova por cada configuração alterada, motivo pelo qual será explicado em capítulos posteriores. Dado isto, e para optimizar ao máximo o tempo, existem *scripts* para correr todas as configurações escolhidas para análise de uma forma automática. Estes ficheiros

estão divididos pelo domínio da configuração (*settings*, *checkpoints* e *archiving*) e têm como comportamento, por iteração, a criação do servidor de base de dados, o *restore* da mesma, a execução das transações durante o período de 10 minutos, e por fim, guardar o ficheiro de dados, com o nome da configuração executada, na pasta respetiva ao seu domínio, bem como executar o *showtpc.py* e guardar o *output* num ficheiro *txt*.

Na fase de escolha das melhores configurações, o *autorun_optimizations.sh* trata de correr o *benchmark* com as combinações escolhidas. Para isto, e como se irá ver posteriormente, foram escolhidas as melhores configurações de cada domínio, e o processo acaba por ser praticamente igual ao explicado no parágrafo anterior, mas neste caso, são executadas três vezes consecutivas as configurações de cada domínio e uma configuração que engloba uma combinação dos dois.

Para facilitar a percepção do que cada *script* faz, existe um ficheiro de ajuda nos anexos onde se encontram alguns exemplos de utilização de cada script.

Apesar de se ter perdido algum tempo na criação de testes automáticos, pode-se constatar que este processo fez com que fosse possível analisar uma maior quantidade de configurações e, ao mesmo tempo, o grupo focar-se na essência do trabalho e não estar a lidar com o processo repetitivo de criação do ambiente de trabalho.

2.2 Ambiente de Teste

O primeiro passo na criação do ambiente de teste foi definir quais as configurações das máquinas que dão suporte ao sistema e, por sua vez, qual sistema operativo a usar. A arquitetura do *benchmark* conta com uma máquina virtual que hospeda o servidor de base de dados e uma outra que faz pedidos ao mesmo. As características de ambas as máquinas podem-se observar na tabela que se segue.

Tabela 1: Características das Máquinas do Sistema

	Server	Bench
Zona	us-central1-a	us-central1-a
Tipo de Máquina	n1-standard-2	f1-micro
CPU	2vCPUs 7.5GB	1 vCPU 0.6GB
Disco	Disco Permanente SSD 10GB	Disco Permanente Padrão 20GB
Disco Extra	Disco de Trabalho SSD Local (NVMe) 375GB	-
SO	Ubuntu 20.04 LTS	Ubuntu 20.04 LTS

No caso da máquina que hospeda a base de dados, foi criado um disco adicional *SSD* com capacidade de 375GB, tanto para se ter mais capacidade de armazenamento, mas também para tornar mais eficiente as leituras e escritas no mesmo. Para isto, foi necessário montar um sistema de ficheiros sobre esse disco e armazenar a directória que dá suporte ao *PostgreSQL* nesse mesmo disco.

2.2.1 Número de *Warehouses*

Uma vez escolhidas as máquinas que irão dar suporte ao sistema, é necessário definir o número de *warehouses* que irão ser usadas durante a análise de desempenho da base de dados. O número de *warehouses* que se pretende obter é um número que, após o povoamento da base de dados, garanta que a mesma tenha um tamanho superior ao da memória da máquina.

Para isto, foram realizados testes com um número variável de *warehouses* e, em cada iteração, foi verificado o tamanho da base de dados com o seguinte comando:

```
SELECT pg_size_pretty( pg_database_size('tpcc') );
```

A Tabela 2 apresenta as dimensões da base de dados obtidas dado o número de *warehouses*. Como se pode observar, quando o número de *warehouses* é igual ou superior a 64, o tamanho da base de dados é superior a 7.5GB, correspondente ao valor da memória da máquina que hospeda a base de dados. Dado isto, decidiu-se que 80 *warehouses* era um número adequado para o ambiente de teste pois já ultrapassa a memória em cerca de 2GB.

Tabela 2: Warehouses x Tamanho da Base de Dados

Warehouses	Tamanho da Base de Dados (MB)
2	257
4	490
8	953
16	1879
32	3778
64	7503
80	9386
100	11000

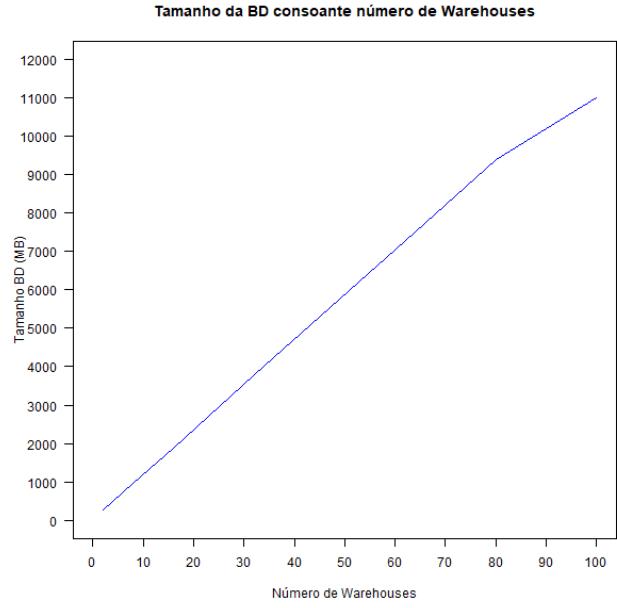


Figura 1: Gráfico que representa a evolução do tamanho da BD consoante o número de *warehouses*

2.2.2 Número de *Clients*

Tendo como referência as 80 *warehouses*, o passo seguinte foi escolher o número de clientes que iria ser usado. Para isto, usou-se um *script* que corria o *benchmark* para vários clientes e que, ao fim de cada iteração, guardava os valores obtidos para uma posterior análise.

Tabela 3: Resultados obtidos ao correr determinado número de clientes

Num Clients	Throughput (tx/s)	Response Time (s)	Abort Rate (%)	CPU Usage (%)
10	69.6038	0.0132	0.2263	45.67%
20	138.6637	0.0189	0.7399	48.29%
30	208.1995	0.0210	1.2444	51.98%
40	277.1346	0.0229	1.7936	54.84%
50	345.5885	0.0162	1.5431	57.04%
60	409.9729	0.0233	2.6108	58.66%
70	472.8912	0.0254	3.1288	60.85%
80	537.3871	0.02419	3.2725	62.11%
90	577.0042	0.0305	4.0995	63.89%
100	615.4372	0.0306	4.2883	66.84%
110	610.1497	0.0311	4.3657	69.03%
120	544.3316	0.0349	4.3648	73.64%

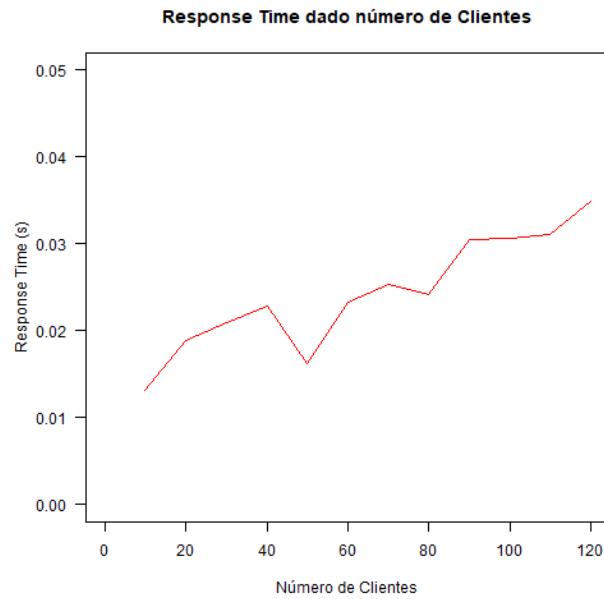


Figura 2: Gráfico que representa a evolução do *response time (s)* consoante o número de clientes

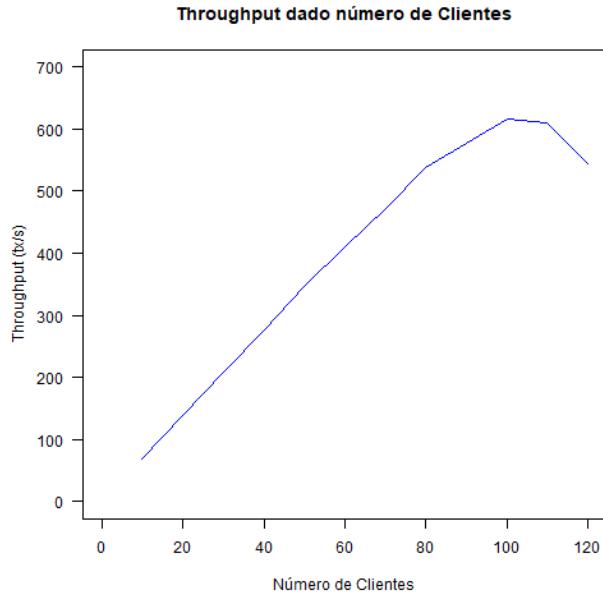


Figura 3: Gráfico que representa a evolução do *throughput* (tx/s) consoante o número de clientes

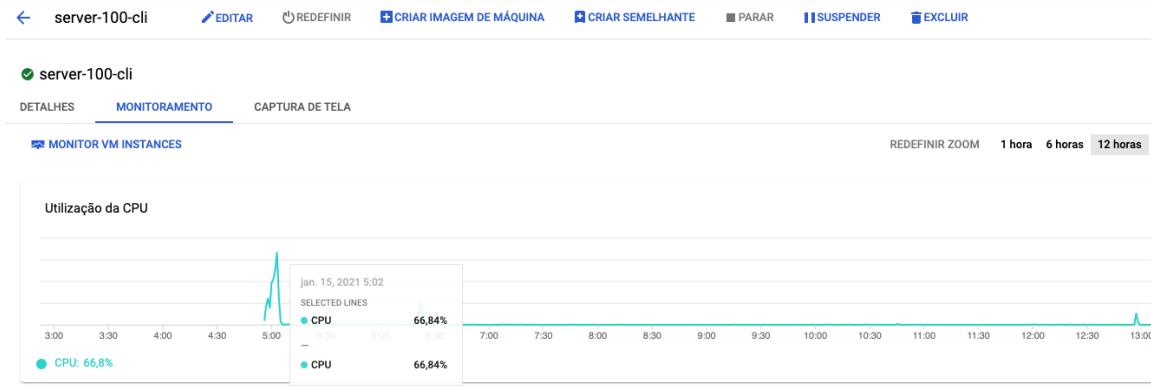


Figura 4: Utilização do CPU com 100 clientes obtida através da ferramenta de monitoramento da *Google Cloud*.

Após correr o *script*, obteve-se a Tabela 3 que apresenta os resultados obtidos para os vários números de clientes. Para obter a utilização do CPU, foi utilizada a ferramenta de monitorização do *Google Cloud* ao que foram lá retirados os valores para cada cliente durante as execuções dos testes.

De seguida, para escolher o valor óptimo, foi escolhido o número de clientes que garanta

os melhores valores de *throughput* e uma utilização do CPU superior a 50%, para exercer uma boa carga computacional à máquina e inferior a 90%, para não haver muita dificuldade nas realizações das tarefas. Portanto, foi necessário procurar na Tabela 3 e no Gráfico 3 para encontrar a configuração em que o seu *throughput* seja o valor mais alto possível e que a *CPU Usage* seja superior a 50% e inferior a 90%. Assim determinou-se que o melhor valor seria 100 clientes que tanto tem os maiores valores de *throughput* como a sua utilização de CPU é ideal.

De maneira a criar um ambiente de teste uniforme e sistemático, decidiu-se que para cada configuração alterada iria ser criada uma nova instância do servidor de base de dados. Esta escolha é melhor justificada na secção 3.1.

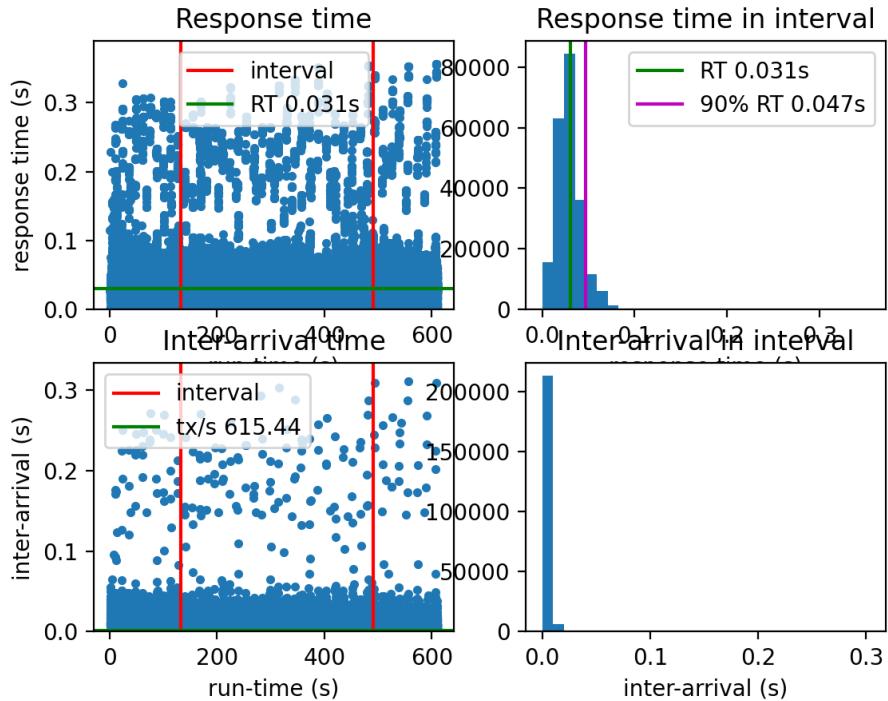


Figura 5: Métricas obtidas com 80 *warehouses* e 100 clientes

3 Otimização do desempenho da carga transacional no *PostgreSQL*

Tendo como base a configuração inicial do ambiente teste, nesta secção serão analisadas as configurações do sistema de base de dados *PostgreSQL*. Estas configurações encontram-se no ficheiro *postgresql.conf* e dizem respeito a vários domínios de configuração. No entanto, esta secção foca-se nas configurações que têm um impacto directo sobre as transações executadas no sistema, nomeadamente, as configurações que pertencem domínios *settings*, *checkpoints* e *archiving*.

Como já foi referido anteriormente, para esta análise foram criados *scripts* para automatizar o processo de análise. Cada *script* é responsável por modificar as configurações de um determinado domínio para que, deste modo, se possam correr as várias configurações em paralelo, para uma obtenção de resultados mais rápida.

3.1 Problemas encontrados

Durante a execução de alguns testes, reparou-se que os dados que se estavam a obter estavam bastante incoerentes. Consequentemente, decidiu-se correr várias vezes consecutivas o *script* *run.sh*, sem se alterar quaisquer configuração da base de dados, com o objectivo de verificar as variações do *throughput* dado o mesmo ambiente de teste. Observou-se então que os valores obtidos variavam bastante, sendo que as primeiras *runs* tinham valores muito altos de *throughput* e que, posteriormente, os valores diminuíam para cerca de metade. É de notar que, nestes testes feitos, o ficheiro *data* do *PostgreSQL* era sempre eliminado e criado um novo para correr o próximo teste. Dado isto, e não tendo encontrado nenhum motivo aparente para o sucedido, decidiu-se que para cada configuração alterada seria criada uma instância nova do servidor de base de dados para que todos os testes fossem feitos sobre as mesmas condições. Procedeu-se então à criação de *scripts* responsáveis pelo seguinte comportamento iterativo:

```
create(ServerDB) → mount(SSD) → create(dataFile) → alter(config) → run()
```

Ao correr os testes com a esta lógica, verificou-se que os resultados obtidos estavam coerentes e, portanto, todos os resultados apresentados nas seções posteriores têm como base esta lógica de execução.

Tabela 4: Diferenças entre os dois métodos de execução na configuração *wal_sync_method*

Método	Opção	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
Na Mesma VM	fsync	560.4566	0.0430	4.2439
	open_datasync	544.1888	0.0350	4.2291
	open_sync	528.8009	0.0360	4.2526
Nova VM por Teste	fsync	609.4365	0.0311	4.2442
	open_datasync	580.2892	0.0328	4.1929
	open_sync	564.9263	0.0332	4.2270

3.2 Settings

3.2.1 wal_level

Este parâmetro determina a quantidade de informação que é escrita no *Write Ahead Log*. Por definição, este parâmetro está com o valor *minimum*, que significa que só a informação que é necessária para recuperação é efectivamente escrita. Definindo este parâmetro como *logical*, alguma informação adicional sobre descodificação lógica é adicionada. No caso em que o valor do *wal_level* é *minimal*, foi necessário alterar uma configuração extra, nomeadamente, *max_wal_senders* que tem que ter o valor de 0.

Tabela 5: Resultados obtidos com a alteração da configuração *wal_level*

Opção	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
logical	521.1982	0.0366	4.3457
minimal	502.5283	0.0381	4.4666

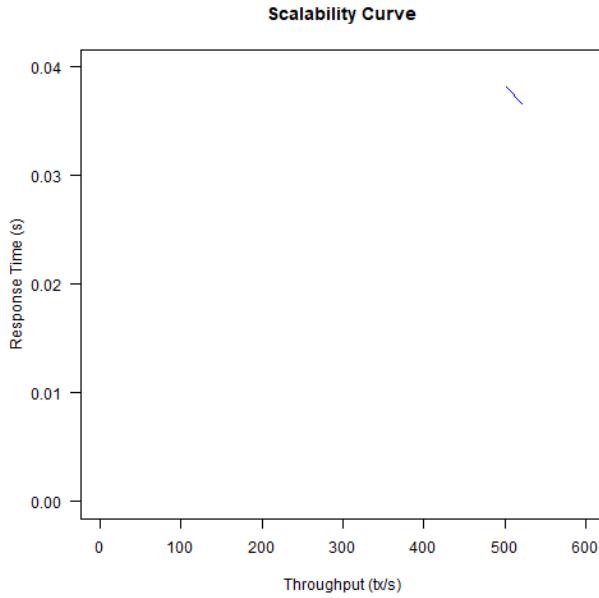


Figura 6: Comparaçāo *wal_level*

3.2.2 *fsync*

Se este parāmetro estiver com o valor *On*, o servidor *PostgreSQL* vai tentar ter a certeza que os updates feitos à base de dados estão fisicamente escritos no disco através de chamadas ao sistema. Este processo garante que o *cluster* da base de dados consegue-se recuperar para um estado consistente caso haja uma falha no sistema.

Caso o parāmetro esteja a *Off* escreve directamente no disco sem tomar estas precauções. Apesar de normalmente melhorar o nível de performance pode resultar em corrupção de dados irrecuperável.

Tabela 6: Resultados obtidos com a alteração da configuração *fsync*

Opção	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
off	694.5720	0.0088	1.1219

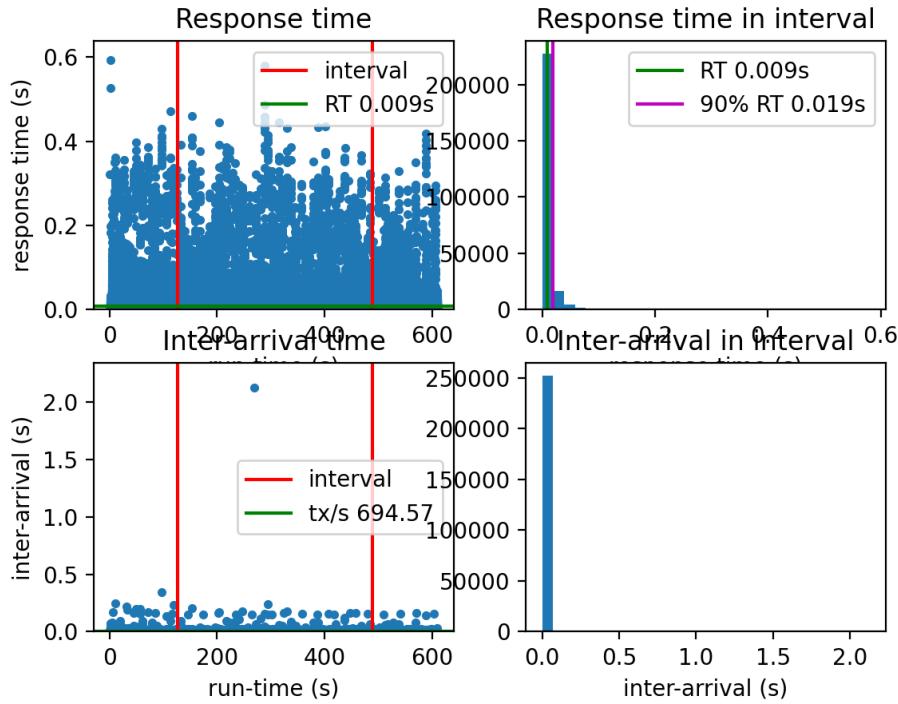


Figura 7: Métricas com a opção $fsync = off$

3.2.3 *synchronous_commit*

Esta configuração especifica quanto processamento *WAL* deve ser concluído antes que o servidor de base de dados retorne uma indicação de "sucesso" ao cliente.

Se estiver no modo *Off* não há espera, portanto, pode haver um atraso entre o momento em que o sucesso é relatado ao cliente e quando a transacção é garantida posteriormente como segura contra uma falha no servidor.

Ao contrário do *fsync*, definir este parâmetro como *Off* não cria nenhum risco de inconsistência da base de dados, uma falha no sistema ou na BD pode resultar na perda de algumas transacções recentes supostamente confirmadas, mas o estado da base de dados irá ser exactamente a mesma como se essas transacções tivessem sido abortadas de forma limpa.

Tabela 7: Resultados obtidos com a alteração da configuração *synchronous_commit*

Opção	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
off	653.8542	0.0168	1.9009
remote_write	637.5204	0.0272	4.0817
local	634.2497	0.0279	4.0467
remote_apply	580.9083	0.0327	4.3064

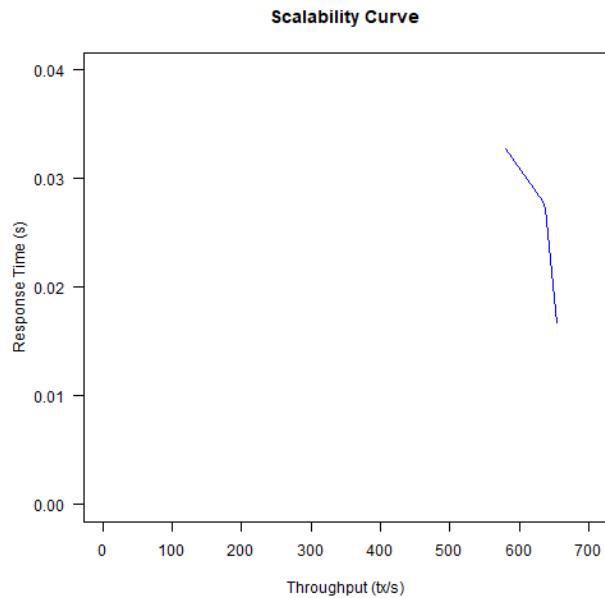


Figura 8: Comparaçāo *synchronous_commit*

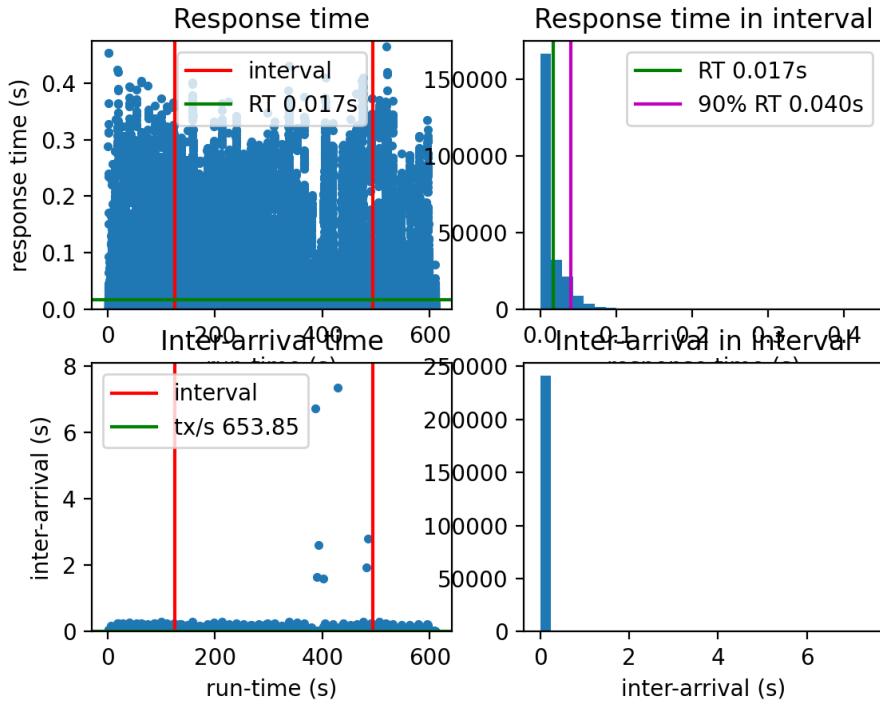


Figura 9: Métricas com a opção *synchronous_commit* = off

3.2.4 wal_sync_method

Nesta configuração é possível determinar o método utilizado para forçar updates *WAL* para o disco. Embora cada sistema operativo já tenha por defeito o melhor método para notificar quando as escritas estão concluídas, decidimos executar este teste apenas a título de curiosidade. Na tabela é baixo é possível visualizar os resultados para os métodos disponíveis.

Tabela 8: Resultados obtidos com a alteração da configuração *wal_sync_method*

Opção	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
fsync	609.4365	0.0311	4.2442
open_datasync	580.2892	0.0328	4.1929
open_sync	564.9263	0.0332	4.2270

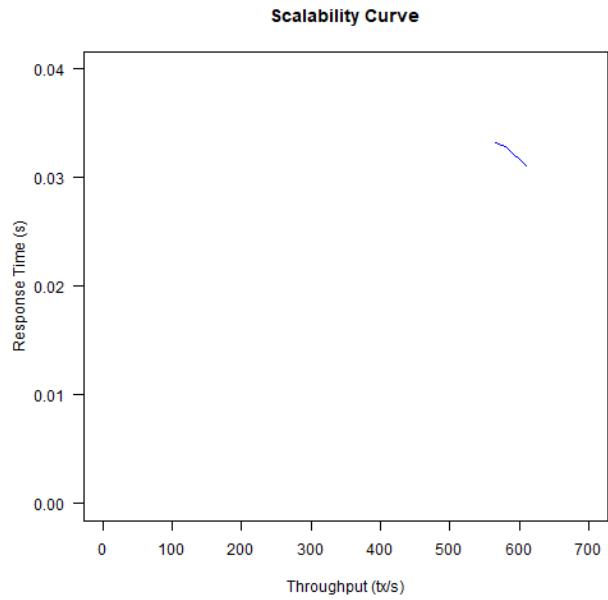


Figura 10: Comparação *wal_sync_method*

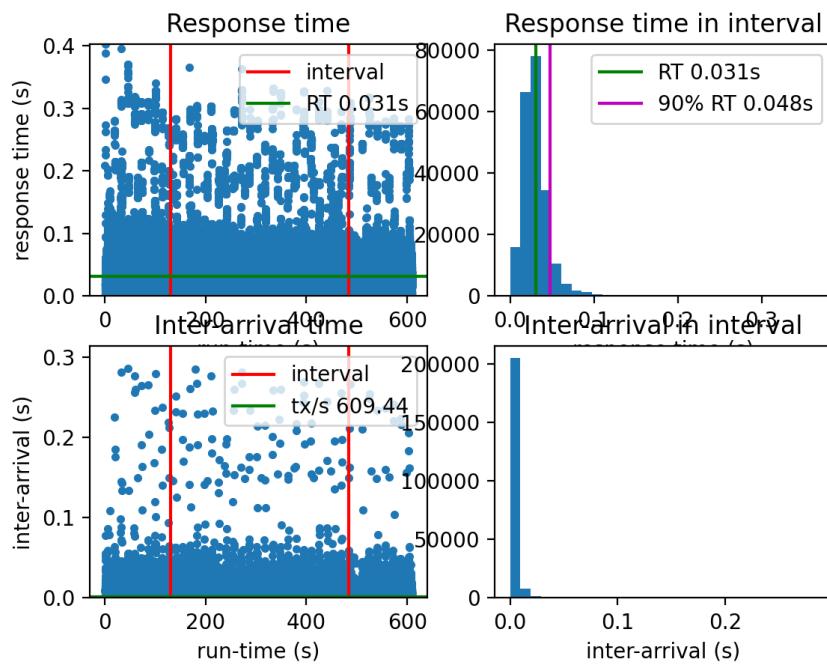


Figura 11: Métricas com a opção *wal_sync_method = fsync*

3.2.5 *full_pages_writes*

Quando este parâmetro está ativado, o servidor *PostgreSQL* grava o conteúdo de cada página do disco no *WAL* durante a primeira modificação da página após um *checkpoint*. Isso é necessário caso haja um *crash* do sistema durante a escrita de uma página e pode levar a uma página do disco que contém uma mistura de dados actualizados ou antigos.

A sua desactivação pode melhorar a *performance* mas, pelo lado negativo, tal como o *fsync*, pode levar a uma corrupção de dados irrecuperável em caso de falha de sistema mas em menor escala.

Tabela 9: Resultados obtidos com a alteração da configuração *full_pages_writes*

Opção	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
off	671.3272	0.0222	3.7180

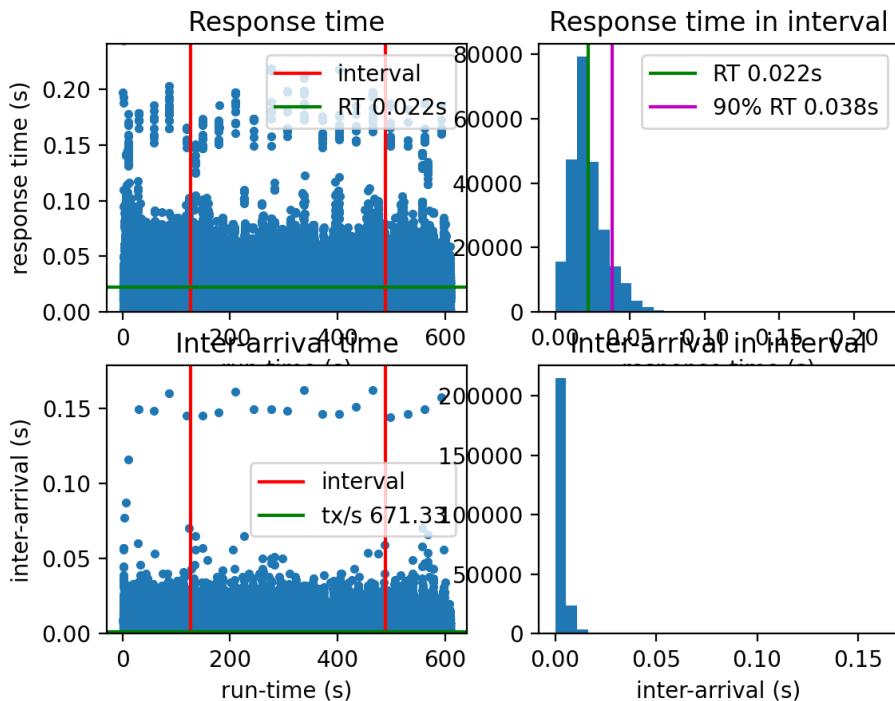


Figura 12: Métricas com a opção *full_page_writes* = *off*

3.2.6 *wal_buffers*

Com esta configuração escolhe-se a quantidade de memória partilhada utilizada para dados do *WAL* que ainda não foram escritos para o disco.

Tabela 10: Resultados obtidos com a alteração da configuração *wal_buffers*

Opção	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
2MB	579.4676	0.0326	4.2497
4MB	561.2677	0.0339	4.3127
8MB	599.4911	0.0315	4.3007
16MB	601.4456	0.0314	4.3489
32MB	611.0160	0.0309	4.2650

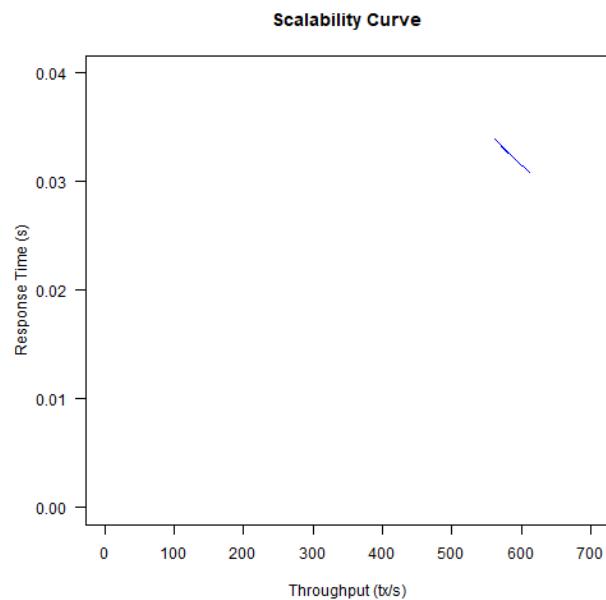


Figura 13: Comparação *wal_buffers*

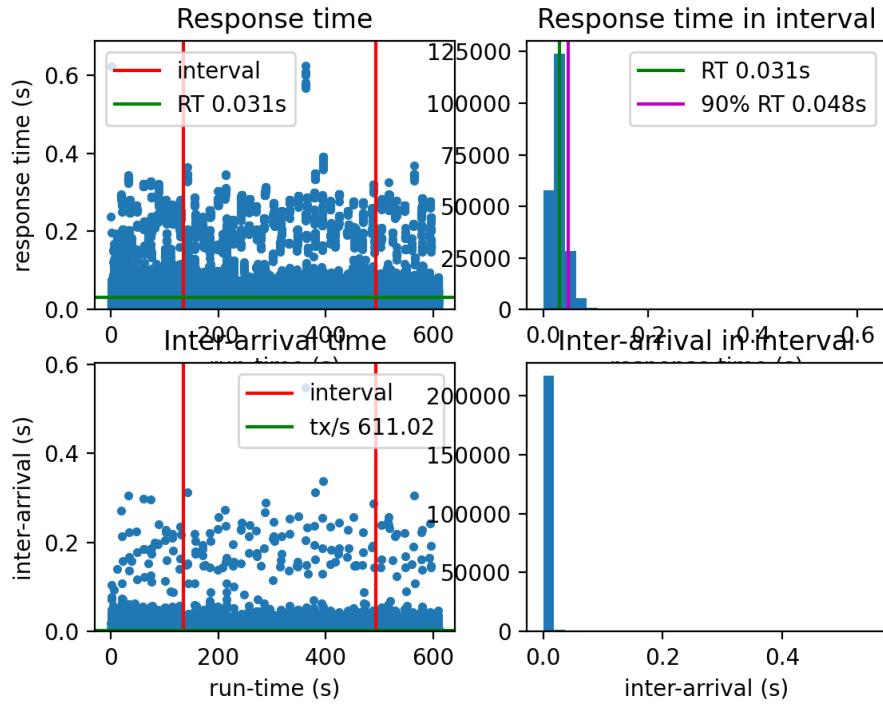


Figura 14: Métricas com a opção $wal_buffers = 32MB$

3.2.7 wal_writer_delay

Especifica com que frequência o *WAL writer* faz *flush* para a *WAL*. Após um *flush*, é feito um *sleep* com a duração definida por este parâmetro.

Tabela 11: Resultados obtidos com a alteração da configuração wal_writer_delay

Opção	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
100ms	528.6381	0.0361	4.3264
500ms	518.0617	0.0368	4.3372
1000ms	638.4395	0.0275	4.0479
2000ms	641.3710	0.0275	3.9612
5000ms	626.1576	0.0292	4.1821

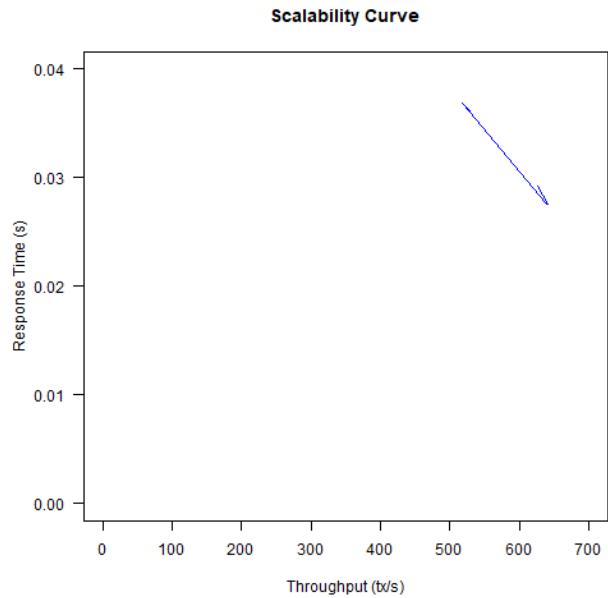


Figura 15: Comparaçāo *wal_writer_delay*

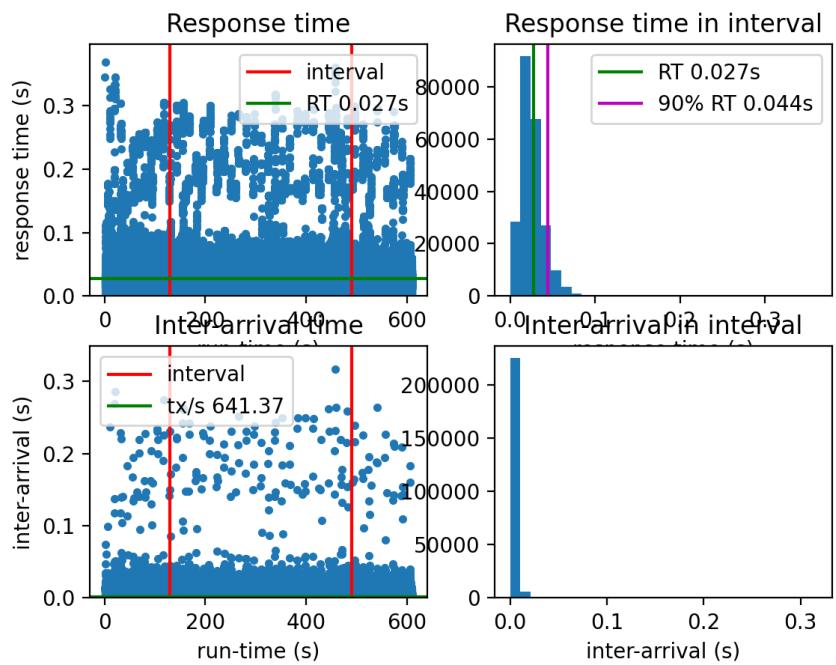


Figura 16: Métricas com a opção *wal_writer_delay = 2000ms*

3.2.8 wal_writer_flush_after

Tal como o *wal_writer_delay*, este parâmetro especifica com que frequência o *WAL writer* faz *flush* para a *WAL*. Estes dois parâmetros estão directamente ligados sendo que, caso o último *flush* tenha acontecido antes *wal_writer_delay* ms e tenha menos de *wal_writer_flush_after bytes*, então o *WAL* só é escrito para o sistema operativo e não para disco.

Tabela 12: Resultados obtidos com a alteração da configuração *wal_writer_flush_after*

Opção	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
2MB	648.0747	0.0250	3.7462
4MB	654.8258	0.0245	3.7055
8MB	583.7721	0.0327	4.2118
16MB	607.4482	0.0314	4.2079
32MB	633.3198	0.0289	4.1169
64MB	584.3400	0.0324	4.3328

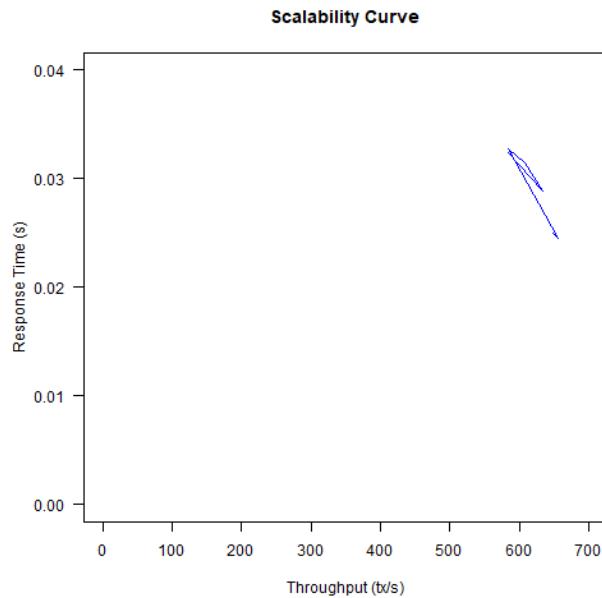


Figura 17: Comparaçāo *wal_writer_flush_after*

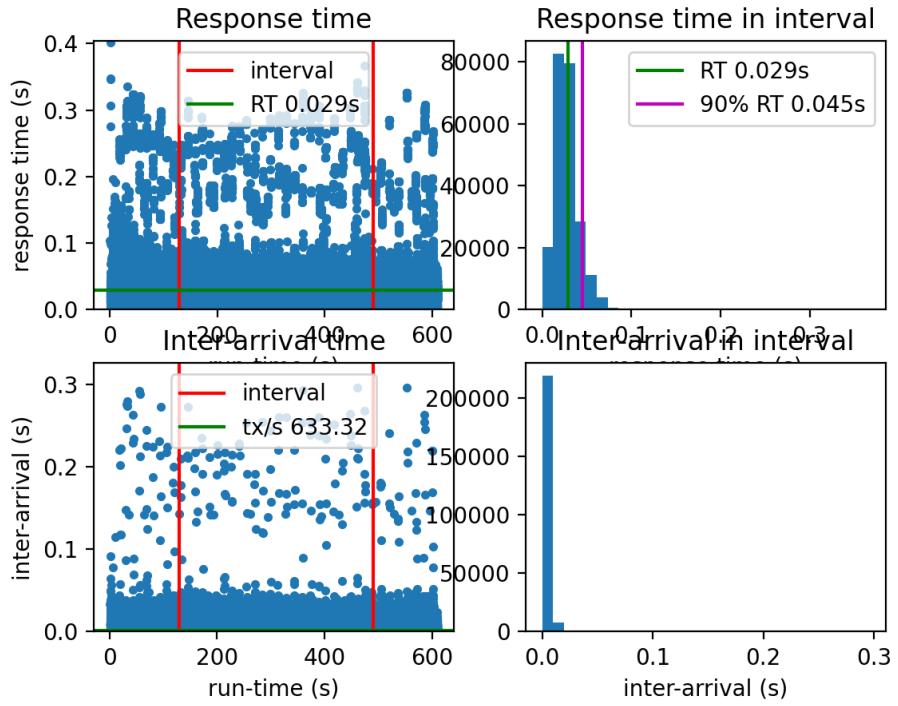


Figura 18: Métricas com a opção `wal_writer_flush_after = 32MB`

3.2.9 `commit_delay`

Com `commit_delay` é possível adicionar um *delay* antes de um *WAL flush* ser iniciado.

Tabela 13: Resultados obtidos com a alteração da configuração `commit_delay`

Opção	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
10ms	624.3614	0.0295	4.1752
200ms	622.4606	0.0289	4.4119
500ms	523.1947	0.0363	4.3891
1000ms	532.2371	0.0357	4.3252
1500ms	471.2457	0.0404	4.2246

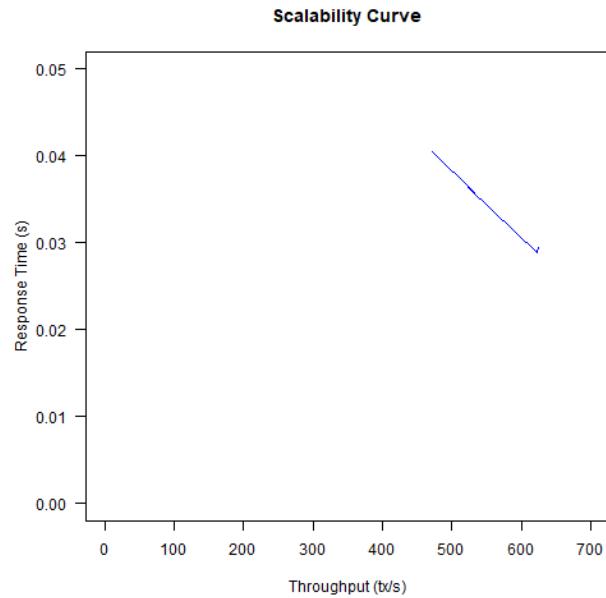


Figura 19: Comparação *commit_delay*

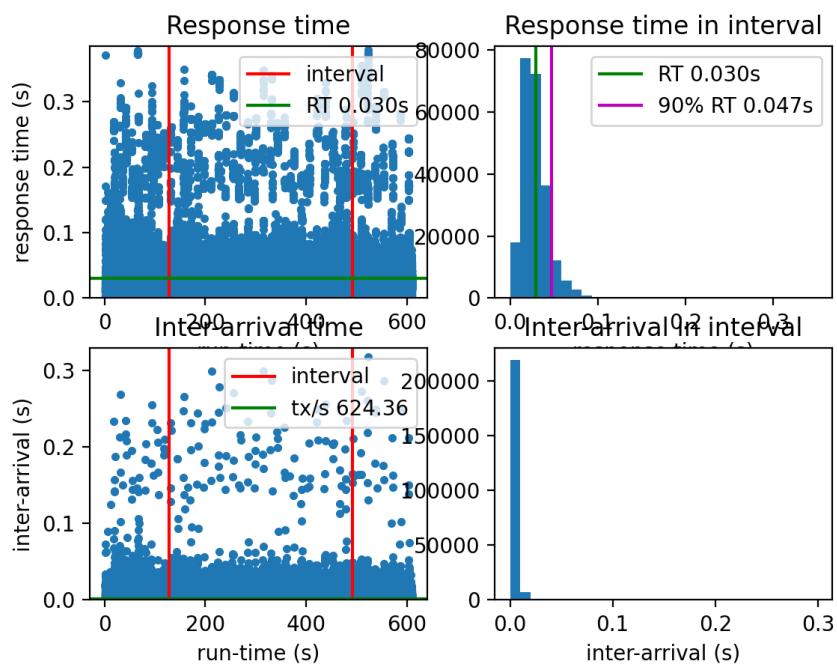


Figura 20: Métricas com a opção *commit_delay = 10ms*

3.2.10 *commit_siblings*

Nesta configuração pode-se escolher o número mínimo de transacções simultâneas antes de executar o atraso do *commit_delay*. Quanto maior o valor maior é a probabilidade que pelo menos uma outra transacção fique pronta para dar *commit* durante o intervalo de atraso.

Tabela 14: Resultados obtidos com a alteração da configuração *commit_siblings*

Opção	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
2	563.5583	0.0336	4.3229
4	537.1517	0.0342	4.3032
8	544.7155	0.0349	4.3028
16	646.2308	0.0257	3.7621

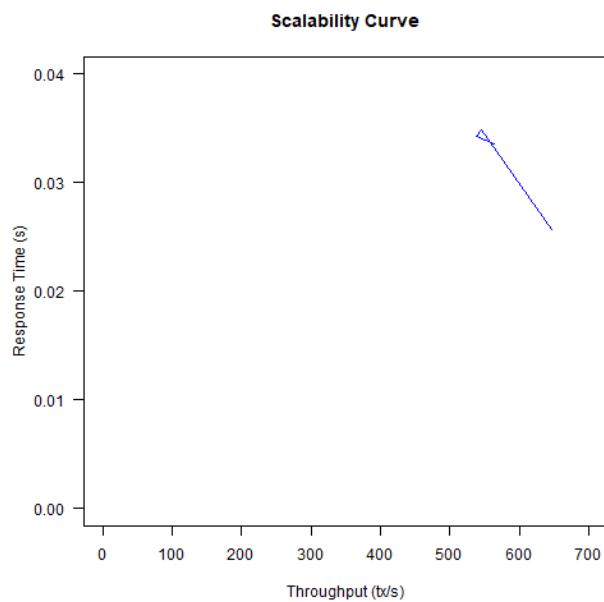


Figura 21: Comparaçāo *commit_siblings*

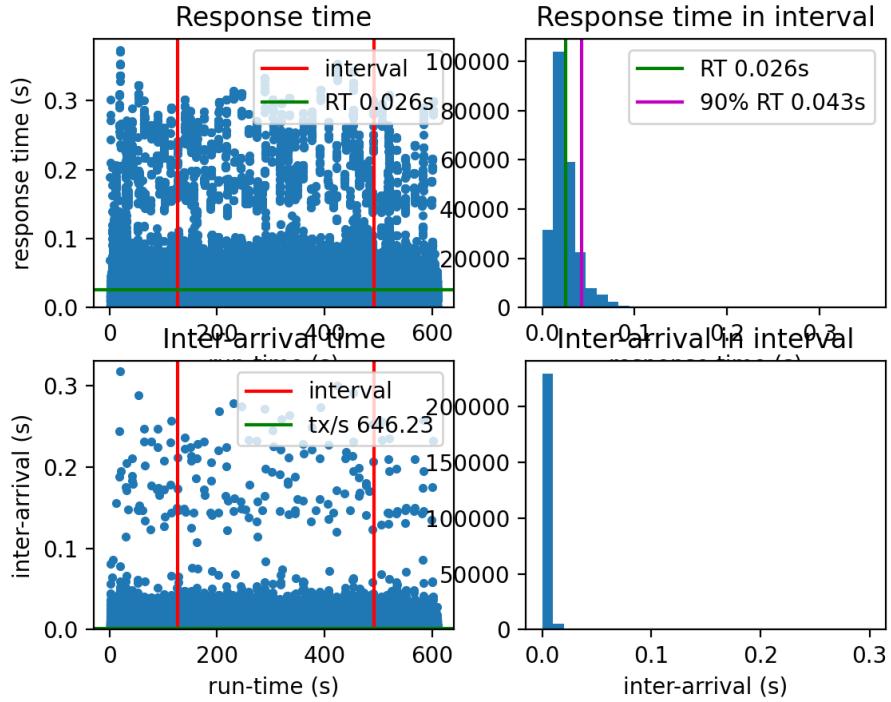


Figura 22: Métricas com a opção $commit_siblings = 16$

3.3 Checkpoints

3.3.1 $checkpoint_timeout$

Este parâmetro corresponde ao tempo máximo entre *WAL checkpoints* automáticos. Este tempo pode ser escolhido desde 30 segundos até um dia.

Foi testado até 15 minutos, uma vez que o *script* transaccional é apenas de 10 minutos. Ao ter um valor de *timeout* superior ao período de teste vamos garantir que não vamos ter *checkpoints* a correr no meio do nosso teste, o que por norma, o valor de *throughput* vai aumentar. Por outro lado em caso de *reboot*, como os *logs* eram feitos com menos frequência, a quantidade de *logs* a processar nessa altura será extremamente grande e ao invés de os processar em 30 minutos podemos levar dias a conseguir fazê-lo, que para o nosso caso estudo não é problemático pois o objectivo é optimizar a carga transaccional do TPCC.

Tabela 15: Resultados obtidos com a alteração da configuração *checkpoint_timeout*

Opção	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
1	625.2369	0.0297	4.2292
2	625.0972	0.0289	4.2029
5	635.7430	0.0278	4.1027
10	618.0168	0.0302	4.2610
15	636.0043	0.0277	4.0724

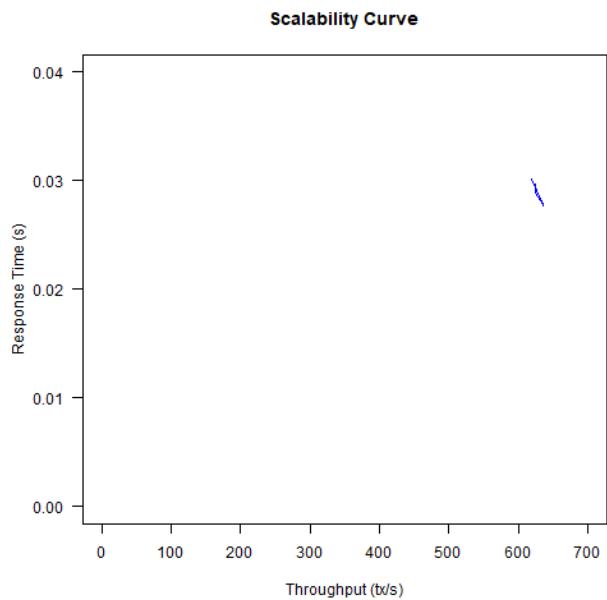


Figura 23: Comparaçāo *checkpoint_timeout*

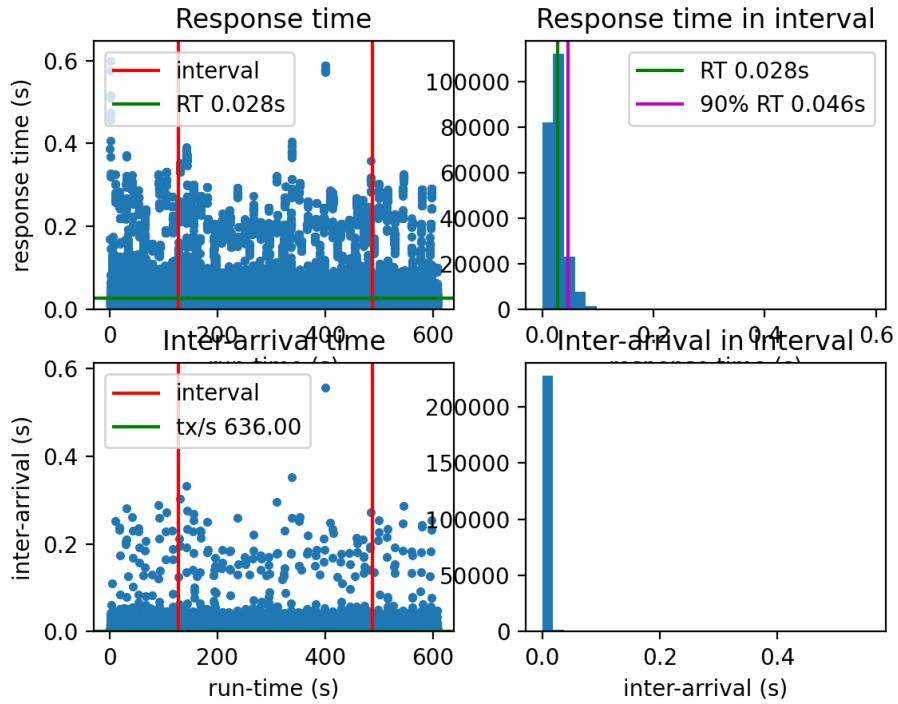


Figura 24: Métricas com a opção *checkpoint_timeout* = 15

3.3.2 *max_wal_size*

Corresponde ao tamanho máximo que o *WAL* pode crescer durante checkpoints automáticos. Este é um limite flexível pois este tamanho pode ser excedido caso a carga seja demasiado pesada.

Durante a execução dos vários testes, recebímos por parte do servidor do *postgres* a sugestão de aumentar este valor, o que nos levou a concluir que a alteração desta configuração iria nos levar a obter melhores resultados.

Tabela 16: Resultados obtidos com a alteração da configuração *max_wal_size*

Opção	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
2GB	651.4825	0.0260	3.9915
3GB	664.3522	0.0237	3.8167
4GB	589.4551	0.0323	4.4371
5GB	664.2822	0.0233	3.5602
6GB	653.4738	0.0247	3.8909

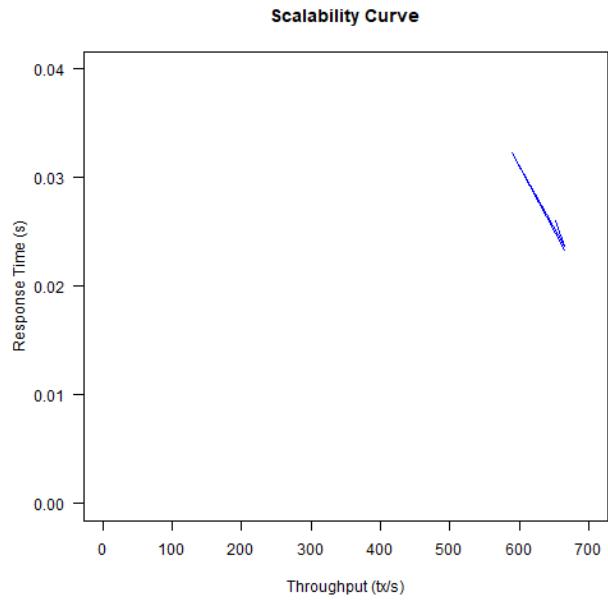


Figura 25: Comparação *max_wal_size*

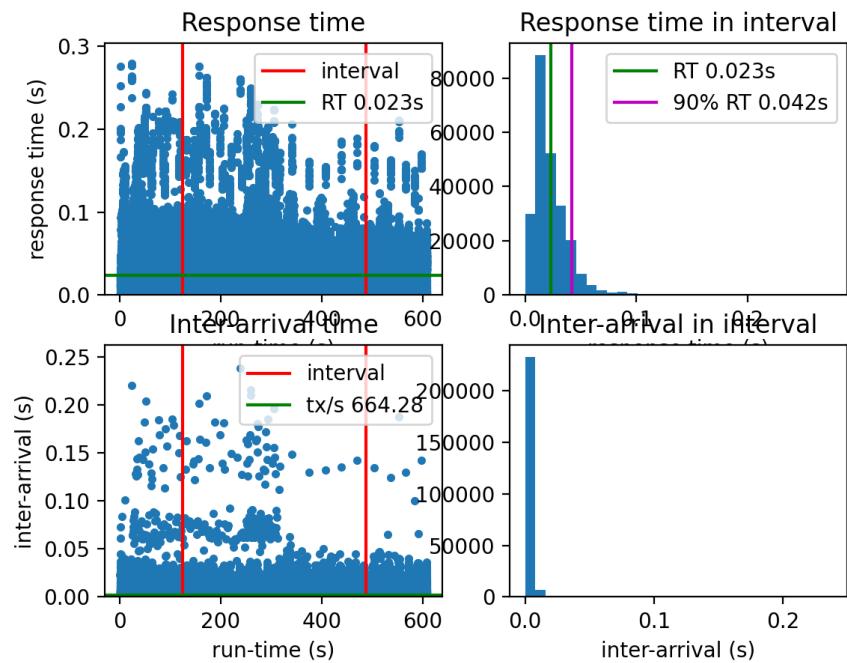


Figura 26: Métricas com a opção *max_wal_size = 5GB*

3.3.3 *min_wal_size*

Contando que o uso do disco *WAL* permaneça abaixo desta configuração, os arquivos do *WAL* antigos são sempre reciclados para uso futuro em vez de serem removidos. Isto pode ser usado para garantir que espaço suficiente do *WAL* seja reservado para lidar com picos no uso do *WAL*, por exemplo, ao executar grandes trabalhos em lote.

Tabela 17: Resultados obtidos com a alteração da configuração *min_wal_size*

Opção	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
80MB	548.5228	0.0348	4.5326
160MB	622.7248	0.0298	4.2495
320MB	589.9841	0.0310	4.3795
640MB	610.8227	0.0309	4.3549

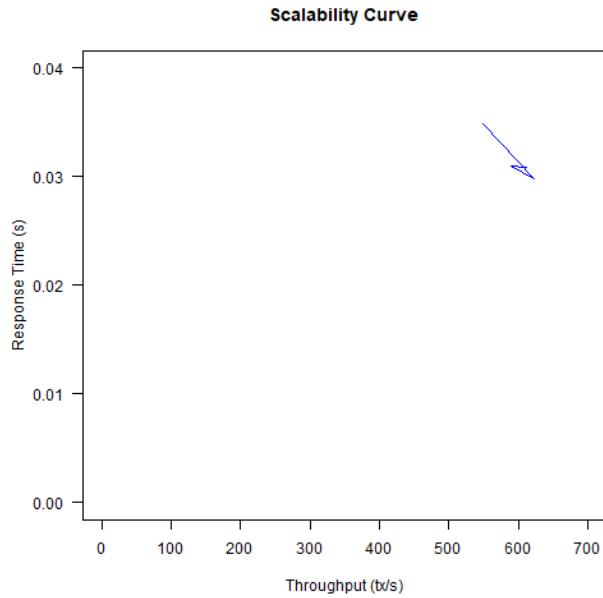


Figura 27: Comparaçao *min_wal_size*

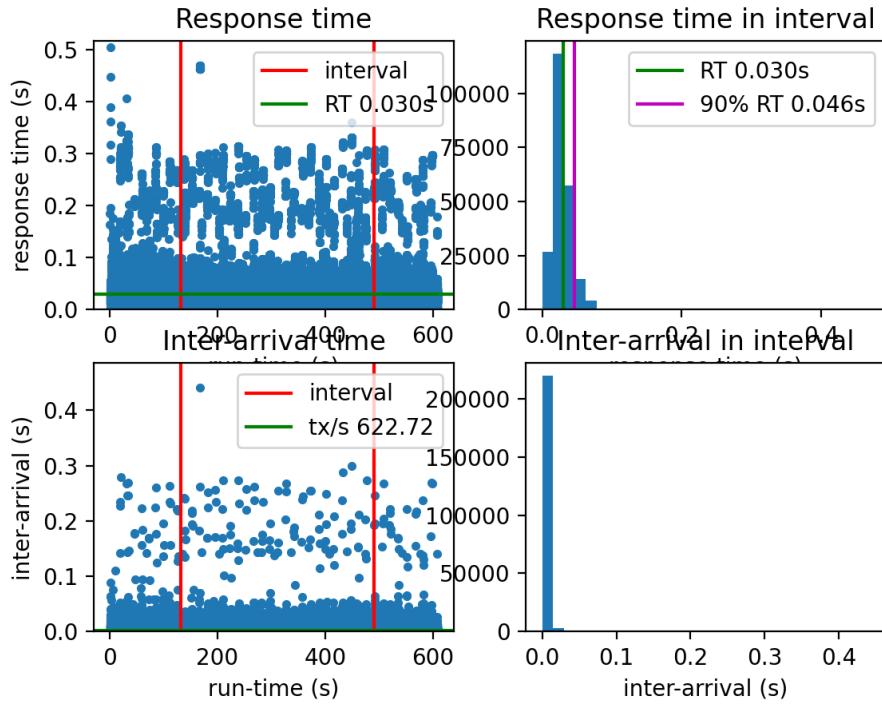


Figura 28: Métricas com a opção $\text{min_wal_size} = 160MB$

3.3.4 *checkpoint_completion_target*

Especifica a meta de conclusão do *checkpoint*, como uma fração do tempo total entre os pontos de verificação.

Tabela 18: Resultados obtidos com a alteração da configuração *checkpoint_completion_target*

Opção	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
0.0	631.1436	0.0288	4.1392
0.2	633.4113	0.0282	4.1248
0.4	631.2983	0.0267	3.7272
0.6	613.5507	0.0299	4.2307
0.8	611.9545	0.0307	4.2712
1.0	592.6204	0.0321	4.2384

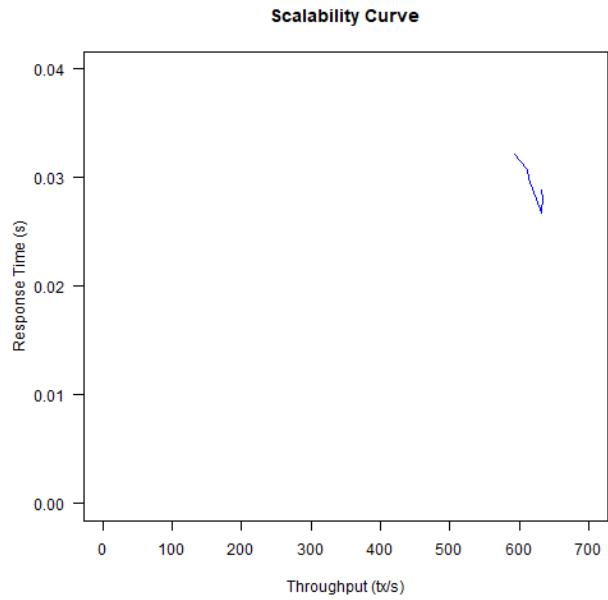


Figura 29: Comparação *checkpoint_completion_target*

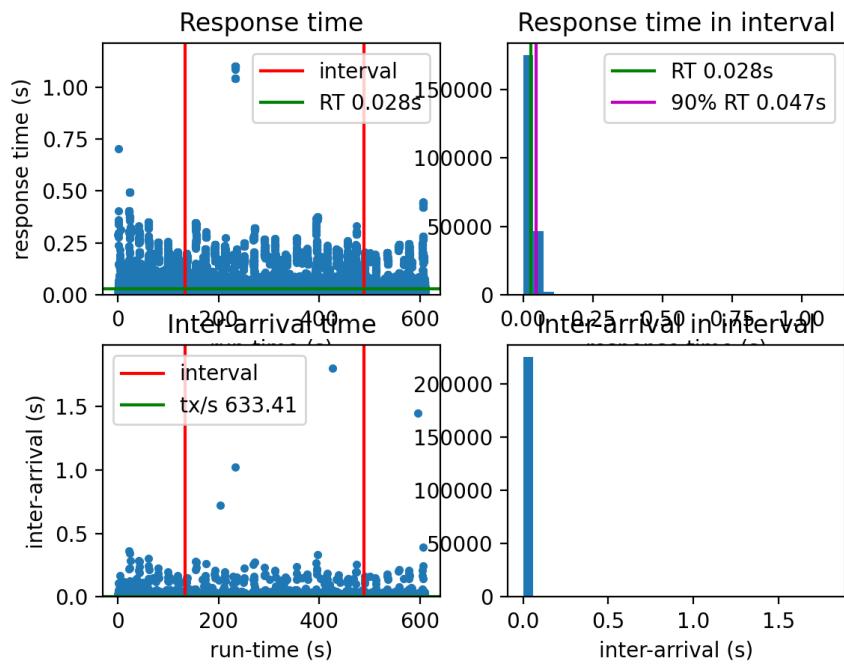


Figura 30: Métricas com a opção *checkpoint_completion_target = 0.2*

3.3.5 *checkpoint_warning*

Esta configuração permite que seja escrita uma mensagem nos *logs* do servidor se os *checkpoints* causados pelo preenchimento dos arquivos do segmento *WAL* acontecerem mais próximos uns dos outros do que o período de tempo definido, ao qual sugere que *max_wal_size* deve ser aumentado.

Tabela 19: Resultados obtidos com a alteração da configuração *checkpoint_warning*

Opção	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
0s	623.9151	0.0295	4.2278
10s	619.6465	0.0300	4.2393
30s	509.6396	0.0374	4.5581
60s	612.2521	0.0307	4.2820
120s	632.5638	0.0277	4.1516

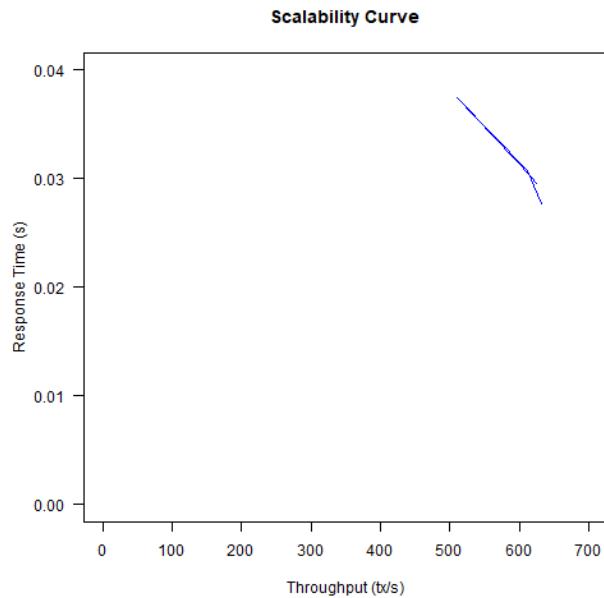


Figura 31: Comparaçāo *checkpoint_warning*

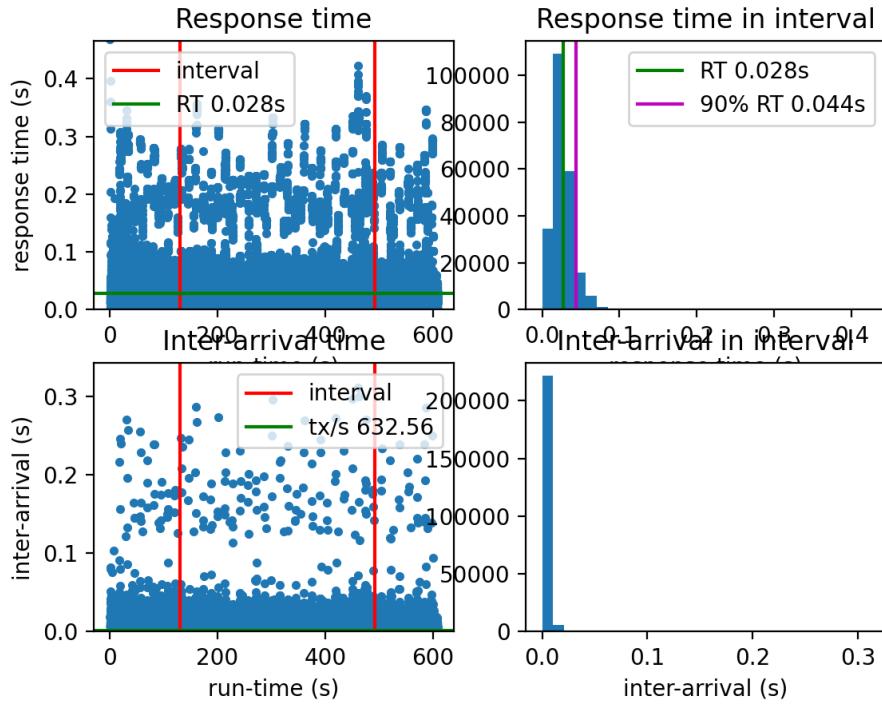


Figura 32: Métricas com a opção *checkpoint_warning* = 120s

3.4 Archiving

3.4.1 *archive_mode*

Quando esta configuração está activada, os segmentos *WAL* concluídos são enviados para o armazenamento de *archive*.

Tabela 20: Resultados obtidos com a alteração da configuração *archive_mode*

Opção	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
always	554.9108	0.0344	4.2405
on	555.3080	0.0345	4.2142
off	648.7737	0.0254	3.8087

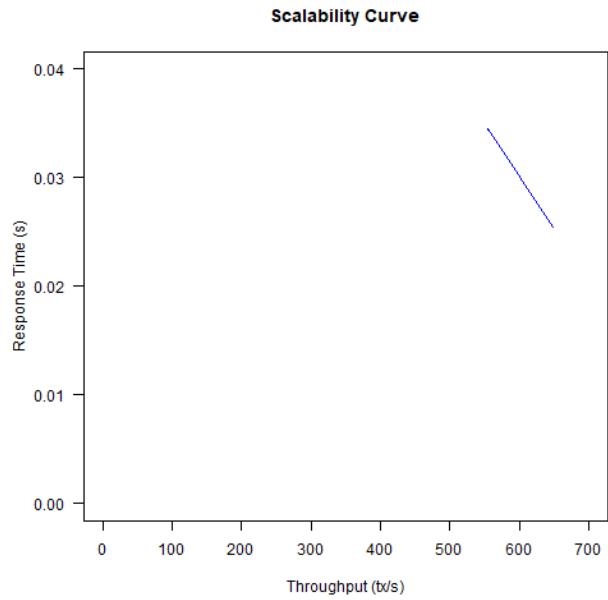


Figura 33: Comparação *archive_mode*

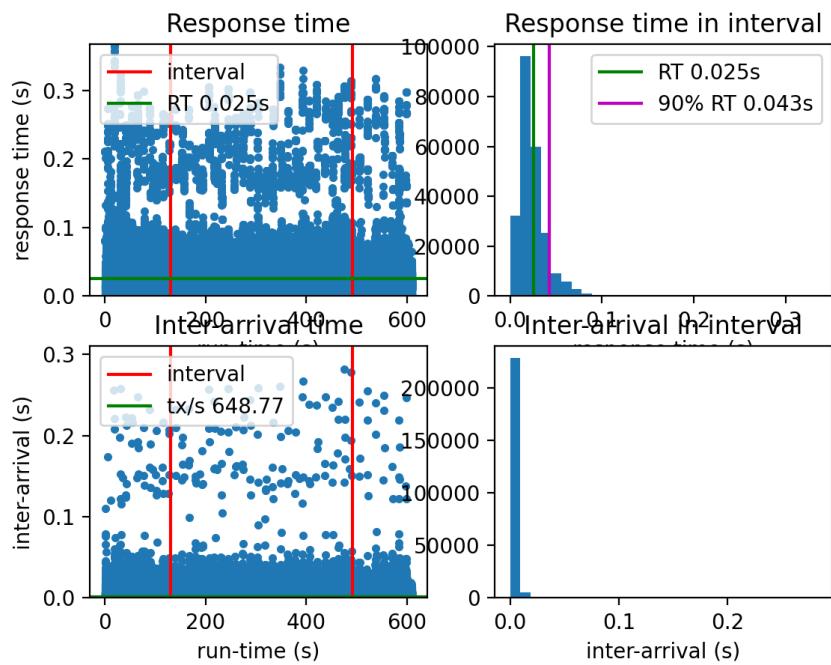


Figura 34: Métricas com a opção *archive_mode = off*

Existem outros testes que se podem fazer com o *archiving*, tais como mudar os parâmetros do *archive_timeout*, mas isso já requeria ter de se fornecer um *archive_command* para guardar os ficheiros WAL. Como o nosso objetivo neste trabalho prático é aumentar o desempenho do TPC-C, decidimos não fazer mais testes relativos à secção *Archiving* uma vez que esta apenas introduz carga extra sobre o nosso servidor de base de dados, no entanto num contexto de uma base de dados de produção poderia ser extremamente útil tirar partido desta secção.

3.5 Isolamento

Um aspecto que se teve em consideração no processo de *benchmark* da aplicação foi considerar os diferentes níveis de isolamento fornecidos pelo sistema de base de dados. Por omissão, o *benchmark* usa o *Repeatable Read* como nível de isolamento onde apenas é possível o fenómeno de *serialization anomaly*. De forma a analisar o impacto que os outros níveis de isolamento têm no desempenho da aplicação, foram analisados os níveis de isolamento apresentados na tabela abaixo. Para além dos resultados obtidos, também se pode observar quais os fenómenos que cada nível de isolamento trata.

Tabela 21: Resultados obtidos com os diferentes níveis de isolamento

	Read Committed	Read Uncommitted	Repeatable Read (Referência)	Serializable
Throughput (tx/s)	621.9828	635.1476	645.2408	576.3303
Response Time (s)	0.0310	0.0296	0.0262	0.0313
Abort Rate (%)	0.003	0.003	3.8587	7.9571
Dirty Read	Impossível	Permitido Não no PostgreSQL	Impossível	Impossível
Nonrepeatable Read	Possível	Possível	Impossível	Impossível
Phantom Read	Possível	Possível	Permitido Não no PostgreSQL	Impossível
Serialization Anomaly	Possível	Possível	Possível	Impossível

Pode-se constatar que a maior diferença entre os diferentes níveis de isolamento é o *abort rate*. Este valor é bastante mais baixo nos isolamentos *Read Committed* e *Read Uncommitted* e, ao contrário do que se estava à espera, é bastante alto no *Serializable*. Quanto ao débito, como era esperado, o *Serializable* apresenta um valor baixo por se tratar de um isolamento mais rigoroso.

Uma vez que o principal objectivo desta fase de análise e testes é obter maior débito possível, decidiu-se usar o nível de isolamento por omissão do *benchmark*, isto é, *Repeatable Read*.

3.6 Análise de resultados

Nesta secção irão ser analisados os resultados de uma forma global. Primeiro vão ser combinadas as melhores configurações relativas ao domínio de *settings*, de seguida as de *checkpoint*, e por fim, uma combinação das configurações de ambos os domínios.

3.6.1 *Settings*

As configurações escolhidas relativas aos *settings* e os respectivos valores obtidos foram os seguintes:

- *wal_buffers* = 32MB
- *wal_writer_delay* = 2000ms
- *wal_writer_flush_after* = 2MB
- *commit_delay* = 10ms
- *commit_siblings* = 16
- *synchronous_commit* = off

	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
1º	687.1240	0.0141	1.5811
2º	677.7936	0.0153	1.6961
3º	692.0450	0.0107	1.2009

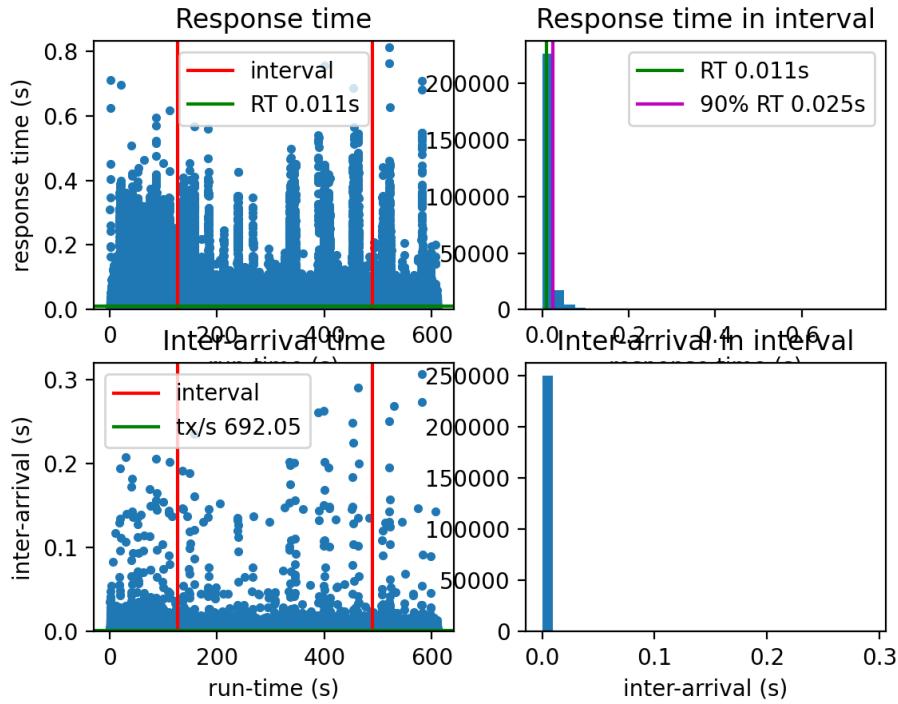


Figura 35: Métricas com as melhores configurações das *settings*

Comparando os resultados obtidos com os valores inciais de referência, pode-se observar que todas as métricas melhoraram com esta configuração, principalmente os valores de débito e o *abort rate*, atingindo um valor inferior ao que se tinha estipulado como objectivo.

3.6.2 Checkpoints

As configurações escolhidas relativas aos *checkpoints* e os respetivos valores obtidos foram os seguintes:

- *checkpoint_timeout* = 15min
- *max_wal_size* = 5GB
- *min_wal_size* = 160MB
- *checkpoint_completion_target* = 0.2
- *checkpoint_warning* = 120s

	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
1º	649.3530	0.0263	3.9409
2º	647.3852	0.0256	3.9428
3º	649.6655	0.0258	4.0057

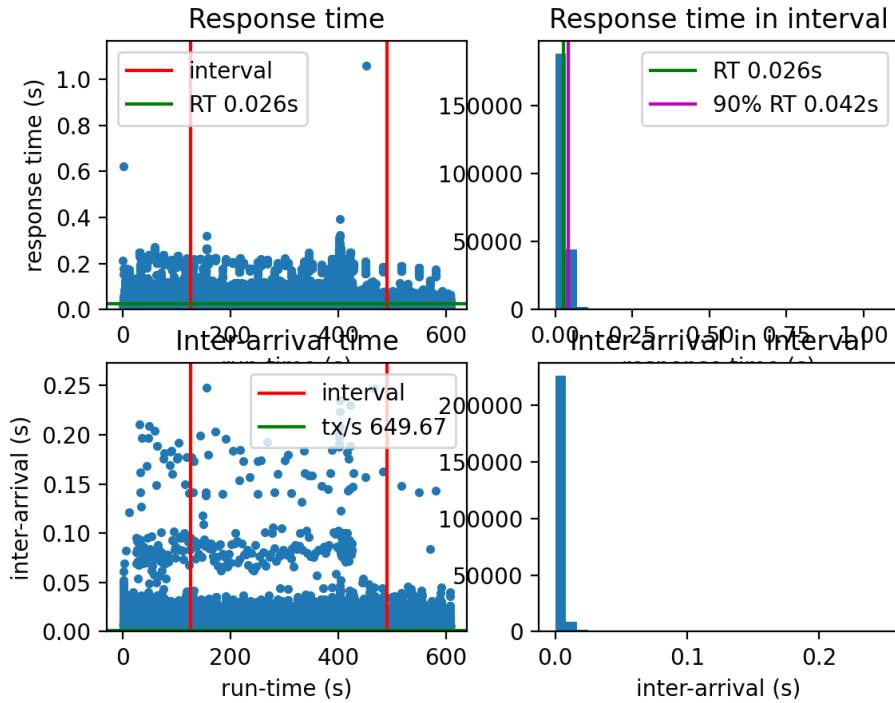


Figura 36: Métricas com as melhores configurações dos *checkpoints*

Em comparação com os resultados obtidos com as configurações de *settings*, as configurações feitas com os *checkpoints* não aumentaram significativamente as métricas, sendo que a que teve maior aumento foi o débito. De qualquer forma, estas configurações serão usadas na configuração final.

3.6.3 Configuração final

Uma vez feita a análise das métricas obtidas de cada configuração feita, decidiu-se que para uma configuração final do *benchmark* seriam usados os dois tipos de configuração mencionadas anteriormente, isto é, uma combinação de configurações relativas aos domínios de *settings* e *checkpoints*.

De seguida é apresentada a configuração final escolhida bem como os valores obtidos com

essa mesma configuração.

- $wal_buffers = 32\text{MB}$
- $wal_writer_delay = 2000\text{ms}$
- $wal_writer_flush_after = 2\text{MB}$
- $commit_delay = 10\text{ms}$
- $commit_siblings = 16$
- $synchronous_commit = \text{off}$
- $checkpoint_timeout = 15\text{min}$
- $max_wal_size = 5\text{GB}$
- $min_wal_size = 160\text{MB}$
- $checkpoint_completion_target = 0.2$
- $checkpoint_warning = 120\text{s}$

	Throughput (tx/s)	Response Time (s)	Abort Rate (%)
1º	690.7203	0.0076	0.9016
2º	690.8735	0.0084	1.0360
3º	691.0072	0.0099	1.1441

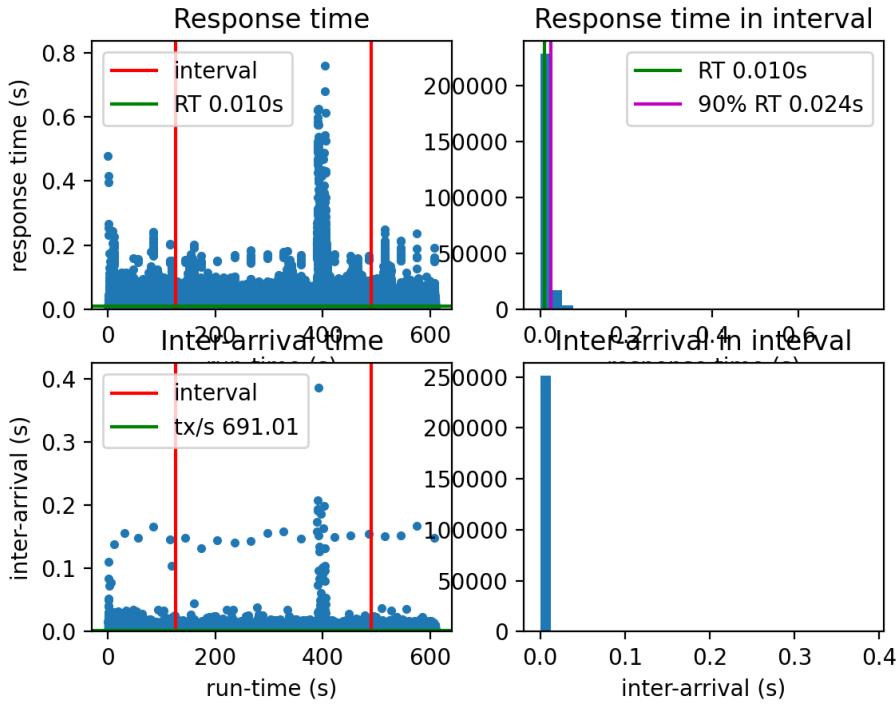


Figura 37: Métricas finais.

Observando os dados obtidos, verifica-se que se consegui melhorar o desempenho do sistema consideravelmente. Em comparação com a configuração inicial, o débito aumentou de 615 para 690 tx/s, o tempo de resposta diminui de 0.0306 s para cerca de 0.008 e o *abort rate* passou de 4.2 % para cerca de 1%.

Em relação a esta fase de análise, pode-se concluir que efectivamente consegui-se melhorar o desempenho da configuração inicial com sucesso. Todas as métricas apresentam valores satisfatórios e, como se pode observar através dos gráficos, não há muita dispersão dos valores das mesmas.

4 Optimização de Interrogações analíticas

Uma vez feita a análise e optimização da carga transacional do sistema, irá ser feita também uma análise e optimização de algumas interrogações analíticas.

É pretendido nesta secção analisar os planos das interrogações e, através de mecanismos de redundância, aumentar o desempenho das mesmas interrogações. Particularmente, serão usados índices e vistas materializadas de maneira a diminuir o tempo de resposta de uma interrogação.

O sistema TPC-C contém já um conjunto de índices definidos para algumas tabelas, nomeadamente, os que se apresentam na Figura 38.

	tablename name	indexname name
1	customer	ix_customer
2	customer	keycustomer
3	customer	pk_customer
4	district	keydistrict
5	district	pk_district
6	history	keyhistory
7	item	keyitem
8	item	pk_item
9	new_order	ix_new_order
10	new_order	keyneworder
11	order_line	ix_order_line
12	order_line	keyorderline
13	order_line	pk_order_line
14	orders	ix_orders
15	orders	keyorders
16	orders	pk_orders
17	stock	ix_stock
18	stock	keystock
19	stock	pk_stock
20	warehouse	keywarehouse
21	warehouse	pk_warehouse

Figura 38: Lista de índices

Nas secções que se seguem, serão apresentadas as interrogações escolhidas para análise, os respectivos planos e tempos de execução pré e pós optimização, os mecanismos de redundância adicionados.

4.1 Query Analítica 1

A *query* apresentada abaixo lista o número de clientes agrupados e ordenados pelo tamanho de encomendas que fizeram. O conjunto resultante da relação entre clientes e o tamanho das suas encomendas é ordenado pelo tamanho das encomendas e conta quantos clientes negociaram da mesma maneira.

```

1 select      c_count , count(*) as custdist
2 from        (select c_id , count(o_id)
3              from customer left outer join orders on (
4                  c_w_id = o_w_id
5                  and c_d_id = o_d_id
6                  and c_id = o_c_id
7                  and o_carrier_id > 8)
8              group by c_id) as c_orders (c_id , c_count)
9 group by c_count
10 order by custdist desc , c_count desc

```

Listing 1: Query Analítica 1

O plano e tempo de execução podem ser observados na Figura 39. O tempo de execução da interrogação é de aproximadamente 2.8s.

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Sort (cost=172820..172820.5 rows=200 width=16) (actual=2842.036..2857.2...	0.075 ms	2857.209 ms	↑ 2.25	89	200	1
2.	→ Aggregate (cost=172787.86..172812.36 rows=200 width=16) (actual=2...	0.499 ms	2857.134 ms	↑ 2.25	89	200	1
3.	→ Sort (cost=172787.86..172795.36 rows=3000 width=8) (actual=284...	1.122 ms	2856.635 ms	↑ 1	3000	3000	1
4.	→ Subquery Scan (cost=171824.55..172614.6 rows=3000 width=...	0.395 ms	2855.514 ms	↑ 1	3000	3000	1
5.	→ Aggregate (cost=171824.55..172584.6 rows=3000 width=...	1.979 ms	2855.12 ms	↑ 1	3000	3000	1
6.	→ Gather Merge (cost=171824.55..172524.6 rows=600...	-5572.174 ms	2853.142 ms	↓ 1.5	9000	6000	1
7.	→ Sort (cost=170824.53..170832.03 rows=3000 w...	4.521 ms	8425.317 ms	↑ 1	3000	3000	3
8.	→ Aggregate (cost=170621.26..170651.26 ro...	891.852 ms	8420.796 ms	↑ 1	3000	3000	3
9.	→ Merge Left Join (cost=93196.85..1656...	1564.833 ms	7528.944 ms	↑ 1.25	800000	1000000	3
10.	→ Index Only Scan using pk_custom...	1889.518 ms	1889.518 ms	↑ 1.25	800000	1000000	3
11.	→ Materialize (cost=93196.42..949...	309.805 ms	4074.594 ms	↑ 1.01	352627	355261	3
12.	→ Sort (cost=93196.42..94084....	1058.835 ms	3764.79 ms	↑ 1.01	352627	355261	3
13.	→ Seq Scan on public.order... Filter: (orders.o_carrier_id > 8) Rows Removed by Filter: 2 134445	2705.955 ms	2705.955 ms	↑ 1.01	352794	355261	3

Figura 39: Plano e Tempo de execução da Query Analítica 1

Pode-se observar no plano de execução que a *query* já está a usar o índice *pk_customer* sobre a tabela *customer* e envolve as três colunas da tabela. Na tabela *pg_indexes* é possível ver a definição deste índice. O índice usado já contempla todos os atributos da tabela *customer* usados para fazer o *join* das tabelas de maneira a optimizar esta mesma operação.

```
CREATE UNIQUE INDEX pk_customer ON public.customer USING btree (c_w_id,
c_d_id, c_id)
```

Verificou-se que nenhum índice está a ser usado sobre a tabela *orders*. De maneira a forçar o uso dos índices definidos sobre a mesma, o *sequential scan* foi desligado para análise do plano de execução, que pode ser observado na figura que se segue.

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Sort (cost=199737.75..199738.25 rows=200 width=16) (actual=3046.572..30...	0.042 ms	3054.014 ms	↑ 3.13	64	200	1
2.	→ Aggregate (cost=199705.61..199730.11 rows=200 width=16) (actual=3...	0.43 ms	3053.972 ms	↑ 3.13	64	200	1
3.	→ Sort (cost=199705.61..199713.11 rows=3000 width=8) (actual=304...	0.955 ms	3053.543 ms	↑ 1	3000	3000	1
4.	→ Subquery Scan (cost=198742.3..199532.35 rows=3000 width=...	0.396 ms	3052.589 ms	↑ 1	3000	3000	1
5.	→ Aggregate (cost=198742.3..199502.35 rows=3000 width=...	1.797 ms	3052.193 ms	↑ 1	3000	3000	1
6.	→ Gather Merge (cost=198742.3..199442.35 rows=600...	-5985.453 ms	3050.397 ms	↓ 1.5	9000	6000	1
7.	→ Sort (cost=197742.27..197749.77 rows=3000 w...	4.468 ms	9035.85 ms	↑ 1	3000	3000	3
8.	→ Aggregate (cost=197539.01..197569.01 ro...	1089.12 ms	9031.383 ms	↑ 1	3000	3000	3
9.	→ Merge Left Join (cost=123114.2..1925...	1339.647 ms	7942.263 ms	↑ 1.25	800000	1000000	3
10.	→ Index Only Scan using pk_custom...	941.883 ms	941.883 ms	↑ 1.25	800000	1000000	3
11.	→ Materialize (cost=123113.77..12...	181.212 ms	5660.733 ms	↓ 1.01	176058	175267	3
12.	→ Sort (cost=123113.77..1235...	1117.31 ms	5479.522 ms	↓ 1.01	176058	175267	3
13.	→ Index Scan using order... Filter: (orders.o_carrier_id > 9) Rows Removed by Filter: 2 311043	4362.213 ms	4362.213 ms	↓ 1.01	176196	175267	3

Figura 40: Plano e Tempo de execução da Query Analítica 1 com *seqcan = OFF*

Apesar de agora estarem a ser usados os índices sobre as duas tabelas, o tempo de execução da interrogação aumentou, pelo que a decisão de usar ambos os índices deve ser descartada.

4.1.1 Índices

Na operação de *join*, está a ser também usada uma comparação com o atributo *o_carrier_id*, cujo sistema de base de dados não contém nenhum índice que envolve o mesmo. De forma a analisar o impacto no desempenho, foi criado um índice sobre esse atributo da tabela.

```
CREATE INDEX i1 ON orders(o_carrier_id);
```

Analizando mais uma vez o plano da interrogação, desta vez com a introdução de um novo índice, verifica-se o sistema decidiu efectivamente usá-lo e pode constatar-se que existe uma ligeira diminuição no tempo de execução que passou de 2.8s para 2.1s.

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Sort (cost=152827.98..152828.48 rows=200 width=16) (actual=2125.055..2125...	0.049 ms	2125.73 ms	↑ 2.25	89	200	1
2.	→ Aggregate (cost=152795.84..152820.34 rows=200 width=16) (actual=212...	0.542 ms	2125.682 ms	↑ 2.25	89	200	1
3.	→ Sort (cost=152795.84..152803.34 rows=3000 width=8) (actual=2124....	1.04 ms	2125.14 ms	↑ 1	3000	3000	1
4.	→ Subquery Scan (cost=151832.53..152622.58 rows=3000 width=8...	0.434 ms	2124.1 ms	↑ 1	3000	3000	1
5.	→ Aggregate (cost=151832.53..152592.58 rows=3000 width=...	1.923 ms	2123.666 ms	↑ 1	3000	3000	1
6.	→ Gather Merge (cost=151832.53..152532.58 rows=6000...	-4121.967 ms	2121.744 ms	↓ 1.5	9000	6000	1
7.	→ Sort (cost=150832.5..150840 rows=3000 width=1...	4.413 ms	6243.711 ms	↑ 1	3000	3000	3
8.	→ Aggregate (cost=150629.24..150659.24 rows...	1173.922 ms	6239.299 ms	↑ 1	3000	3000	3
9.	→ Merge Left Join (cost=73204.83..14562...	1561.195 ms	5065.377 ms	↑ 1.25	800000	1000000	3
10.	→ Index Only Scan using pk_customer...	1091.607 ms	1091.607 ms	↑ 1.25	800000	1000000	3
11.	→ Materialize (cost=73204.4..74980.7...	326.877 ms	2412.576 ms	↑ 1.01	352590	355261	3
12.	→ Sort (cost=73204.4..74092.55 ...	1063.386 ms	2085.699 ms	↑ 1.01	352590	355261	3
13.	→ Bitmap Heap Scan on pub... Recheck Cond: (orders.o_carr... Heap Blocks: exact=17554	879.999 ms	1022.314 ms	↑ 1.01	352794	355261	3
14.	→ Bitmap Index Scan u... Index Cond: (orders.o_c... carrier_id > 8)	142.315 ms	142.315 ms	↑ 1.01	352794	355261	3

Figura 41: Plano e Tempo de execução da Query Analítica 1 usando o índice *i1*

4.1.2 Vistas Materializadas

Como se pode observar na definição da interrogação, existe uma *subquery* responsável por criar uma tabela que consiste na junção da tabela *customer* e *orders*. Uma outra forma que pode ser usada para optimizar o desempenho da interrogação é criar uma vista materializada sobre essa tabela intermédia fazendo com que a mesma seja guardada em disco e não ter que ser calculada em *runtime*.

```
1  create materialized view q13_view as (
2      select c_id, count(o_id) from customer left outer join orders on (
3          c_w_id = o_w_id
4          and c_d_id = o_d_id
5          and c_id = o_c_id
6          and o_carrier_id > 8)
7      group by c_id)
```

Listing 2: Vista Materializada 1

Tal como era esperado, tanto o tempo de execução como o plano de execução variam drasticamente. Após a introdução da vista, o tempo de execução da interrogação diminui de 2.1s para 0.867ms. Quanto ao plano de execução, este passou apenas três operações, diminuindo bastante a sua complexidade.

#	Node	Timings		Rows				Loops
		Exclusive	Inclusive	Rows X	Actual	Plan		
1.	→ Sort (cost=65.77..65.99 rows=89 width=16) (actual=0.861..0.867 row...)	0.032 ms	0.867 ms	↑ 1	89	89	1	
2.	→ Aggregate (cost=62..62.89 rows=89 width=16) (actual=0.822..0....)	0.549 ms	0.836 ms	↑ 1	89	89	1	
3.	→ Seq Scan on public.q13_view as c_orders (cost=0..47 rows=...	0.287 ms	0.287 ms	↑ 1	3000	3000	1	

Figura 42: Plano e Tempo de execução da Query Analítica 1 com Vista Materializada

A criação da vista materializada tem um impacto gigante no desempenho da interrogação pois as operações com maior custo (criação da *subquery*) são eliminadas pela materialização das mesmas.

Uma vez que toda a tabela definida pela vista materializada tem que ser percorrida para se proceder ao agrupamento de valores bem como a sua contagem, qualquer índice sobre esta tabela acaba por ser desnecessário.

4.2 Query Analítica 2

A query analítica 2 selecionada determina todos os fornecedores que não enviaram alguns itens necessários para uma encomenda num tempo útil para um determinado país.

```

1 select su_name, count(*) as numwait
2 from supplier, order_line 11, orders, stock, nation
3 where ol_o_id = o_id
4   and ol_w_id = o_w_id
5   and ol_d_id = o_d_id
6   and ol_w_id = s_w_id
7   and ol_i_id = s_i_id
8   and mod((s_w_id * s_i_id), 10000) = su_suppkey
9   and 11.ol_delivery_d > o_entry_d
10  and not exists (select *
11    from order_line 12
12    where 12.ol_o_id = 11.ol_o_id
13      and 12.ol_w_id = 11.ol_w_id
14      and 12.ol_d_id = 11.ol_d_id
15      and 12.ol_delivery_d > 11.ol_delivery_d)
16  and su_nationkey = n_nationkey
17  and n_name = 'GERMANY'
18 group by su_name

```

```
order by numwait desc , su_name
```

Listing 3: Query Analítica 2

Durante a obtenção dos valores relativos ao tempo de execução desta interrogação reparamos que na primeira execução os valores obtidos foram cerca de 2 minutos e 17 segundos, um valor extremamente elevado! O plano de execução desta query, pode ser observado na figura abaixo. Este plano tem cerca de 19 passos.

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Sort (cost=1308408.51..1308433.51 rows=10000 width=34) (actual=29770.299, 32568.697 rows=396 loops=1)		0.309 ms	32568.697 ms	↑ 25.26	396	10000
2.	→ Aggregate (cost=1305050.14..1307744.12 rows=10000 width=34) (actual=29722.555, 32568.389 rows=396 loops=1)		0.394 ms	32568.389 ms	↑ 25.26	396	10000
3.	→ Gather Merge (cost=1305050.14..1307544.12 rows=20000 width=34) (actual=29722.314, 32567.996 rows=1188 loops=1)		-56411.89 ms	32567.996 ms	↑ 16.84	1188	20000
4.	→ Aggregate (cost=1304059.11..1304235.6 rows=10000 width=34) (actual=29598.014, 29659.962 rows=396 loops=3)		88.839 ms	88979.866 ms	↑ 25.26	396	10000
5.	→ Sort (cost=1304050.11..1304078.61 rows=1398 width=26) (actual=29597.901, 29630.349 rows=10560 loops=3)		2796.222 ms	88891.047 ms	↓ 9.27	105602	11398
6.	→ Nested Loop Anti Join (cost=430124.96..1302282.09 rows=11398 width=26) (actual=14333.386, 28698.275...		816.603 ms	86094.826 ms	↓ 9.27	105602	11398
7.	→ Nested Loop Inner Join (cost=430124.4..1157513.2 rows=17098 width=46) (actual=14333.242, 21988.2...		1040.116 ms	65964.648 ms	↓ 15.07	257514	17098
8.	→ Hash Inner Join (cost=430123.97..956601.07 rows=50330 width=50) (actual=14333.135, 19016.44...		24704.07 ms	57049.332 ms	↓ 5.22	262507	50330
9.	→ Hash Cond: ((1.ol_w_id = stock.s_w_id) AND (1.ol_u_id = stock.s_u_id))		13121.257 ms	13121.257 ms	↑ 1.26	6942984	8721613
10.	→ Seq Scan on public.order_line as I1 (cost=0.351835.13 rows=8721613 width=24) (actual=0.08...		389.797 ms	19224.006 ms	↓ 5.15	100800	19609
	→ Hash (cost=429829.83..429829.83 rows=19609 width=34) (actual=6407.995, 5408.002 rows=1...		Buckets: 65536 Batches: 8 Memory Usage: 3232 kB				
11.	→ Hash Inner Join (cost=430123.96..429829.83 rows=19609 width=34) (actual=1422.36, 6278.0...		2211.665 ms	18834.21 ms	↓ 5.15	100800	19609
12.	→ Hash Cond: (mod((stock.s_w_id * stock.s_u_id), 10000) = supplier.su_suppkey)		12356.35 ms	12356.35 ms	↑ 1.26	2666667	3333550
13.	→ Seq Scan on public.stock as stock (cost=0..408425.5 rows=3333550 width=8) (actual=...		0.555 ms	4266.195 ms	↓ 6.72	396	59
	→ Hash (cost=372.23..372.23 rows=59 width=30) (actual=1422.061, 1422.065 rows=39...		Buckets: 1024 Batches: 1 Memory Usage: 33 kB				
14.	→ Hash Inner Join (cost=12.14..372.23 rows=59 width=30) (actual=1408.992, 1421...		14.133 ms	4265.64 ms	↓ 6.72	396	59
15.	→ Hash Cond: (supplier.su_nationkey = nation.n_nationkey)		26.238 ms	26.238 ms	↑ 1	10000	10000
16.	→ Seq Scan on public.supplier as supplier (cost=0..322 rows=10000 width=34)...		0.039 ms	4225.269 ms	↑ 1	1	1
	→ Hash (cost=12.12..12.12 rows=1 width=4) (actual=1408.422, 1408.423 row...		Buckets: 1024 Batches: 1 Memory Usage: 9 kB				
17.	→ Seq Scan on public.nation as nation (cost=0..12.12 rows=1 width=4) (a...		4225.231 ms	4225.231 ms	↑ 1	1	1
	Filter: (nation.n_name = 'GERMANY') Rows Removed by Filter: 24						
18.	→ Index Scan using pk_orders on public.orders as orders (cost=0..43.398 rows=1 width=20) (actual=0...		7875.2 ms	7875.2 ms	↑ 1	1	1
	Filter: (1.ol_delivery_d > orders.o_entry_d)						
	Index Cond: ((orders.o_w_id = 11.ol_w_id) AND (orders.o_d_id = 11.ol_d_id) AND (orders.o_id = 11.ol_o_id))						
	Rows Removed by Filter: 0						
19.	→ Index Scan using pk_order_line on public.order_line as l2 (cost=0..56..11.37 rows=3 width=20) (actual=0...		19313.575 ms	19313.575 ms	↑ 3	1	3
	Filter: ((2.ol_delivery_d = 11.ol_delivery_d))						
	Index Cond: ((2.ol_w_id = 11.ol_w_id) AND (2.ol_d_id = 11.ol_d_id) AND (2.ol_o_id = 11.ol_o_id))						
	Rows Removed by Filter: 7						

Figura 43: Plano e Tempo de execução da Query Analítica 2 - Primeira execução.

No entanto após termos obtido um valor tão alto na execução da interrogação, decidimos correr novamente a interrogação tendo verificado que agora o tempo de execução era de apenas de 47 segundos, ou seja uma redução de mais de 50% do tempo face à primeira execução! Para além disso é possível observar que agora o plano tem 23 passos face aos 19 obtidos na primeira execução. O plano de execução pode ser observado na figura abaixo.

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Sort (cost=2146437.3..2146462.3 rows=10000 width=34) (actual=38399.471..41038.07 rows=396 loops=1)	0.437 ms	41038.07 ms	↑ 25.26	396	10000	1
2.	→ Aggregate (cost=2142588.68..2145772.91 rows=10000 width=34) (actual=38334.458..41037.634 rows=396 loops=1)	0.516 ms	41037.634 ms	↑ 25.26	396	10000	1
3.	→ Gather Merge (cost=2142588.68..214572.91 rows=20000 width=34) (actual=38334.174..41037.118 rows=1188 loops=1)	-73886.69 ms	41037.118 ms	↑ 16.84	1188	20000	1
4.	→ Aggregate (cost=2141588.66..2142264.39 rows=10000 width=34) (actual=38242.532..38307.936 rows=396 loops=1)	85.612 ms	114923.808 ms	↑ 25.26	396	10000	3
5.	→ Sort (cost=2141588.66..2141780.57 rows=76765 width=26) (actual=38242.599..38279.399 rows=105602 loops=1)	2804.733 ms	114883.197 ms	↓ 1.38	105602	76765	3
6.	→ Hash Anti Join (cost=1798375.98..213522.38 rows=76765 width=26) (actual=35550.39..37344.488 rows=105602 loops=1)	6945.745 ms	112033.464 ms	↓ 1.38	105602	76765	3
7.	→ Hash Cond: ((ol_o_id > ol_o_id) AND (ol_o_id <= ol_w_id) AND (ol_o_id <= ol_d_id))						
8.	→ Merge Inner Join (cost=1244248.06..1282564.73 rows=115148 width=46) (actual=22599.541..260.. rows=115148 loops=1)	3236.376 ms	78050.19 ms	↓ 2.24	257514	115148	3
9.	→ Hash Inner Join (cost=224465.37..793819.33 rows=346859 width=50) (actual=9628.234.. rows=346859 loops=1)	5546.707 ms	53139.014 ms	↑ 1.33	262507	346859	3
10.	→ Hash Inner Join (cost=224095.55..235784.23 rows=8671471 width=28) (actual=962.. rows=8671471 loops=1)	29953.444 ms	45760.116 ms	↑ 1.25	6942984	8671471	3
11.	→ Hash Cond: ((ol_w_id > stock_s_w_id) AND (ol_w_id <= stock_s_u_id))						
12.	→ Seq Scan on public.order_line as I1 (cost=0.351403.7 rows=8678470 width=24..)	8160.162 ms	8160.162 ms	↑ 1.25	6942984	8678470	3
13.	→ Hash (cost=161077.48..161077.48 rows=3333138 width=8) (actual=2548.668.. rows=2548.668 loops=1)	3959.032 ms	7646.01 ms	↑ 1.25	2666657	3333138	3
14.	→ Index Only Scan using pk_stock on public.stock as stock (cost=0.43..16107.. rows=1024.82..1024.82 loops=1)	3686.979 ms	3686.979 ms	↑ 1.25	2666657	3333138	3
15.	→ Hash (cost=364.82..364.82 rows=400 width=30) (actual=4.061..4.064 rows=396 loops=1)	0.433 ms	12.192 ms	↑ 1.02	396	400	3
16.	→ Hash Inner Join (cost=1.32..364.82 rows=400 width=30) (actual=0.17..3.92 rows=1)	7.166 ms	11.76 ms	↑ 1.02	396	400	3
17.	→ Hash Cond: (supplier_su.nationkey = nation.nationkey)						
18.	→ Seq Scan on public.supplier as supplier (cost=0..322 rows=10000 width=34..)	4.234 ms	4.234 ms	↑ 1	10000	10000	3
19.	→ Hash (cost=1.31..1.31 rows=1 width=4) (actual=0.119..0.12 rows=1 loops=1)	0.012 ms	0.36 ms	↑ 1	1	1	3
20.	→ Buckets: 1024 Batches: 1 Memory Usage: 9 kB						
21.	→ Seq Scan on public.nation as nation (cost=0..1.31 rows=1 width=4) (actual=0..1.31 rows=1 width=4) Filter: (nation.name = 'GERMANY') Rows Removed by Filter: 24	0.349 ms	0.349 ms	↑ 1	1	1	3
22.	→ Materialize (cost=406650.36..418864.72 rows=2442873 width=20) (actual=4997.693..6951.79.. rows=105602 loops=1)	2506.761 ms	20855.37 ms	↓ 1.01	2456555	2442873	3
23.	→ Sort (cost=406650.36..412757.54 rows=2442873 width=20) (actual=4997.684..6116.203.. rows=105602 loops=1)	16044.4 ms	18348.609 ms	↑ 1.01	2442860	2442873	3
24.	→ Seq Scan on public.orders as orders (cost=0..47261.73 rows=2442873 width=20) (actual=9012.509..9012.51 rows=69429.. loops=1)	2304.21 ms	2304.21 ms	↑ 1	2442873	2442873	3
25.	→ Hash (cost=351403.7..351403.7 rows=6578470 width=20) (actual=9012.509..9012.51 rows=69429.. loops=1)	14584.716 ms	27037.53 ms	↑ 1.25	6942984	8678470	3
26.	→ Hash Scan on public.order_line as I2 (cost=0..351403.7 rows=8678470 width=20) (actual=1674.. rows=105602 loops=1)	12452.814 ms	12452.814 ms	↑ 1.25	6942984	8678470	3

Figura 44: Plano e Tempo de execução da Query Analítica 2 - Segunda execução.

Como se pode observar pelo plano acima conseguimos perceber que a redução drástica do tempo deve-se ao facto do PostgreSQL ter decidido colocar a tabela *orders* em memória (materialização), ou invés de utilizar index scan (como fez no plano de execução 1) podendo depois aceder a esta várias vezes sem ter de a carregar novamente para memória. No plano de execução 1 desta query o PostgreSQL decidiu fazer um index scan sobre a tabela *orders* toda, o que possivelmente aconteceu foi que o PostgreSQL verificou que utilizando índices acabava por ter de percorrer a mesma página várias vezes e colocá-la em memória o que se refletia numa degradação da performance na execução da query, posto isto o PostgreSQL na execução seguinte decidiu então colocar diretamente a tabela toda em memória poupando assim a leitura de várias páginas várias vezes. Esta é a prova viva dos problemas expostos durante a UC de Administração de Bases de Dados de que a utilização de índices nem sempre é vantajosa.

Ao modificar o ficheiro de configuração do PostgreSQL e forçar o PostgreSQL a utilizar índices desativando o *sequential scan* verificamos que o tempo de execução é parecido com o valor da primeira execução, sendo neste caso de 2 minutos e 24 segundos, contendo 21 passos para o plano de execução. Este plano pode ser observado na figura abaixo.

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Sort (cost=20036272934.95..20036272959 95 rows=10000 width=34) (actual=143389.308..143389.343 rows=10000 width=34)	0.286 ms	143389.343 ms	↑ 25.26	396	10000	1
2.	→ Aggregate (cost=20036270788.79..20036272270 57 rows=10000 width=34) (actual=143301.597..143..)	34.373 ms	143389.057 ms	↑ 25.26	396	10000	1
3.	→ Sort (cost=20036270788.79..20036271249.38 rows=184237 width=26) (actual=143301.443..143..)	2064.517 ms	143354.685 ms	↓ 1.72	316806	184237	1
4.	→ Nested Loop Anti Join (cost=20031108697.22..20036250266.16 rows=184237 width=26) (actual=141290.168..141290.168 rows=184237 width=26)	552.791 ms	141290.168 ms	↓ 1.72	316806	184237	1
5.	→ Nested Loop Inner Join (cost=20031108696.65..20033924554.92 rows=276356 width=34..276356 width=34)	5632.409 ms	136874.662 ms	↓ 2.8	772543	276356	1
6.	→ Merge Inner Join (cost=10031108696.65..10033820919.96 rows=6908910 width=5..6908910 width=5)	5023.464 ms	131242.254 ms	↓ 2.96	20387388	6908910	1
7.	→ Index Scan using pk_orders on public.orders as orders (cost=0.43..2174673.94..2174673.94)	623.937 ms	623.937 ms	↑ 1	2442873	2442873	1
8.	→ Materialize (cost=10031108696.22..10031212753.87 rows=20811530 width=5..20811530 width=5)	2814.026 ms	125594.853 ms	↓ 1.01	20815476	20811530	1
9.	→ Sort (cost=10031108696.22..10031160725.05 rows=20811530 width=5..20811530 width=5)	79422.058 ms	122780.827 ms	↓ 1.01	20815476	20811530	1
10.	→ Merge Inner Join (cost=10025968569.14..10026444964.21 rows=20..20)	5147.089 ms	43358.769 ms	↓ 1.01	20815476	20811530	1
11.	→ Sort (cost=10001692865.51..10001712864.34 rows=7999532..7999532)	5900.87 ms	8729.734 ms	↑ 1.01	7994880	7999532	1
12.	→ Hash Inner Join (cost=10000000447.43..10000338180.81..10000338180.81)	1776.84 ms	2828.865 ms	↑ 1.01	7994880	7999532	1
13.	→ Index Only Scan using pk_stock on public.stock as stock (cost=10..10)	1048.165 ms	1048.165 ms	↓ 1.01	8000000	7999532	1
14.	→ Hash (cost=10000000322..10000000322 rows=10000..10000 Buckets: 16384 Batches: 1 Memory usage: 793 kB)	1.787 ms	3.86 ms	↑ 1	10000	10000	1
15.	→ Seq Scan on public.supplier as supplier (cost=10..10)	2.073 ms	2.073 ms	↑ 1	10000	10000	1
16.	→ Materialize (cost=24275703.63..24379845.27 rows=20828328..20828328)	2793.773 ms	29481.947 ms	↓ 1.01	20828951	20828328	1
17.	→ Sort (cost=24275703.63..24327774.45 rows=20828328 width=5..20828328 width=5)	21562.283 ms	26688.175 ms	↓ 1.01	20828951	20828328	1
18.	→ Index Scan using pk_order_line on public.order_line as line (cost=10..10)	5125.892 ms	5125.892 ms	↓ 1.01	20828951	20828328	1
19.	→ Materialize (cost=10000000000..10000000001.32 rows=1 width=4) (actual=0.0 rows=0)	-0.05 ms	0 ms	↑ 1	1	1	20387388
20.	→ Seq Scan on public.nation as nation (cost=10000000000..10000000001.31 rows=1 width=4) (actual=0.0 rows=0)	0.05 ms	0.05 ms	↑ 1	1	1	1
21.	→ Index Scan using pk_order_line on public.order_line as l2 (cost=0.56..11.2 rows=3 width=32) (actual=0.0 rows=0)	3862.715 ms	3862.715 ms	↑ 3	1	3	772543

Figura 45: Plano e Tempo de execução da Query Analítica 2 - Sequential Scan Off.

Como ponto de inicialização da otimização desta query vamos considerar então o caso em que temos o *sequential scan* a *off*, por forma a conseguirmos forçar o PostgreSQL a utilizar índices. Temos então objectivo de tentar melhorar o tempo de 2 minutos e 24 segundos de execução, que é considerado um tempo extremamente alto.

Tomemos como ponto de partida a visualização de alguns estatísticas relativas ao plano de execução, onde podemos visualizar uma série de estatísticas do plano de execução por nodo e por tabela, que serão úteis para tomar decisões sobre que melhorias fazer.

Statistics per Node Type

Node type	Count	Time spent	%% of query
Aggregate	1	34.373 ms	0.03%
Hash	1	1.787 ms	0.01%
Hash Inner Join	1	1776.84 ms	1.24%
Index Only Scan	1	1048.165 ms	0.74%
Index Scan	3	9612.544 ms	6.71%
Materialize	3	5607.749 ms	3.92%
Merge Inner Join	2	10170.553 ms	7.1%
Nested Loop Anti Join	1	552.791 ms	0.39%
Nested Loop Inner Join	1	5632.409 ms	3.93%
Seq Scan	2	2.123 ms	0.01%
Sort	5	108950.014 ms	75.99%

Figura 46: Estatísticas por tipo de nodo.

Statistics per Table

Table name	Scan count	Total time	%% of query
Node type	Count	Sum of times	%% of table
public.nation	1	0.05 ms	0.01%
Seq Scan	1	0.05 ms	100%
public.order_line	2	8988.607 ms	6.27%
Index Scan	2	8988.607 ms	100%
public.orders	1	623.937 ms	0.44%
Index Scan	1	623.937 ms	100%
public.stock	1	1048.165 ms	0.74%
Index Only Scan	1	1048.165 ms	100%
public.supplier	1	2.073 ms	0.01%
Seq Scan	1	2.073 ms	100%

Figura 47: Estatísticas por tabela.

As figuras acima foram conseguida através da utilização do pgAdmin, que se demonstrou bastante útil para perceber melhor onde podíamos atuar na resolução dos problemas com as interrogações. Na interface gráfica do pgAdmin fizemos a execução com o *EXPLAIN ANALYZE*,

tendo conseguido obter o tempo de execução de cada interrogação individual. Através da análise da figura relativa ao plano de execução, figura 45, é fácil de perceber que as zonas a vermelho são as zonas mais problemáticas e iremos tentar atuar através dessas zonas e tentar introduzir redundância a nível de índices e vistas materializadas.

4.2.1 Índices

Para otimização da interrogação analítica, foram analizados todos os índices que podiam ser criados e que ainda não existem. Como é possível ver na interrogação analítica alguns índices já estão a ser utilizados, como é o caso do índice *pk_orders*, *pk_order_line* e *pk_stock*. Estes índices são sobre a chave primária da tabela *orders*, *order_line* e *stock*, respetivamente.

A escolha dos índices foi simples, primeiro começamos a verificar as condições associadas ao *where*, e reparamos que no caso da tabela *supplier* era feito um *group by* pelo nome do *supplier* podendo ser criado um índice sobre isso. Além disso depois era feita uma verificação entre a *nation key* do *supplier* e a *nation key* obtida através das linhas filtradas por *GERMANY*, então criamos um índice que junta a *nation name* e *nation key*, para facilitar depois na obtenção de páginas que já tenham ambas as informações.

No caso do *supplier*, é feita uma verificação sobre a sua *key* e sobre a *nation key*, decidindo criar um índice para cada um destes.

No caso da tabela *order line*, o único índice que existe é o índice relativo à coluna *id* e uma vez que na cláusula *where* são feitas várias verificações sobre diversas colunas da tabela *order line*, mais do que uma vez, criou-se um índice único considerando essas colunas (*ol_o_id*, *ol_w_id*, *ol_d_id* e *ol_i_id*) e ainda outro índice extra sobre a data de entrega (*ol_delivery_d*) uma vez que no select mais aninhado é feita verificação por esta coluna.

No caso da tabela *orders*, verificamos que seria interessante criar um índice sobre as colunas *o_id*, *o_w_id* e *o_d_id* mas após consulta dos índices da tabela *orders* verificamos que este já existia. Como tal restou apenas criar o índice relativo à data de entrada da encomenda (*o_entry_d*).

Os índices criados foram então os seguintes:

```

1  create index i_name_supplier on public.supplier using btree (su_name);
2
3  create index i_key_supplier on public.supplier using btree (su_suppkey);
4
5  create index i_n_key_supplier on public.supplier using btree (su_nationkey);
6
7  create index i_key_name_nation on public.nation using btree (n_nationkey , n_name
   );
8
9  create index i_info_order_line on public.order_line using btree (ol_o_id ,
   ol_w_id , ol_d_id , ol_i_id);
```

```

10
11  create index i_delivery_d_order_line on public.order_line using btree (
12    ol_delivery_d);
13
14 # Indice ja existia previamente
15  create index i_info_orders on public.orders using btree (o_id, o_w_id, o_d_id);
16  create index i_entry_d_order on public.orders using btree (o_entry_d);

```

Listing 4: Índices criados

Após a criação dos índices corremos novamente a interrogação analítica tendo-se obtido o seguinte resultado:

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	Sort (cost=15011460.15..15011485.15 rows=10000 width=34) (actual=38379.433..38625.154 rows=396 loops=1)	0.292 ms	38625.154 ms	↑ 25.26	396	10000	1
2.	→ Aggregate (cost=15007595.59..15010795.77 rows=10000 width=34) (actual=38289.199..38624.863 rows=3...	0.469 ms	38624.863 ms	↑ 25.26	396	10000	1
3.	→ Gather Merge (cost=15007595.59..15010595.77 rows=20000 width=34) (actual=38289.947..38624.395...	-7627.76 ms	38624.395 ms	↑ 16.84	1188	20000	1
4.	→ Aggregate (cost=15006595.56..15007287.25 rows=10000 width=34) (actual=38219.723..38300.38...	93.564 ms	114901.155 ms	↑ 25.26	396	10000	3
5.	→ Sort (cost=15006595.56..15006792.79 rows=78891 width=26) (actual=38219.635..38269.197...	2766.922 ms	114807.591 ms	↓ 1.34	105602	78891	3
6.	→ Nested Loop Anti Join (cost=224431.59..14998288.74 rows=78891 width=26) (actual=22...	1230.79 ms	112040.67 ms	↓ 1.34	105602	78891	3
7.	→ Hash Inner Join (cost=224431.02..14000178.04 rows=118337 width=46) (actual=22...	5558.808 ms	90723.762 ms	↓ 2.18	257514	118337	3
8.	→ Hash Inner Join (cost=224170.42..13980243.88 rows=2958430 width=28) (actual=...	32533.61 ms	82022.295 ms	↓ 2.3	6800102	2958430	3
9.	→ Nested Loop Inner Join (cost=0.99..13668517.22 rows=2881759 width=28...)	8157.694 ms	41743.92 ms	↓ 2.36	6800102	2881759	3
10.	→ Index Scan using pk_orders on public.orders as orders (cost=0.43..21...	1828.878 ms	1828.878 ms	↑ 1.26	814291	1017864	3
11.	→ Index Scan using pk_order_line on public.order_line as l1 (cost=0.56..1...	31757.349 ms	31757.349 ms	↓ 2.67	8	3	2442873
12.	→ Hash (cost=161117.24..161117.24 rows=3334946 width=8) (actual=2581...	3927.711 ms	7744.765 ms	↑ 1.26	2666667	3334946	3
13.	Buckets: 1310172 Batches: 128 Memory Usage: 3520 kB						
14.	→ Index Only Scan using pk_stock on public.stock as stock (cost=0.43..1...	3817.054 ms	3817.054 ms	↑ 1.26	2666667	3334946	3
15.	→ Hash (cost=255.6..255.6 rows=400 width=30) (actual=1047.55..1047.553 rows...	0.483 ms	3142.66 ms	↑ 1.02	396	400	3
16.	Buckets: 1024 Batches: 1 Memory Usage: 33 kB						
17.	→ Nested Loop Inner Join (cost=11.52..255.6 rows=400 width=30) (actual=1...	0.261 ms	3142.177 ms	↑ 1.02	396	400	3
18.	→ Index Only Scan using l1_key_nation on public.nation as nation (...	0.141 ms	0.141 ms	↑ 1	1	1	3
19.	Index Cond: (nation.n_name = 'GERMANY'->char)						
20.	→ Bitmap Heap Scan on public.supplier as supplier (cost=11.38..243.27...	3141.262 ms	3141.775 ms	↑ 1.02	396	400	3
21.	→ Bitmap Heap Scan on public.supplier as supplier (cost=11.38..243.27...	3141.262 ms	3141.775 ms	↑ 1.02	396	400	3
22.	→ Bitmap Index Scan using l1n_key_supplier (cost=0..11.29 rows=4...	0.513 ms	0.513 ms	↑ 1.02	396	400	3
23.	Index Cond: (supplier.supplierkey = nation.n_nationkey)						
24.	→ Index Scan using pk_order_line on public.order_line as l2 (cost=0.56..11.23 rows=3 w...	20086.118 ms	20086.118 ms	↑ 3	1	3	772543
25.	Filter: (l2.ol_delivery_d < l1.ol_delivery_d)						
26.	Index Cond: (l2.ol_w_id = l1.ol_w_id) AND (l2.ol_d_id = l1.ol_d_id) AND (l2.ol_o_id = l1.ol_o_i...						
27.	Rows Removed by Filter: 7						

Figura 48: Plano da Query Analítica 2 com índices.

```

tpcc/Joel@server-joel ~
Messages
Successfully run. Total query runtime: 40 secs 615 msec.
1 rows affected.

```

Figura 49: Tempo de execução da Query Analítica 2 com índices.

Após avaliação dos resultados da execução observa-se uma redução no tempo de execução enorme! Garantindo que a introdução de redundância ao nível dos índices foi bem conseguida para otimizar a query, uma vez que se observou uma redução de aproximadamente 73% face ao tempo de execução inicial.

Através da observação das estatísticas por nodo e por tabela, figura 50 e 51 respetivamente, conseguimos observar que já não existem referências a *sequential scans* sendo praticamente todos os scans efetuados através de índices.

Statistics per Node Type

Node type	Count	Time spent	% of query
Aggregate	2	94.033 ms	0.25%
Bitmap Heap Scan	1	3141.262 ms	8.14%
Bitmap Index Scan	1	0.513 ms	0.01%
Gather Merge	1	-76276.76 ms	-197.47%
Hash	2	3928.194 ms	10.18%
Hash Inner Join	2	38092.418 ms	98.63%
Index Only Scan	2	3817.195 ms	9.89%
Index Scan	3	53672.345 ms	138.96%
Nested Loop Anti Join	1	1230.79 ms	3.19%
Nested Loop Inner Join	2	8157.956 ms	21.13%
Sort	2	2767.214 ms	7.17%

Figura 50: Estatísticas por nodo da execução da Query 2 com índices.

Statistics per Table

Table name	Scan count	Total time	%% of query
Node type	Count	Sum of times	%% of table
public.nation	1	0.141 ms	0.01%
Index Only Scan	1	0.141 ms	100%
public.order_line	2	51843.467 ms	134.23%
Index Scan	2	51843.467 ms	100%
public.orders	1	1828.878 ms	4.74%
Index Scan	1	1828.878 ms	100%
public.stock	1	3817.054 ms	9.89%
Index Only Scan	1	3817.054 ms	100%
public.supplier	1	3141.262 ms	8.14%
Bitmap Heap Scan	1	3141.262 ms	100%

Figura 51: Estatísticas por tabela da execução da Query 2 com índices.

Através da análise das estatísticas por tabela é possível verificar que existe um tempo bastante elevado no *index scan* na tabela *order_line* e que está a ser usada a tabela na totalidade e que esta operação representa 134.23% da query, sendo por isso extremamente tentar criar vistas materializadas para diminuir ainda mais o tempo de execução.

4.2.2 Vistas Materializadas

Tal como exposto anteriormente o nosso foco será otimizar o tempo de execução do *index scan* na tabela *order line*. Para determinar as causas para um tempo tão elevado foi necessário avaliar o plano de execução em detalhe (figura 48). Após avaliar o mesmo foi possível perceber que o *index scan* é executado sobre essa tabela mais do que 3 milhões de vezes no total!

Através da análise da utilização da tabela *order line* na interrogação analítica verificamos que esta era usada para dar *match* com a tabela *order* e *supplier*, usando as colunas id, item, warehouse, data de encomenda e data de entrega do item. Embora a tabela *order line* seja habitualmente atualizada com frequência sempre que exista uma encomenda, a data de entrega de um determinado item após este ter sido inserido na respetiva tabela é normalmente superior à data em que esta linha de encomenda foi introduzida na tabela (assumindo que seja igual à data da encomenda). Pelo que caso se considere um *refresh* da nossa vista materializada igual

à média do número de dias entre a data de encomenda e a data de entrega, conseguimos de certa forma garantir que a vista estará mais ou menos atualizada com as linhas de encomenda que tenham ultrapassado o prazo de entrega.

Considerando isto será extremamente útil recorrer à utilização da redundância ao nível de vistas materializadas e como tal foi criada a seguinte vista materializada:

```

1  create materialized view m_view_order_line as (
2      select ol_o_id, ol_w_id, ol_d_id, ol_delivery_d, mod((s_w_id * s_i_id)
3          ,10000) k
4      from order_line, orders, stock
5      where ol_o_id = o_id
6          and ol_w_id = o_w_id
7          and ol_d_id = o_d_id
8          and ol_w_id = s_w_id
9          and ol_i_id = s_i_id
10         and ol_delivery_d > o_entry_d
11 );

```

Listing 5: Vista Materializada

Posto isto a interrogação ficou da seguinte forma:

```

1  select su_name, count(*) as numwait
2  from supplier, (select * from m_view_order_line) l1, nation
3  where l1.k = su_suppkey
4  and not exists (select *
5      from order_line l2
6      where l2.ol_o_id = l1.ol_o_id
7          and l2.ol_w_id = l1.ol_w_id
8          and l2.ol_d_id = l1.ol_d_id
9          and l2.ol_delivery_d > l1.ol_delivery_d)
10     and su_nationkey = n_nationkey
11     and n_name = 'GERMANY'
12 group by su_name
13 order by numwait desc, su_name

```

Listing 6: Interrogação Analítica com Vista Materializada

Após correr novamente a query, tal como esperado vemos uma redução gigante no tempo de execução, passando dos 40 segundos obtidos anteriormente para cerca de 16 segundos! Tivemos uma redução de mais de 50% face ao teste anterior. O plano obtido está então na figura seguinte, verificando que também o número de passos foi reduzido para apenas 12 passos.

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Sort (cost=10001146576.86..10001146601.86 rows=10000 width=34) (actual... Rows Removed by Filter: 5)	0.272 ms	15793.431 ms	↑ 25.26	396	10000	1
2.	→ Aggregate (cost=10001141732.39..10001145912.47 rows=10000 width=34) Hash Cond: (m_view_order.line.o_l_id = m_view_order_line.o_l_id)	33.488 ms	15793.16 ms	↑ 25.26	396	10000	1
3.	→ Sort (cost=10001141732.39..10001143092.42 rows=544011 width=34) Hash Cond: (m_view_order.line.o_l_id = m_view_order_line.o_l_id)	931.14 ms	15759.672 ms	↑ 1.72	316806	544011	1
4.	→ Nested Loop Anti Join (cost=10000000261.16..10001076889.97 rows=10000 width=34) Hash Cond: (m_view_order.line.o_l_id = m_view_order_line.o_l_id)	168.969 ms	14828.533 ms	↑ 1.72	316806	544011	1
5.	→ Hash Inner Join (cost=10000000260.6..10000438929.37 rows=10000 width=34) Hash Cond: (m_view_order.line.k = supplier.su_suppkey)	2171.23 ms	4616.505 ms	↑ 1.06	772543	816016	1
6.	→ Seq Scan on public.m_view_order_line as m_view_ord...	2204.325 ms	2204.325 ms	↑ 1.01	20400306	20400408	1
7.	→ Hash (cost=255.6..255.6 rows=400 width=30) (actual... Buckets: 1024 Batches: 1 Memory Usage: 33 kB)	0.079 ms	240.951 ms	↑ 1.02	396	400	1
8.	→ Nested Loop Inner Join (cost=11.52..255.6 rows=400) Index Cond: (nation.n_name = 'GERMANY') Heap Blocks: exact=191	0.062 ms	240.872 ms	↑ 1.02	396	400	1
9.	→ Index Only Scan using i_key_name_nation on nation Index Cond: (nation.n_name = 'GERMANY') Heap Blocks: exact=191	0.014 ms	0.014 ms	↑ 1	1	1	1
10.	→ Bitmap Heap Scan on public.supplier as supplier Recheck Cond: (supplier.su_nationkey = nation.n_nationkey) Heap Blocks: exact=191	240.725 ms	240.797 ms	↑ 1.02	396	400	1
11.	→ Bitmap Index Scan using i_n_key_supplier on supplier Index Cond: (supplier.su_nationkey = nation.n_nationkey)	0.072 ms	0.072 ms	↑ 1.02	396	400	1
12.	→ Index Scan using i_info_order_line on public.order_line as ol Filter: (ol.delivery_d > m_view_order_line.ol_delivery_d) Index Cond: ((2.ol_o_id = m_view_order_line.ol_o_id) AND (2.ol_w_id = m_view_order_line.ol_w_id) AND (2.ol_d_id = m_view_order_line.ol_d_id)) Rows Removed by Filter: 5	10043.059 ms	10043.059 ms	↑ 3	1	3	772543

Figura 52: Plano da Query Analítica 2 com índices e vista materializada.



Messages

Successfully run. Total query runtime: 16 secs 239 msec.
1 rows affected.

Figura 53: Tempo de execução da Query Analítica 2 com índices e vista materializada.

As estatísticas obtidas foram as seguintes:

Statistics per Node Type

Node type	Count	Time spent	%% of query
Aggregate	1	33.488 ms	0.22%
Bitmap Heap Scan	1	240.725 ms	1.53%
Bitmap Index Scan	1	0.072 ms	0.01%
Hash	1	0.079 ms	0.01%
Hash Inner Join	1	2171.23 ms	13.75%
Index Only Scan	1	0.014 ms	0.01%
Index Scan	1	10043.059 ms	63.6%
Nested Loop Anti Join	1	168.969 ms	1.07%
Nested Loop Inner Join	1	0.062 ms	0.01%
Seq Scan	1	2204.325 ms	13.96%
Sort	2	931.412 ms	5.9%

Figura 54: Estatísticas por nodo da execução da Query 2 com índices e vista materializada.

Statistics per Table

Table name	Scan count	Total time	%% of query
Node type	Count	Sum of times	%% of table
public.m_view_order_line	1	2204.325 ms	13.96%
Seq Scan	1	2204.325 ms	100%
public.nation	1	0.014 ms	0.01%
Index Only Scan	1	0.014 ms	100%
public.order_line	1	10043.059 ms	63.6%
Index Scan	1	10043.059 ms	100%
public.supplier	1	240.725 ms	1.53%
Bitmap Heap Scan	1	240.725 ms	100%

Figura 55: Estatísticas por tabela da execução da Query 2 com índices e vista materializada.

Como podemos ver pela figura 55, neste momento temos um *sequential scan* sobre a nossa vista materializada e reparamos também que o tempo usado na tabela *order line* foi reduzido em praticamente 5x.

Apenas como observação final decidimos voltar a ativar o *sequential scan*, podendo assim o PostgreSQL optar se utiliza os índices criados por nós ou não e os resultados obtidos foram os seguintes:

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Sort (cost=542446.68..542471.68 rows=10000 width=34) (actual=11817.958..11818.928 rows=396 lo...	0.46 ms	11818.928 ms	↑ 25.26	396	10000	1
2.	→ Aggregate (cost=539248.8..541782.3 rows=10000 width=34) (actual=11805.577..11818.468 row...	0.493 ms	11818.468 ms	↑ 25.26	396	10000	1
3.	→ Gather Merge (cost=539248.8..541582.3 rows=20000 width=34) (actual=11805.528..11817...	-23485.271 ms	11817.976 ms	↑ 16.84	1188	20000	1
4.	→ Sort (cost=538248.78..538273.78 rows=10000 width=34) (actual=11767.679..11767.74...	8.1 ms	35303.247 ms	↑ 25.26	396	10000	3
5.	→ Aggregate (cost=537484.39..537584.39 rows=10000 width=34) (actual=11764.86...	412.466 ms	35295.148 ms	↑ 25.26	396	10000	3
6.	→ Nested Loop Anti Join (cost=254.14..536351.04 rows=226671 width=26) (actu...	964.269 ms	34882.683 ms	↑ 2.15	105602	226671	3
7.	→ Hash Inner Join (cost=253.58..270533.99 rows=340007 width=46) (actu...	5805.174 ms	13059.753 ms	↑ 1.33	257514	340007	3
8.	→ Seq Scan on public.m_view_order_line as m_view_order_line (cost=0...	5273.841 ms	5273.841 ms	↑ 1.26	6800102	8500170	3
9.	→ Hash (cost=248.58..248.58 rows=400 width=30) (actual=660.243..6...	0.418 ms	1980.738 ms	↑ 1.02	396	400	3
10.	Buckets: 1024 Batches: 1 Memory Usage: 33 kB						
11.	→ Nested Loop Inner Join (cost=11.38..248.58 rows=400 width=3...	0.24 ms	1980.321 ms	↑ 1.02	396	400	3
12.	→ Seq Scan on public.nation as nation (cost=0..1.31 rows=1 ... Filter: (nation.n_name = 'GERMANY') Rows Removed by Filter: 24	1977.279 ms	1977.279 ms	↑ 1	1	1	3
13.	→ Bitmap Heap Scan on public.supplier as supplier (cost=11... Recheck Cond: (supplier.su_nationkey = nation.n_nationkey) Heap Block: exact=191	2.541 ms	2.802 ms	↑ 1.02	396	400	3
14.	→ Bitmap Index Scan using L_n_key_supplier (cost=0..11... Index Cond: (supplier.su_nationkey = nation.n_nationkey) → Index Scan using L_info_order_line on public.order_line as l2 (cost=0..56.... Filter: ((2.ol_delivery_d > m_view_order_line.ol_delivery_d) Index Cond: ((2.ol_o_id = m_view_order_line.ol_o_id) AND ((2.ol_w_id = m_view_order_line.ol_w_id) AND (2.ol_d_id = m_view_order_line.ol_d_id))) Rows Removed by Filter: 5	20858.661 ms	20858.661 ms	↑ 3	1	3	772543

Figura 56: Plano da Query Analítica 2 com índices, vista materializada e com sequential scan a on.



Messages

Successfully run. Total query runtime: 12 secs 467 msec.
1 rows affected.

Figura 57: Tempo de execução da Query Analítica 2 com índices, vista materializada e com sequential scan a on.

Statistics per Node Type				Statistics per Table			
Node type	Count	Time spent	% of query	Table name	Scan count	Total time	% of query
Aggregate	2	412.959 ms	3.5%	public.m_view_order_line	1	5273.841 ms	44.63%
Bitmap Heap Scan	1	2.541 ms	0.03%		Seq Scan	5273.841 ms	100%
Bitmap Index Scan	1	0.261 ms	0.01%	public.nation	1	1977.279 ms	16.73%
Gather Merge	1	-23485.271 ms	-198.7%		Seq Scan	1977.279 ms	100%
Hash	1	0.418 ms	0.01%	public.order_line	1	20858.661 ms	176.49%
Hash Inner Join	1	5805.174 ms	49.12%		Index Scan	20858.661 ms	100%
Index Scan	1	20858.661 ms	176.49%	public.supplier	1	2.541 ms	0.03%
Nested Loop Anti Join	1	964.269 ms	8.16%		Bitmap Heap Scan	2.541 ms	100%
Nested Loop Inner Join	1	0.24 ms	0.01%				
Seq Scan	2	7251.121 ms	61.36%				
Sort	2	8.56 ms	0.08%				

Figura 58: Estatísticas da execução da Query 2 com índices, vista materializada e com sequential scan a on.

Com o *sequential scan a on*, conseguimos verificar que ainda obtemos uma melhoria mais significativa, conseguindo executar a interrogação em cerca de 12 segundos apenas.

Como face ao tempo inicial de 2 minutos e 24 segundos se conseguiu obter uma melhoria de 89%, consideramos por terminada a otimização desta query. No entanto ainda seria possível otimizar a mesma introduzindo mais redundância, como por exemplo pela introdução de índices na vista materializada, evitando assim o *sequential scan*.

4.3 Query Analítica 3

A query analítica 3 seleccionada determina todos os fornecedores de um país específico que seleccionaram peças que podem ser candidatadas a uma oferta promocional se a quantidade desses itens for superior à quantidade total que foi solicitada desde uma determinada data.

```

1 select su_name, su_address
2   from supplier, nation
3 where su_suppkey in
4   (select mod(s_i_id * s_w_id, 10000)
5    from stock, order_line
6   where s_i_id in
7     (select i_id
8      from item
9       where i_data like 'co%')
10    and ol_i_id=s_i_id
11    and ol_delivery_d > '2010-05-23 12:00:00'
12   group by s_i_id, s_w_id, s_quantity
13   having 10*s_quantity > sum(ol_quantity))
14  and su_nationkey = n_nationkey
15  and n_name = 'GERMANY'
16 order by su_name

```

Listing 7: Query Analítica 3

Depois de efectuar o *EXPLAIN ANALYSE* para a *query 3*, foi obtido um tempo de execução de 0.766 s (766.806 ms) como se pode ver na imagem 59, o que é bastante pouco para fazer uma optimização desse mesmo tempo neste trabalho prático.

```
Planning Time: 1.826 ms
Execution Time: 766.806 ms
(47 rows)
```

Figura 59: Tempo de execução da Query Analítica 3 sem qualquer alteração.

Para aumentar o respectivo tempo de execução, foi decidido alterar os vários parâmetros da *query*. Após modificar os valores nas zonas "n_name = 'GERMANY'", "and ol_delivery_d > '2010-05-23 12:00:00'" e "i_data like 'co%'" houve um elevado aumento no tempo de execução de 0,766 s para 27,697 s (27697.012 ms) como se pode verificar no plano de execução na imagem 60.

```
1 select su_name, su_address
2   from supplier, nation
3  where su_suppkey in
4    (select mod(s_i_id * s_w_id, 10000)
5     from stock, order_line
6    where s_i_id in
7      (select i_id
8        from item
9       where i_data like 'c%')
10      and ol_i_id=s_i_id
11      and ol_delivery_d > '2010-05-23 12:00:00'
12    group by s_i_id, s_w_id, s_quantity
13    having 10*s_quantity > sum(ol_quantity))
14   and su_nationkey = n_nationkey
15   and n_name = 'JAPAN'
16  order by su_name
```

Listing 8: Query Analítica 3 alterada

```

-----  

QUERY PLAN  

-----  

Sort (cost=3299573.53..3299574.03 rows=200 width=51) (actual time=27603.301..27644.323 rows=61 loops=1)
  Sort Key: supplier.su_name
  Sort Method: quicksort  Memory: 32kB
-> Hash Join (cost=3299199.12..3299565.89 rows=200 width=51) (actual time=27597.734..27644.197 rows=61 loops=1)
  Hash Cond: (supplier.su_suppkey = ANY_subquery .mod)
    -> Hash Join (cost=3299199.12..3299565.89 rows=200 width=51) (actual time=613.185..618.716 rows=377 loops=1)
      Hash Cond: (supplier.su_nationkey = ANY_subquery .nationkey)
        -> Seq Scan on supplier (cost=0.00..322.00 rows=10000 width=59) (actual time=0.031..3.921 rows=10000 loops=1)
          Buckets: 1024  Batches: 1  Memory Usage: 9KB
            -> Seq Scan on nation (cost=0.00..1.31 rows=1 width=4) (actual time=613.119..613.120 rows=1 loops=1)
              Filter: (n_name = ANY_subquery .n_name)
              Rows Removed by Filter: 24
-> Hash (cost=3299195.29..3299195.29 rows=200 width=4) (actual time=26984.249..27025.262 rows=1878 loops=1)
  Buckets: 2048  Originally 1  Batches: 1 (originally 1)  Memory Usage: 83KB
-> HashAggregate (cost=3299193.29..3299195.29 rows=200 width=4) (actual time=26983.273..27024.789 rows=1878 loops=1)
  Group Key: "ANY_subquery ".mod
-> Subquery Seq Scan on ANY_subquery  (cost=2936606.37..3298526.77 rows=266607 width=4) (actual time=21090.963..27021.472 rows=2240 loops=1)
  Group Key: stock.s_i_id, stock.s_w_id
  Filter: ((10 * stock.s_quantity) > sum(order_line.ol_quantity))
  Rows Removed by Filter: 113840
-> Gather Merge (cost=2936606.37..3270533.04 rows=1599642 width=20) (actual time=21088.866..26929.737 rows=348240 loops=1)
  Workers Launched: 2
-> PartialGroupAggregate (cost=2935606.35..3084894.64 rows=799821 width=20) (actual time=20914.582..26096.170 rows=116080 loops=3)
  Group Key: stock.s_i_id, stock.s_w_id
-> Sort (cost=2935606.35..2970928.87 rows=14129008 width=16) (actual time=20914.511..24253.732 rows=7888027 loops=3)
  Sort Key: stock.s_i_id, stock.s_w_id
  Sort Method: external merge  Disk: 199072KB
  Worker 0: Sort Method: external merge  Disk: 196760KB
  Worker 1: Sort Method: external merge  Disk: 206368KB
-> Parallel Hash Join (cost=119084.96..774704.28 rows=14129008 width=16) (actual time=742.903..7422.693 rows=7888027 loops=3)
  Hash Cond: (order_line.ol_i_id = stock.s_i_id)
-> Parallel Seq Scan on order_line (cost=0.00..384280.69 rows=8498720 width=8) (actual time=2.083..2235.105 rows=6800102 loops=3)
  Filter: (ol_delivery_d ~ '2010-05-23 12:00:00'::timestamp without time zone)
  Rows Removed by Filter: 32174
-> Parallel Hash (cost=118243.48..118243.48 rows=67318 width=16) (actual time=739.575..739.578 rows=38693 loops=3)
  Buckets: 262144  Batches: 1  Memory Usage: 7520KB
-> Nested Loop (cost=0.72..118243.48 rows=67318 width=16) (actual time=459.531..569.154 rows=38693 loops=3)
  -> Parallel Hash using pk_item on item (cost=0.29..3667.13 rows=842 width=4) (actual time=459.413..481.703 rows=484 loops=3)
  Filter: (i_data = ANY_subquery .n_name)
  Rows Removed by Filter: 32850
-> Index Scan using ix_stock on stock (cost=0.43..133.38 rows=270 width=12) (actual time=0.041..0.153 rows=80 loops=1451)
  Index Cond: (s_i_id = item.i_id)

Planning Time: 2.008 ms
 JIT:
  Functions: 99
  Options: Inlining true, Optimization true, Expressions true, Deforming true
  Timing: Generation 21.607 ms, Inlining 506.108 ms, Optimization 934.626 ms, Emission 548.330 ms, Total 2010.672 ms
Execution Time: 27697.012 ms
(51 rows)

```

Figura 60: Plano e Tempo de execução da Query Analítica 3 com novos parâmetros.

4.3.1 Índices

Para otimização da interrogação analítica, foram analizados possíveis índices que poderiam ser criados. Através da ferramenta *PgAdmin* foi possível verificar que apesar de haver índices pré-existentes a serem utilizados nas tabelas *stock* e *item* existem vários *Sequential Scans* ao que criação de índices sobre estes podem melhorar a optimizar a interrogação analítica.

Um dos índices que pareceu importante ser criado foi relativamente a *ol_delivery_d* sobre a tabela *order_line* em que o seu *sequential scan* apresenta um tempo total de 9.86 s (9862.728 ms), como se pode reparar na figura 61. Para além disso foram também criados índices para as tabelas *supplier* e *nation* às quais existe um *Seq Scan* nas mesmas.

```

1 CREATE INDEX idx_ol_delivery_d ON order_line(ol_delivery_d);
2 CREATE INDEX idx_su_nationkey ON supplier(su_nationkey);
3 CREATE INDEX idx_n_name ON nation(n_name);

```

Listing 9: Criação de índices

Table name	Scan count	Total time	% of query
Node type	Count	Sum of times	% of table
item	1	104.307 ms	0.39%
Index Scan	1	104.307 ms	100%
nation	1	581.721 ms	2.17%
Seq Scan	1	581.721 ms	100%
order_line	1	9862.728 ms	36.8%
Seq Scan	1	9862.728 ms	100%
stock	1	371.456 ms	1.39%
Index Scan	1	371.456 ms	100%
supplier	1	2.664 ms	0.01%
Seq Scan	1	2.664 ms	100%

Figura 61: Tabela sobre os Scans do plano de execução obtido através do PgAdmin.

Alguns dos índices criados não foram utilizados como se pode averiguar na figura 62. Estes índices foram *idx_n_name* e *idx.ol_delivery_d*, apesar do índice *idx_su_nationkey* ter sido utilizado não houve grandes alterações no tempo de execução.

```

QUERY_PLAN
Sort (cost=1951059.85..1951060.35 rows=200 width=51) (actual_time=27997.891..28029.654 rows=61 loops=1)
  Sort Key: supplier.su_name
  Sort Method: quicksort Memory: 32K
-> Hash Join (cost=1951059.74..1951052.21 rows=200 width=51) (actual_time=27994.895..28029.527 rows=61 loops=1)
    Hash Cond: (supplier.su_supkey = "ANY_subquery".mod)
      >> Nested Loop (cost=11.38..248.58 rows=400 width=55) (actual_time=545.387..552.979 rows=377 loops=1)
        >>> Seq Scan on nation (cost=0.00..1.31 rows=1 width=4) (actual_time=545.229..545.236 rows=1 loops=1)
          Filter: (n_name = 'JAPAN'::bpchar)
          Rows Removed by Filter: 2
        >>> Bitmap Scan on supplier (cost=11.38..243.27 rows=400 width=59) (actual_time=0.142..7.639 rows=377 loops=1)
          Recheck Cond: (su_nationkey = nation.n_nationkey)
          Heap Blocks: exact=179
          >>> Bitmap Index Scan on idx_su_nationkey (cost=0.00..11.29 rows=400 width=0) (actual_time=0.082..0.082 rows=377 loops=1)
-> Hash (cost=1951059.85..1951057.85 rows=200 width=4) (actual_time=27444.599..27476.353 rows=1878 loops=1)
  Buckets: 2048 (originally 1024) (actual_time=27444.599..27476.353 rows=200 width=4) (actual_time=27443.650..27475.904 rows=1878 loops=1)
  >>> HashAggregate (cost=1950795.85..1950797.85 rows=200 width=4) (actual_time=27443.650..27475.904 rows=1878 loops=1)
    Group Key: "ANY_subquery".mod
    >>> Subquery Scan on "ANY_subquery" (cost=1658127.93..1950129.11 rows=266698 width=4) (actual_time=21520.506..27472.628 rows=2240 loops=1)
      >>> Finalize GroupAggregate (cost=1658127.93..1947462.13 rows=266698 width=16) (actual_time=21520.503..27471.742 rows=2240 loops=1)
      >>> Bitmap Key Scan on item (cost=0.00..1.31 rows=1 width=16)
      >>> Filter: ((stock.s_quantity > sum(order_line.ol_quantity)))
      >>> Rows Removed by Filter: 113840
      >>> Gather Merge (cost=1658127.93..1922125.85 rows=1600186 width=20) (actual_time=21519.656..27360.650 rows=348240 loops=1)
        workers Planned: 2
        workers Actually: 2
        >>> Partial Aggregate (cost=1657127.91..1736424.66 rows=800093 width=20) (actual_time=21387.239..26614.430 rows=116080 loops=3)
          Group Key: stock.s_i_id, stock.s_w_id
          >>> Sort (cost=1657127.91..1674951.86 rows=129582 width=16) (actual_time=21387.163..24710.182 rows=888027 loops=3)
            Sort Key: stock.s_i_id, stock.s_w_id
            Sort Method: external merge Disk: 197200K
            worker 0: Sort Method: external merge Disk: 197880K
            worker 1: Sort Method: external merge Disk: 207128K
            >>> Parallel Hash Join (cost=72185.04..601898.47 rows=7129582 width=16) (actual_time=728.012..7498.351 rows=788027 loops=3)
              Hash Cond: (order_line.ol_i_id = stock.s_i_id)
              >>> Parallel Seq Scan on order_line (cost=0.00..384269.12 rows=8417325 width=16) (actual_time=0.056..2264.539 rows=6800102 loops=3)
                Filter: (ol_delivery_d > '2010-05-23 12:00:00'::timestamp without time zone)
              >>> Parallel Hash (cost=1764.16..71764.16 rows=33670 width=16) (actual_time=725.101..725.103 rows=38693 loops=3)
                Buckets: 131072 Batches: 1 Memory Usage: 6528K
                >>> Nested Loop (cost=0.72..71764.16 rows=33670 width=16) (actual_time=457.972..572.788 rows=38693 loops=3)
                  >>> Parallel Index Scan using pk_item on item (cost=0.29..3667.13 rows=421 width=4) (actual_time=457.845..477.690 rows=484 loops=3)
                  >>> Filter: (i_data ~* 'CX'::text)
                  >>> Rows Removed by Filter: 32850
                  >>> Index Scan using IX_stock on stock (cost=0.43..159.04 rows=271 width=12) (actual_time=0.035..0.182 rows=80 loops=1451)
                    Index Cond: (s_i_id = item.i_id)
Planning Time: 2.142 ms
 JIT:
  Functions: 94
  Options: Inlining true, Optimization true, Expressions true, Deforming true
  Timing: Generation 21.099 ms, Inlining 452.420 ms, Optimization 933.303 ms, Emission 530.757 ms, Total 1937.607 ms
Execution Time: 28081.740 ms
(52 rows)

```

Figura 62: Plano e Tempo de execução da Query Analítica 3 com novos índices.

Numa tentativa de forçar o motor a utilizar os outros índices criados, foi alterado no ficheiro *postgresql.conf* o uso de *seqscan* para *off*. Apesar de resultar e o motor utilizar os

outros índices criados anteriormente, o tempo de execução subiu para 29.73 s como se pode verificar na figura 63.

```

Sort (Cost=2328797.72..2328798.22 rows=200 width=51) (actual time=29618.463..29663.588 rows=61 loops=1)
  Sort Key: supplier.su_name
  Sort Method: quicksort Memory: 32kB
-> Hash Join (cost=2328542.90..2328790.07 rows=200 width=51) (actual time=29616.987..29663.461 rows=61 loops=1)
    Hash Cond: (supplier.su_name = ANY(supplier.su_name))
-> Nested Loop (cost=11.52..255.42 rows=400 width=55) (actual time=536.924..538.248 rows=377 loops=1)
    > Index Scan using idx_n_name on nation (cost=0.14..8.15 rows=1 width=4) (actual time=0.039..0.043 rows=1 loops=1)
      Index Cond: (n_name = 'JAPAN'::bpchar)
    > Bitmap Heap Scan on supplier (cost=11.38..243.27 rows=400 width=59) (actual time=536.874..538.125 rows=377 loops=1)
      Bitmap Heap Scan on nationkey = nation.n_nationkey
      Recheck Cond: nationkey = nation.n_nationkey
      Heap Blocks: exact=179
      > Bitmap Index Scan on idx_su_nationkey (cost=0.00..11.29 rows=400 width=0) (actual time=0.095..0.096 rows=377 loops=1)
        Index Cond: (su_nationkey = nation.n_nationkey)
-> Hash (cost=2328528.88..2328528.88 rows=200 width=4) (actual time=29079.974..29125.090 rows=1878 loops=1)
  Buckets: 1 (originally 1)  Memory Usage: 83kB
-> HashAggregate (cost=2328526.88..2328528.88 rows=200 width=4) (actual time=29079.010..29124.627 rows=1878 loops=1)
  Group Key: "ANY_subquery".mod
-> Subquery Scan on "ANY_subquery" (cost=2035858.95..2327860.13 rows=266698 width=4) (actual time=23079.136..29121.141 rows=2240 loops=1)
  > Finalize GroupAggregate (cost=2035858.95..2325193.15 rows=266698 width=16) (actual time=23079.133..29120.179 rows=2240 loops=1)
  Group Key: stock.s_i_id, stock.s_w_id
  Filter: (sum(order_line.ol_quantity) > sum(order_line.ol_quantity))
  Rows Removed by Filter: 113840
  > Gather Merge (cost=2035858.95..2299856.87 rows=1600186 width=20) (actual time=23078.034..29023.133 rows=348240 loops=1)
    Workers Planned: 2
    Workers Launched: 2
-> Partial GroupAggregate (cost=2034858.93..2114155.68 rows=800093 width=20) (actual time=22878.206..28245.255 rows=116080 loops=3)
  Group Key: stock.s_i_id, stock.s_w_id
  > Sort (cost=2034858.93..2052682.88 rows=7129582 width=16) (actual time=22878.120..26278.809 rows=7888027 loops=3)
    Sort Key: stock.s_i_id, stock.s_w_id
    Sort Method: external merge Disk: 213848kB
    Worker 0: Sort Method: external merge Disk: 194672kB
    Worker 1: Sort Method: external merge Disk: 193600kB
    > Parallel Hash Join (cost=450639.85..979629.49 rows=7129582 width=16) (actual time=1832.497..9271.952 rows=7888027 loops=3)
      Hash cond: (order_line.ol_i_id = stock.s_i_id)
      > Parallel Bitmap Heap Scan on order_line (cost=378454.81..762000.15 rows=8417325 width=8) (actual time=1121.356..3977.569 rows=6800102 loops=3)
        Recheck Cond: (ol_delivery_d > '2010-05-23 12:00:00'::timestamp without time zone)
        Rows Removed by Recheck: 1000000
        Heap Blocks: exact=1216
        > Bitmap Index Scan on idx_order_line (cost=0.00..373404.41 rows=20201580 width=0) (actual time=1109.727..1109.728 rows=20400306 loops=1)
          Index Cond: (ol_delivery_d > '2010-05-23 12:00:00'::timestamp without time zone)
-> Parallel Hash (cost=71764.16..71764.16 rows=33670 width=16) (actual time=709.724..709.726 rows=38693 loops=3)
  Buckets: 131072  Batches: 1  Memory Usage: 6528kB
-> Nested Loop (cost=45.37..45.419 rows=421 width=4) (actual time=45.397..545.419 rows=38693 loops=3)
  > Parallel Index Scan using ix_stock on item (cost=0.29..3667.13 rows=421 width=4) (actual time=45.316..472.273 rows=484 loops=3)
    Filter: (i_data ~~ '%c%'::text)
    Rows Removed by Filter: 32850
  -> Index Scan using ix_stock on stock (cost=0.43..159.04 rows=271 width=12) (actual time=0.029..0.136 rows=80 loops=1451)
    Index Cond: (s_i_id = item.i_id)

Planning Time: 2.086 ms
 JIT:
  Functions: 94
  Options:Inlining true, Optimization true, Expressions true, Deforming true
  Timing: Generation 30.706 ms, Inlining 519.293 ms, Optimization 854.523 ms, Emission 526.525 ms, Total 1931.048 ms
Execution Time: 29.34.616 ms
54 rows)
```

Figura 63: Plano e Tempo de execução da Query Analítica 3 com SeqScan a off com índices.

Na tentativa de melhorar o tempo de execução, criou-se novos índices, por exemplo sobre o *i_data* para melhorar no "*where i_data like 'c%'*". Não se obteve sucesso porque o *postgres* não recorre aos mesmos.

1 `CREATE INDEX idx_i_data ON item(i_data);`

Listing 10: Criação de índice

4.3.2 Vistas Materializadas

Após correr mais testes com a *query* foi notado através da ferramenta *PgAdmin* que o custo maioritário está presente na zona da *sub query*. Deparado com este facto, foi tomada a decisão de criar uma vista materializada para que o tempo de execução total fosse diminuído na sua maioria.

SQL		Statistics		Dependencies		Dependents		tpcc/joaolinhar...		tpcc/joaolinhas@tpcc *	
Data Output		Explain		Messages		Notifications					
Graphical		Analysis		Statistics							
3.	→ Hash Inner Join (cost=1.32..364.82 rows=400 width=55) (actual=594.317..594.32 rows=1 ... Hash Cond: (supplier.su_nationkey = nation.n_nationkey)					0.033 ms	594.32 ms	1 400	1	400	1
4.	→ Seq Scan on supplier as supplier (cost=0..322 rows=10000 width=59) (actual=0.028....)					0.031 ms	0.031 ms	1 232.56	43	10000	1
5.	→ Hash (cost=1.31..1.31 rows=1 width=4) (actual=594.256..594.257 rows=1 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 9 kB					0.007 ms	594.257 ms	1 1	1	1	1
6.	→ Seq Scan on nation as nation (cost=0..1.31 rows=1 width=4) (actual=594.246..5... Filter: (n.name = 'JAPAN') Rows Removed by Filter: 24					594.25 ms	594.25 ms	1 1	1	1	1
7.	→ Hash (cost=3299195.29..3299195.29 rows=200 width=4) (actual=27066.263..27127.134 r... Buckets: 1024 Batches: 1 Memory Usage: 8 kB					0.002 ms	27127.134 ms	1 0	0	200	1
8.	→ Aggregate (cost=3299193.29..3299195.29 rows=200 width=4) (actual=27066.262..27...					0.006 ms	27127.132 ms	1 0	0	200	1
9.	→ Subquery Scan (cost=2936606.37..3298526.77 rows=266607 width=4) (actual=...)					0.003 ms	27127.127 ms	1 0	0	266607	1
10.	→ Aggregate (cost=2936606.37..3295860.7 rows=266607 width=16) (actual=... Filter: ((2 * stock.s_quantity) > sum(order_line.ol_quantity)) Rows Removed by Filter: 116080					112.112 ms	27127.125 ms	1 0	0	266607	1
11.	→ Gather Merge (cost=2936606.37..3270533.04 rows=1599642 width=2...)					-51355.741 ms	27015.014 ms	1 4.6	348240	1599642	1
12.	→ Aggregate (cost=2935606.35..3084894.64 rows=799821 width=2...)					5455.647 ms	78370.755 ms	1 6.9	116080	799821	3
13.	→ Sort (cost=2935606.35..2970928.87 rows=14129008 width=...)					50048.188 ms	72915.109 ms	1 1.8	7888027	14129008	3
14.	→ Hash Inner Join (cost=119084.66..774704.28 rows=141... Hash Cond: (order_line.ol_id = stock.s_id)					13776.72 ms	22866.921 ms	1 1.8	7888027	14129008	3
15.	→ Seq Scan on order_line as order_line (cost=0..3842... Filter: (ol.delivery_d > '2010-05-23 12:00:00'::timestamp without time zone) Rows Removed by Filter: 321744					6787.527 ms	6787.527 ms	1 1.25	6800102	8498720	3
16.	→ Hash (cost=118243.48..118243.48 rows=67318 width=... Buckets: 262144 Batches: 1 Memory Usage: 7552 kB)					382.974 ms	2302.674 ms	1 1.74	38693	67318	3
17.	→ Nested Loop Inner Join (cost=0.72..118243.48...)					39.38 ms	1919.7 ms	1 1.74	38693	67318	3
18.	→ Index Scan using pk_item on item as item... Filter: (i.data ~~ 'c%'::text) Rows Removed by Filter: 32850					1529.178 ms	1529.178 ms	1 1.74	484	842	3
19.	→ Index Scan using ix_stock on stock as sto... Index Cond: (s_id = item.i_id)					351.142 ms	351.142 ms	1 3.38	80	270	1451

Figura 64: Região crítica do tempo de execução da *query 3*.

Desta forma com a criação da vista materializada, os resultados da *sub query* estarão guardados em memória o que possibilita a obtenção dos mesmos de uma forma rápida.

O comando utilizado para a criação da vista materializada foi o seguinte:

```

1 CREATE MATERIALIZED VIEW qa3_mv AS
2   (select mod(s_i_id * s_w_id, 10000)
3    from stock , order_line
4    where s_i_id in
5      (select i_id
6       from item
7       where i_data like 'c%')
8      and ol_i_id=s_i_id
9      and ol_delivery_d > '2010-05-23 12:00:00'
10     group by s_i_id , s_w_id , s_quantity
11     having 10*s_quantity > sum(ol_quantity));

```

Listing 11: Vista Materializada

De seguida a interrogação analítica, para a utilização da vista materializada criada, foi transformada da seguinte forma:

```

1 select su_name, su_address
2 from supplier, nation
3 where su_suppkey in
4   (select * from qa3_mv)
5   and su_nationkey = n_nationkey
6   and n_name = 'JAPAN'
7 order by su_name

```

Listing 12: Nova estrutura da query

Deste modo, mantemos na interrogação analítica a restrição relativa ao país obtendo ainda assim flexibilidade na execução da mesma.

Como esperado, o tempo de execução da *query* diminuiu de 27,697 s (27697.012 ms) para 0.006573 s (6.573 ms) o que representa uma redução colossal. Com isto, se demonstra que a criação da vista materializada foi extremamente benéfica para a extrema redução no tempo da interrogação analítica.

```

-----[ QUERY PLAN ]-----
Sort (cost=429.82..430.01 rows=75 width=51) (actual time=6.449..6.455 rows=61 loops=1)
  Sort Key: supplier.su_name
  Sort Method: quicksort Memory: 32kB
-> Hash Semi Join (cost=61.73..427.48 rows=75 width=51) (actual time=1.001..6.374 rows=61 loops=1)
    Hash Cond: (supplier.su_suppkey = qa3_mv.mod)
      -> Hash Join (cost=1.32..364.82 rows=400 width=55) (actual time=0.133..5.518 rows=377 loops=1)
        Hash Cond: (supplier.su_nationkey = nation.n_nationkey)
          -> Seq Scan on supplier (cost=0.00..322.00 rows=10000 width=59) (actual time=0.015..4.281 rows=10000 loops=1)
          -> Hash (cost=1.31..1.31 rows=1 width=4) (actual time=0.067..0.068 rows=1 loops=1)
            Buckets: 1024 Batches: 1 Memory Usage: 9kB
              -> Seq Scan on nation (cost=0.00..1.31 rows=1 width=4) (actual time=0.059..0.061 rows=1 loops=1)
                Filter: (n_name = 'JAPAN'::bpchar)
                Rows Removed by Filter: 24
      -> Hash (cost=32.40..32.40 rows=2240 width=4) (actual time=0.723..0.723 rows=2240 loops=1)
        Buckets: 4096 Batches: 1 Memory Usage: 111kB
          -> Seq Scan on qa3_mv (cost=0.00..32.40 rows=2240 width=4) (actual time=0.058..0.353 rows=2240 loops=1)
Planning Time: 0.770 ms
Execution Time: 6.573 ms
(18 rows)

```

Figura 65: Tempo e plano de execução da query 3 após criação da vista materializada.

Apesar de obter sucesso na diminuição do tempo de execução, deparou-se com o facto de existir *sequential scans* a serem utilizados o que permite a possibilidade de ainda obter um melhor resultado. Com isto, foram criados novamente alguns dos índices referidos na subsecção 4.3.1, nomeadamente os índices *idx_su_nationkey* e *idx_n_name* de forma a que o filtro sobre as nações seja mais rápido. Obteve-se os seguintes resultados:

```

QUERY PLAN
Sort (cost=313.57..313.76 rows=75 width=51) (actual time=2.693..2.698 rows=61 loops=1)
  Sort Key: supplier.su_name
  Sort Method: quicksort  Memory: 32kB
-> Hash Semi Join (cost=71.78..311.24 rows=75 width=51) (actual time=0.838..2.622 rows=61 loops=1)
  Hash Cond: (supplier.su_suppkey = qa3_mv.mod)
    -> Nested Loop (cost=11.38..248.58 rows=400 width=55) (actual time=0.132..1.878 rows=377 loops=1)
      -> Seq Scan on nation (cost=0.00..1.31 rows=1 width=4) (actual time=0.019..0.024 rows=1 loops=1)
        Filter: (n.name = 'JAPAN'::bpchar)
        Rows Removed by Filter: 24
      -> Bitmap Heap Scan on supplier (cost=11.38..243.27 rows=400 width=59) (actual time=0.109..1.793 rows=377 loops=1)
        Recheck Cond: (su_nationkey = nation.n_nationkey)
        Heap Blocks: exact=179
        -> Bitmap Index Scan on idx_su_nationkey (cost=0.00..11.29 rows=400 width=0) (actual time=0.077..0.077 rows=377 loops=1)
          Index Cond: (su_nationkey = nation.n_nationkey)
-> Hash (cost=32.40..32.40 rows=2240 width=4) (actual time=0.640..0.640 rows=2240 loops=1)
  Buckets: 4096  Batches: 1  Memory Usage: 111kB
    -> Seq Scan on qa3_mv (cost=0.00..32.40 rows=2240 width=4) (actual time=0.020..0.267 rows=2240 loops=1)
Planning Time: 0.715 ms
Execution Time: 2.859 ms
[19 rows]

```

Figura 66: Tempo e plano de execução da query 3 após criação da vista materializada com índices.

Apesar de que no plano apenas o índice *idx_su_nationkey* é utilizado, o resultado diminui ainda mais de 0.006573 s (6.573 ms) para 0.002859 s (2.859 ms) o que demonstra que foi uma boa decisão a criação dos mesmos.

No final foi possível obter uma melhoria de 27,697 s (27697.012 ms) para 0.002859 s (2.859 ms) o que se traduz numa diminuição de tempo de execução extremamente alta o que é bastante satisfatório para o pretendido ao que a criação da vista materializada foi um factor crucial para a optimização desta interrogação.

Por outro lado, devido ao facto das vistas materializadas ficarem estáticas no tempo existe o problema dos dados ficarem desactualizados, logo torna-se necessário actualizar o conteúdo da vista materializada criada. Foram pensadas em duas possibilidades para actualizar estes dados, uma delas seria através de *triggers*, aos quais caso haja uma alteração numa das tabelas que são utilizadas pela *query* da vista materializada os dados da mesma são actualizados. A outra possibilidade seria através do comando *REFRESH MATERIALIZED VIEW* que nos permite reconstruir o resultado da nossa *materialized view*.

Apesar do processo do comando referido anteriormente ser custoso em termos de tempo, parece ser mais vantajoso o utilizar em intervalos de tempo do que através de *triggers* devido às constantes inserções na tabela *order_line*. Este processo de actualização parece ser razoável considerando a natureza da interrogação analítica.

5 Conclusão

Numa primeira fase do trabalho prático, a principal decisão a ser tomada foi qual seria o ambiente de testes no qual iria ser executado o sistema de *benchmark*. Decidiu-se então usar as máquinas sugeridas pelo docente da cadeira, nomeadamente, *n1-standard-2* e *f1-micro*. De forma a melhorar o desempenho das operações de escrita e leitura do sistema, decidiu-se introduzir um disco *SSD* na máquina que hospedava o servidor de base de dados e constatou-se que, efectivamente, o uso deste tipo de discos aumenta o desempenho de um sistema de base de dados. Para além da arquitectura das máquinas e ainda em relação ao ambiente de teste, foi escolhida uma configuração de base em relação ao número de *warehouses* e o número máximo de clientes. Em relação ao número de *warehouses* foi escolhido o número que fez com que a base de dados tenha um tamanho superior à memória *RAM* da máquina, para garantir que os dados processados não aconteceriam sempre em memória, obrigando a haver trocas entre memória e disco. Quanto ao número de clientes decidiu-se usar o valor com maior *throughput* com o objectivo de aumentar ainda mais o mesmo.

Dada a configuração base do sistema, o próximo passo foi decidir quais configurações se iriam analisar em relação ao *PostgreSQL*. Decidiu-se alterar as configurações que tinham um impacto directo nas operações de transações, nomeadamente, as configurações correspondentes aos domínios *Settings* e *Checkpoints*. Quanto ao domínio *Archiving*, foi testado o *archiving_mode*, simplesmente para ver qual seria o impacto de ter esta funcionalidade ligada mas, no entanto, decidiu-se descartar este domínio de configurações pois não se achou que fossem muito relevantes dada a natureza do trabalho prático. Ainda em relação a configurações do sistema de base de dados, foram analisados diferentes níveis de isolamento e verificou-se que, neste caso, o *Repeatable Read* era o que oferecia maior desempenho a nível de débito. O *Read Committed* e *Read Committed* também seriam níveis de isolamento viáveis se se quisesse garantir menor *abort rate*.

Após toda a análise feita a cada configuração, chegou-se a uma configuração final que combinava vários valores de *settings* e *checkpoints*. Dada esta configuração foi possível melhorar significativamente o desempenho do sistema, tanto a nível de débito, tempo de resposta e *abort rate*, conseguindo atingir os valores estipulados no início do trabalho prático. Desta forma, pode-se concluir que o objectivo desta fase foi feita com sucesso.

A segunda parte do trabalho prático tinha como objectivo analisar três interrogações analíticas do *benchmark TPC-H*. Era pretendido melhorar o desempenho das interrogações acrescentando mecanismos de redundância na base de dados, nomeadamente, índices e vistas materializadas. Em todas as interrogações analisadas consegui-se um aumento considerável no desempenho das mesmas. A primeira interrogação passou de 2.8s para 0.8ms, a segunda de 2min e 24s para 12s e a terceira de 27s para 2.8ms. Pode-se então concluir que os mecanismos de redundância acrescentados foram realmente úteis na optimização das interrogações.

Em relação à automatização do processo de análise, os *scripts* criados permitiram uma

análise muita mais rápida de resultados permitindo ao grupo forcar-se no ponto essencial do trabalho que era a implicação que as diferentes configurações do sistema de base de dados têm no desempenho das transações e interrogações.

Para terminar, é de notar a importância de um administrador de base dados no desenvolvimento de qualquer aplicação que use um *DBMSs* para persistir dados. Como se pode constatar através de todos os resultados obtidos, mesmo tendo discos e máquinas eficientes, à sempre margem para melhorar o desempenho de um sistema de dados.

6 Anexos

Script de Configuração do Database Server

```
1 #!/bin/bash
2
3 # -----
4 #     DATABASE SERVER CONFIGURATION, INITIATE, RECOVER
5 #             AND GOOGLE CLOUD LOCALDISKS OPTIONS
6 #                     ABD UMINHO
7 # -----
8
9 helpFunction(){
10     echo ""
11     echo "Usage: $0 -a listenAddress -n localNetwork -d(optional) -i(optional)"
12     echo "-e "\t-a Define the listen address (eg: server) at the postgresql.conf"
13     echo "-e "\t-n Define the local network (eg: 10.128.0.0) at the pg_hba.conf"
14     echo "-e "\t-d Option if we want to use local disks from Google Cloud"
15     echo "-e "\t-i Option if it's the initial configuration , if this options is
16     echo "not active then it will just recover from the reboot"
17     exit 1 # Exit script after printing help
18 }
19
20 unset LOCALDISKS INITIAL_CONFIGURATION
21
22 while getopts "a:n:id" opt
23 do
24     case "$opt" in
25         a ) listenAddress="$OPTARG" ;;
26         n ) localNetwork="$OPTARG" ;;
27         i ) INITIAL_CONFIGURATION='yes' ;;
28         d ) LOCALDISKS='yes' ;;
29         ? ) helpFunction ;; # Print helpFunction in case parameter is non-existent
30     esac
31 done
32
33 # Print helpFunction in case number of listen address or local network was not
34 # provided
35 if [ -z "$listenAddress" ] || [ -z "$localNetwork" ]
36 then
37     echo "Some or all of the parameters are empty.";
38     helpFunction
39 fi
40
41 # Verify if it's the initial configuration
42 if [ -n "$INITIAL_CONFIGURATION" ]
43 then
```

```

42 # Update packages
43 yes | sudo apt-get update
44
45 # Install postgres 12
46 yes | sudo apt-get install postgresql-12
47
48 # Stop postgres
49 sudo systemctl stop postgresql
50
51 # Disable postgres
52 sudo systemctl disable postgresql
53 fi
54
55 # Verify if the local disks option is active to change to the correct directory
56 if [ -n "$LOCALDISKS" ]
57 then
58     echo ">>>>>>> Using Google Cloud Local Disks."
59
60     # Run the script to configure the local disks
61     sh ~/scripts/auxiliary_scripts/localdisksconfiguration.sh
62
63     # Change to the install directory
64     cd /mnt/disks/postgresql/
65 else
66     # Change to the install directory
67     cd
68 fi
69
70 # Create configuration data in the current dir
71 /usr/lib/postgresql/12/bin/initdb -D data
72
73 # Define the listening address
74 sed -i.bak "s/^#listen_addresses =.*/listen_addresses ='localhost , $listenAddress
75 '/g" data/postgresql.conf
76
77 # Define the listening address
78 echo -e "host      all          all          $localNetwork/24
79   trust" >> data/pg_hba.conf
80
81 # Initialize the server
82 /usr/lib/postgresql/12/bin/postgres -D data -k. </dev/null &>/dev/null &

```

Listing 13: Script de Configuração do Database Server

Script de Configuração do Servidor de Benchmark

```
1 #!/bin/bash
2
3 # -----
4 #      BENCHMARK SERVER CONFIGURATION – SCRIPT
5 #          ABD UMINHO
6 #
7
8 helpFunction(){
9     echo ""
10    echo "Usage: $0 -s dbServerName -u dbUser -w nrWarehouses"
11    echo -e "\t-s Define the server name"
12    echo -e "\t-u Define the database user"
13    echo -e "\t-w Define the Number of Warehouses"
14    exit 1 # Exit script after printing help
15}
16
17 while getopts "s:u:w:" opt
18 do
19     case "$opt" in
20         s ) dbServerName="$OPTARG" ;;
21         u ) dbUser="$OPTARG" ;;
22         w ) nrWarehouses="$OPTARG" ;;
23         ? ) helpFunction ;; # Print helpFunction in case parameter is non-existent
24     esac
25 done
26
27 # Print helpFunction in case number of db server name, db user and number of
28 # warehouses was not provided
29 if [ -z "$dbServerName" ] || [ -z "$dbUser" ] || [ -z "$nrWarehouses" ]
30 then
31     echo "Some or all of the parameters are empty.";
32     helpFunction
33 fi
34
35 # Update packages
36 yes | sudo apt-get update
37
38 # Install Java
39 yes | sudo apt-get install openjdk-8-jdk
40
41 # Install pip
42 yes | sudo apt install python3-pip
43
44 # Install scipy
45 yes | pip3 install scipy
```

```

46 # Move file showtpc.py to the directory results
47 mkdir -p ~/results
48 cp ~/scripts/files/showtpc.py ~/results
49
50 # Untar files
51 tar xvf ~/scripts/files/tpc-c-0.1-SNAPSHOT-tpc-c.tar.gz -C ~/
52 tar xvf ~/scripts/files/extra.tgz -C ~/
53
54 # Cd to the tpcc directory
55 cd ~/tpc-c-0.1-SNAPSHOT
56
57 # Define the database connection address
58 sed -i.bak "s/^db.connection.string=.*#db.connection.string=jdbc:postgresql://${dbServerName}/tpcc#g" etc/database-config.properties
59
60 # Define the database username
61 sed -i.bak "s/^db.username=.*/db.username=${dbUser}/g" etc/database-config.properties
62
63 # Define the database password
64 sed -i.bak "s/^db.password=.*/db.password=/g" etc/database-config.properties
65
66 # Install postgresql client
67 yes | sudo apt-get install postgresql-client-12
68
69 # Run the script to create a new db
70 sh ~/scripts/auxiliary_scripts/createdb.sh $dbServerName $nrWarehouses

```

Listing 14: Script de Configuração do Servidor de Benchmark

Script Limpeza Base de Dados

```
1 #!/bin/bash
2
3 # -----
4 #     DATABASE SERVER CLEAN POSTGRES DATA FOLDER
5 #             AND CREATE A NEW ONE
6 #                     ABD UMINHO
7 # -----
8
9 helpFunction(){
10     echo ""
11     echo "Usage: $0 -a listenAddress -n localNetwork -d(optional)"
12     echo "-e "\t-a Define the listen address (eg: server) at the postgresql.conf"
13     echo "-e "\t-n Define the local network (eg: 10.128.0.0) at the pg_hba.conf"
14     echo "-e "\t-d Option if we want to use local disks from Google Cloud"
15     echo "-e "\t-i Option if it's the initial configuration , if this options is
16         not active then it will just recover from the reboot"
17     exit 1 # Exit script after printing help
18 }
19
20 unset LOCALDISKS
21
22 while getopts "a:n:id" opt
23 do
24     case "$opt" in
25         a ) listenAddress="$OPTARG" ;;
26         n ) localNetwork="$OPTARG" ;;
27         d ) LOCALDISKS='yes' ;;
28         ? ) helpFunction ;; # Print helpFunction in case parameter is non-existent
29     esac
30 done
31
32 # Print helpFunction in case number of listen address or local network was not
33 # provided
34 if [ -z "$listenAddress" ] || [ -z "$localNetwork" ]
35 then
36     echo "Some or all of the parameters are empty.";
37     helpFunction
38 fi
39
40 # Verify if the local disks option is active to change to the correct directory
41 if [ -n "$LOCALDISKS" ]
42 then
43     echo ">>>>>>>> Using Google Cloud Local Disks."
44     # Change to the install directory
45     cd /mnt/disks/postgresql/
```

```

45
46     else
47         # Change to the install directory
48         cd
49 fi
50
51 # Stop postgres database
52 /usr/lib/postgresql/12/bin/pg_ctl stop -D data
53
54 # Remove data folder
55 sudo rm -rf data
56
57 # Create a new configuration data in the current dir
58 /usr/lib/postgresql/12/bin/initdb -D data
59
60 # Define the listening address
61 sed -i.bak "s/^#listen_addresses =.*/listen_addresses ='localhost ,${listenAddress}
62 '/g" data/postgresql.conf
63
64 # Define the listening address
65 echo -e "host    all            all          $localNetwork/24
66     trust" >> data/pg_hba.conf
67
68 # Initialize the server
69 /usr/lib/postgresql/12/bin/postgres -D data -k. </dev/null &>/dev/null &

```

Listing 15: Script Limpeza Base de Dados

Script Drop Base de Dados e Criação de Nova Dado um Número de Warehouses

```
1 #!/bin/bash
2
3 # -----
4 #     CLEAN DATABASE AND CREATE A NEW ONE BY A GIVEN
5 #             NUMBER OF WAREHOUSES – SCRIPT
6 #                     ABD UMINHO
7 # -----
8
9 helpFunction(){
10     echo ""
11     echo "Usage: $0 -s dbServerName -w nrWarehouses"
12     echo -e "\t-s Define the PostgreSQL DB Server Name"
13     echo -e "\t-w Define the Number of Warehouses"
14     exit 1 # Exit script after printing help
15 }
16
17 while getopts "s:w:" opt
18 do
19     case "$opt" in
20         w ) nrWarehouses="$OPTARG" ;;
21         s ) dbServerName="$OPTARG" ;;
22         ? ) helpFunction ;; # Print helpFunction in case parameter is non-existent
23     esac
24 done
25
26 # Print helpFunction in case number of warehouses or db server name was not
27 # provided
27 if [ -z "$nrWarehouses" ] || [ -z "$dbServerName" ]
28 then
29     echo "Some or all of the parameters are empty.";
30     helpFunction
31 fi
32
33 # Begin script in case all parameters are correct
34
35 # Drop database
36 dropdb -h $dbServerName tpcc
37
38 # Run the script to create the db again
39 sh ~/scripts/auxiliary_scripts/createdb.sh $dbServerName $nrWarehouses
```

Listing 16: Script Drop Base de Dados e Criação de Nova Dado um Número de Warehouses

Script que Corre o Script Transacional Dado um Número de Clientes e Warehouses

```

1  #!/bin/bash
2
3  #
4  #      RUN THE TRANSACTION SCRIPT FOR A GIVEN NUMBER
5  #          OF CLIENTS AND WAREHOUSES – SCRIPT
6  #                      ABD UMINHO
7  #
8
9  helpFunction(){
10    echo ""
11    echo "Usage: $0 -s dbServerName -w nrWarehouses -c nrClients"
12    echo -e "\t-s Define the Database Server Name"
13    echo -e "\t-w Define the Number of Warehouses"
14    echo -e "\t-u Define the database user"
15    echo -e "\t-c Define the Number of Clients"
16    exit 1 # Exit script after printing help
17 }
18
19 while getopts "c:w:s:u:" opt
20 do
21   case "$opt" in
22     w ) nrWarehouses="$OPTARG" ;;
23     s ) dbServerName="$OPTARG" ;;
24     u ) dbUser="$OPTARG" ;;
25     c ) nrClients="$OPTARG" ;;
26     ? ) helpFunction ;; # Print helpFunction in case parameter is non-existent
27   esac
28 done
29
30 # Print helpFunction in case number of db server name or db user or warehouses
31 # or number of clients was not provided
32 if [ -z "$dbServerName" ] || [ -z "$dbUser" ] || [ -z "$nrWarehouses" ] || [ -z
33   "$nrClients" ]
34 then
35   echo "Some or all of the parameters are empty.";
36   helpFunction
37 fi
38
39 # Begin script in case all parameters are correct
40
41 # Cd to the correct directory
42 cd ~/tpc-c-0.1-SNAPSHOT
43
44 # Define the database connection address

```

```

43 sed -i.bak "s#\^db.connection.string=.*#\^db.connection.string=jdbc:postgresql://\${{  
44 dbServerName}}/tpcc#g" etc/database-config.properties  
45  
46 # Define the database username  
47 sed -i.bak "s/^db.username=.*/db.username=\$\{dbUser\}/g" etc/database-config.  
48 properties  
49  
50 # Define the number of warehouses  
51 sed -i.bak "s/^tpcc.number.warehouses=.*/tpcc.number.warehouses=\$nrWarehouses/g"  
52 etc/workload-config.properties  
53  
54 # Define the number of clients  
55 sed -i.bak "s/^tpcc.numclients =.*/tpcc.numclients = \$nrClients/g" etc/workload-  
56 config.properties  
57  
58 # Run the transaction script  
59 echo ">>>>> Running the transaction script..."  
60 sh ~/tpc-c-0.1-SNAPSHOT/run.sh

```

Listing 17: Script que Corre o Script Transacional Dado um Número de Clientes e Warehouses

Script Restore da Base de Dados Dado um Ficheiro de Backup, Número de Warehouses e Clientes

```
1 #!/bin/bash
2
3 # -----
4 #          DROP DATABASE AND RESTORE THE DB FROM
5 #          A PREVIOUS BACKUP – SCRIPT
6 #          ABD UMINHO
7 # -----
8
9 helpFunction(){
10    echo ""
11    echo "Usage: $0 -s dbServerName -w nrWarehouses -b backupFile"
12    echo -e "\t-s Define the PostgreSQL DB Server Name"
13    echo -e "\t-w Define the Number of Warehouses"
14    echo -e "\t-b Define the backup file path"
15    exit 1 # Exit script after printing help
16}
17
18 while getopts "s:b:w:" opt
19 do
20     case "$opt" in
21         b ) backupFile="$OPTARG" ;;
22         w ) nrWarehouses="$OPTARG" ;;
23         s ) dbServerName="$OPTARG" ;;
24         ? ) helpFunction ;; # Print helpFunction in case parameter is non-existent
25     esac
26 done
27
28 # Print helpFunction in case number of warehouses or db server name was not
29 # provided
30 if [ -z "$nrWarehouses" ] || [ -z "$backupFile" ] || [ -z "$dbServerName" ]
31 then
32     echo "Some or all of the parameters are empty."
33     helpFunction
34 fi
35
36 # Begin script in case all parameters are correct
37
38 # Drop database
39 dropdb -h $dbServerName tpcc
40
41 backupFilePath=$(dirname $(readlink -f "$backupFile"))
42 backupFileName=$(basename -- "$backupFile")
43
44 # Run the script to create tables and restore the db
```

```
44 sh ~/scripts/auxiliary_scripts/createdb.sh $dbServerName $nrWarehouses "  
$backupFilePath/$backupFileName"
```

Listing 18: Script Restore da Base de Dados Dado um Ficheiro de Backup, Número de Warehouses e Clientes

Script Criação e Povoamento da Base de Dados com Ficheiro de Backup ou Correndo o Script Transacional

```
1 #!/bin/bash
2
3 # -----
4 #          CREATE DATABASE SCRIPT
5 #          ABD UMINHO
6 # -----
7
8 dbServerName=$1
9 nrWarehouses=$2
10 backupFile=$3
11
12 # Create a new database
13 createdb -h $dbServerName tpcc
14
15 # Load SQL scripts
16 cd ~/tpc-c-0.1-SNAPSHOT
17 psql -h $dbServerName tpcc < etc/sql/postgresql/createtable.sql
18 psql -h $dbServerName tpcc < etc/sql/postgresql/createindex.sql
19 for i in etc/sql/postgresql/*01; do psql -h $dbServerName tpcc < $i; done
20
21 # Define the number of warehouses
22 sed -i.bak "s/^tpcc.number.warehouses=.*/tpcc.number.warehouses=$nrWarehouses/g"
23           etc/workload-config.properties
24
25 if [ -z "$backupFile" ]
26 then
27     # Execute load script
28     echo ">>>>>>> Executing load script..."
29     sh ~/tpc-c-0.1-SNAPSHOT/load.sh
30
31     # Create extra tables
32     cd ~/extra
33     psql -h $1 tpcc < createExtraTables.sql
34 else
35     # Create extra tables
36     cd ~/extra
37     psql -h $1 tpcc < createExtraTables.sql
38
39     # Restore the database
40     echo ">>>>>>> Restore initiated..."
41     pg_restore -h $dbServerName -c -d tpcc < $backupFile
42 fi
```

Listing 19: Script Criação e Povoamento da Base de Dados com Ficheiro de Backup ou Correndo o Script Transacional

Script Configuração de Discos Locais da Google

```
1 #!/bin/bash
2
3 # -----
4 #      GOOGLE CLOUD LOCAL DISKS CONFIGURATION SCRIPT
5 #          ABD UMINHO
6 #
7
8 # Create the file system
9 sudo mkfs.ext4 -F /dev/nvme0n1
10
11 # Create the directory
12 sudo mkdir -p /mnt/disks/postgresql
13
14 # Mount the filesystem
15 sudo mount /dev/nvme0n1 /mnt/disks/postgresql
16
17 # Give permissions to the mounted directory
18 sudo chmod a+rwx /mnt/disks/postgresql
```

Listing 20: Script Configuração de Discos Locais da Google

Script Execução Teste Para Clientes Criando Máquinas Virtuais Novas

```
1 #!/bin/bash
2
3 # Create a new server VM
4 echo ">>>>>>>>>> Creating a new Server VM on Google Cloud."
5 gcloud beta compute --project=abd-2020-2021 instances create $1 --zone=us-
6   central1-a --machine-type=n1-standard-2 --subnet=default --network-tier=
7     PREMIUM --maintenance-policy=MIGRATE --service-account=536808376468-
8       compute@developer.gserviceaccount.com --scopes=https://www.googleapis.com/
9         auth/devstorage.read_only,https://www.googleapis.com/auth/logging.write,
10        https://www.googleapis.com/auth/monitoring.write,https://www.googleapis.com/
11        auth/servicecontrol,https://www.googleapis.com/auth/service.management.
12        readonly,https://www.googleapis.com/auth/trace.append --image=ubuntu-2004-
13      focal-v20201211 --image-project=ubuntu-os-cloud --boot-disk-size=10GB --boot-
14      --disk-type=pd-ssd --boot-disk-device-name=$1 --local-ssd-interface=NVME --no-
15      --shielded-secure-boot --shielded-vtpm --shielded-integrity-monitoring --
16      reservation-affinity=any
17
18 # Wait 90 seconds for instance to be on
19 sleep 90s
20
21 # Copy scripts to new server VM
22 echo ">>>>>>>>>> Copying scripts folder to the new Server VM on Google Cloud.
23   "
24 gcloud compute scp --zone us-central1-a --recurse ~/scripts $1:
25
26 # Run the server configuration script
27 echo ">>>>>>>>>> Running the initial configuration script on new Server."
28 gcloud compute ssh --zone us-central1-a $1 --command "~/scripts/
29   dbserverconfiguration.sh -a $1 -n 10.128.0.0 -d -i"
30
31 # Connect to the remote server and execute the script to clean the DB
32 echo ">>>>>>>>>>> Going to clear the database on db server."
33 gcloud compute ssh --zone us-central1-a $1 --command "~/scripts/
34   dbservercleanpostgresdata.sh -a $1 -n 10.128.0.0 -d"
35
36 # Wait 30 seconds
37 sleep 30s
38
39 # Execute the restore script
40 echo ">>>>>>>>>>> Restoring started."
41 ~/scripts/restoredb.sh -s $1 -w $3 -b $4
42
43 # Run the transaction script
44 echo ">>>>>>>>>>> Run transactional script."
45 ~/scripts/runclients.sh -s $1 -u $2 -w $3 -c $5
```

```

32
33 # Delete the VM
34 echo ">>>>>>>>> Deleting the database server."
35 yes | gcloud compute instances delete $1 --zone us-central1-a
36
37 # Create dat results directory and change file name
38 echo ">>>>>>>>> Changing file name."
39 mkdir -p ~/dat_results/$6
40 mv ~/tpc-c-0.1-SNAPSHOT/TPCC*.dat ~/dat_results/$6/$7.dat
41
42 # Run script
43 echo ">>>>>>>>> Running showtpc.py"
44 mkdir -p ~/results/$6
45 ~/results/showtpc.py -bc ~/dat_results/$6/$7.dat >> ~/results/$6/$7.txt

```

Listing 21: Script Execução Teste Para Clientes Criando Máquinas Virtuais Novas

Script Execução Teste Para Pâmetros do PostgreSQL Criando Máquinas Virtuais Novas

```
1 #!/bin/bash
2
3 # Create a new server VM
4 echo ">>>>>>>>> Creating a new Server VM on Google Cloud."
5 gcloud beta compute --project=abd-2020-2021 instances create $1 --zone=us-
6   central1-a --machine-type=n1-standard-2 --subnet=default --network-tier=
7     PREMIUM --maintenance-policy=MIGRATE --service-account=536808376468-
8       compute@developer.gserviceaccount.com --scopes=https://www.googleapis.com/
9         auth/devstorage.read_only,https://www.googleapis.com/auth/logging.write,
10        https://www.googleapis.com/auth/monitoring.write,https://www.googleapis.com/
11        auth/servicecontrol,https://www.googleapis.com/auth/service.management.
12        readonly,https://www.googleapis.com/auth/trace.append --image=ubuntu-2004-
13      focal-v20201211 --image-project=ubuntu-os-cloud --boot-disk-size=10GB --boot-
14      --disk-type=pd-ssd --boot-disk-device-name=$1 --local-ssd-interface=NVME --no-
15      --shielded-secure-boot --shielded-vtpm --shielded-integrity-monitoring --
16      reservation-affinity=any
17
18 # Wait 90 seconds for instance to be on
19 sleep 90s
20
21 # Copy scripts to new server VM
22 echo ">>>>>>>>> Copying scripts folder to the new Server VM on Google Cloud.
23   "
24 gcloud compute scp --zone us-central1-a --recurse ~/scripts $1:
25
26 # Run the server configuration script
27 echo ">>>>>>>>> Running the initial configuration script on new Server."
28 gcloud compute ssh --zone us-central1-a $1 --command "~/scripts/
29   dbserverconfiguration.sh -a $1 -n 10.128.0.0 -d -i"
30
31 # Connect to the remote server and execute the script to clean the DB
32 echo ">>>>>>>>> Going to clear the database on db server."
33 gcloud compute ssh --zone us-central1-a $1 --command "~/scripts/
34   dbservercleanpostgresdata.sh -a $1 -n 10.128.0.0 -d"
35
36 # Wait 30 seconds
37 sleep 30s
38
39 # Execute the restore script
40 echo ">>>>>>>>> Restoring started."
41 ~/scripts/restoredb.sh -s $1 -w $3 -b $4
42
43 # Stop the postgresql server
44 gcloud compute ssh --zone us-central1-a $1 --command '/usr/lib/postgresql/12/bin'
45   /pg_ctl stop -D /mnt/disks/postgresql/data'
```

```

31 # Edit the postgresql.conf with the optimization
32 gcloud compute ssh --zone us-central1-a $1 --command "sed -i.bak 's/^$6.*/$7/g' \
33   /mnt/disks/postgresql/data/postgresql.conf"
34
35 # Save a log with the edited file
36 gcloud compute ssh --zone us-central1-a $1 --command "mkdir -p ~/logs && cp /mnt \
37   /disks/postgresql/data/postgresql.conf ~/logs/$9.conf"
38
39 # Start the postgresql server
40 gcloud compute ssh --zone us-central1-a $1 --command '/usr/lib/postgresql/12/bin \
41   /postgres -D /mnt/disks/postgresql/data -k. </dev/null >/dev/null &'
42
43 # Run the transaction script
44 echo ">>>>>>>>>> Run transactional script."
45 ~/scripts/runclients.sh -s $1 -u $2 -w $3 -c $5
46
47 # Copy the logs folder
48 echo ">>>>>>>>>> Copying the logs folder from the database server."
49 gcloud compute scp --zone us-central1-a --recurse $1:~/logs ~/ \
50
51 # Delete the VM
52 echo ">>>>>>>>>> Deleting the database server."
53 yes | gcloud compute instances delete $1 --zone us-central1-a
54
55 # Create dat results directory and change file name
56 echo ">>>>>>>>>> Changing file name."
57 mkdir -p ~/dat_results/$8
58 mv ~/tpc-c-0.1-SNAPSHOT/TPCC*.dat ~/dat_results/$8/$9.dat
59
60 # Run script
61 echo ">>>>>>>>>> Running showtpc.py"
62 mkdir -p ~/results/$8
63 ~/results/showtpc.py -bc ~/dat_results/$8/$9.dat >> ~/results/$8/$9.txt

```

Listing 22: Script Execução Teste Para Pârametros do PostgreSQL Criando Máquinas Virtuais Novas

Script Execução Testes das Combinações Criando Máquinas Virtuais Novas

```
1 #!/bin/bash
2
3 # Create a new server VM
4 echo ">>>>>>>>> Creating a new Server VM on Google Cloud."
5 gcloud beta compute --project=abd-2020-2021 instances create $1 --zone=us-
6   central1-a --machine-type=n1-standard-2 --subnet=default --network-tier=
7     PREMIUM --maintenance-policy=MIGRATE --service-account=536808376468-
8       compute@developer.gserviceaccount.com --scopes=https://www.googleapis.com/
9         auth/devstorage.read_only,https://www.googleapis.com/auth/logging.write,
10        https://www.googleapis.com/auth/monitoring.write,https://www.googleapis.com/
11        auth/servicecontrol,https://www.googleapis.com/auth/service.management.
12        readonly,https://www.googleapis.com/auth/trace.append --image=ubuntu-2004-
13      focal-v20201211 --image-project=ubuntu-os-cloud --boot-disk-size=10GB --boot-
14      --disk-type=pd-ssd --boot-disk-device-name=$1 --local-ssd-interface=NVME --no-
15      --shielded-secure-boot --shielded-vtpm --shielded-integrity-monitoring --
16      reservation-affinity=any
17
18 # Wait 90 seconds for instance to be on
19 sleep 90s
20
21 # Copy scripts to new server VM
22 echo ">>>>>>>>> Copying scripts folder to the new Server VM on Google Cloud.
23   "
24 gcloud compute scp --zone us-central1-a --recurse ~/scripts $1:
25
26 # Run the server configuration script
27 echo ">>>>>>>>> Running the initial configuration script on new Server."
28 gcloud compute ssh --zone us-central1-a $1 --command "~/scripts/
29   dbserverconfiguration.sh -a $1 -n 10.128.0.0 -d -i"
30
31 # Connect to the remote server and execute the script to clean the DB
32 echo ">>>>>>>>> Going to clear the database on db server."
33 gcloud compute ssh --zone us-central1-a $1 --command "~/scripts/
34   dbservercleanpostgresdata.sh -a $1 -n 10.128.0.0 -d"
35
36 # Wait 30 seconds
37 sleep 30s
38
39 # Execute the restore script
40 echo ">>>>>>>>> Restoring started."
41 ~/scripts/restoredb.sh -s $1 -w $3 -b $4
42
43 # Stop the postgresql server
44 gcloud compute ssh --zone us-central1-a $1 --command '/usr/lib/postgresql/12/bin'
45   /pg_ctl stop -D /mnt/disks/postgresql/data'
```

```

31 # Edit the postgresql.conf with the optimization
32 gcloud compute ssh --zone us-central1-a $1 --command "mv ~/scripts/
33   postgresqlconf_files/$6 /mnt/disks/postgresql/data/postgresql.conf"
34
35 # Edit the postgresql.conf to change the listen address
36 gcloud compute ssh --zone us-central1-a $1 --command "sed -i.bak \"/s|^
37   listen_addresses =.*|listen_addresses = 'localhost,$1'|\" /mnt/disks/
38   postgresql/data/postgresql.conf"
39
40 # Save a log with the edited file
41 gcloud compute ssh --zone us-central1-a $1 --command "mkdir -p ~/logs && cp /mnt
42   /disks/postgresql/data/postgresql.conf ~/logs/$8.conf"
43
44 # Start the postgresql server
45 gcloud compute ssh --zone us-central1-a $1 --command '/usr/lib/postgresql/12/bin/
46   /postgres -D /mnt/disks/postgresql/data -k. </dev/null &>/dev/null &'
47
48 # Run the transaction script
49 echo ">>>>>>>>>> Run transactional script."
50 ~/scripts/runclients.sh -s $1 -u $2 -w $3 -c $5
51
52 # Copy the logs folder
53 echo ">>>>>>>>>> Copying the logs folder from the database server."
54 gcloud compute scp --zone us-central1-a --recurse $1:~/logs ~/ 
55
56 # Delete the VM
57 echo ">>>>>>>>>> Deleting the database server."
58 yes | gcloud compute instances delete $1 --zone us-central1-a
59
60 # Create dat_results directory and change file name
61 echo ">>>>>>>>>> Changing file name."
62 mkdir -p ~/dat_results/$7
63 mv ~/tpc-c-0.1-SNAPSHOT/TPCC*.dat ~/dat_results/$7/$8.dat
64
65 # Run script
66 echo ">>>>>>>>>> Running showtpc.py"
67 mkdir -p ~/results/$7
68 ~/results/showtpc.py -bc ~/dat_results/$7/$8.dat >> ~/results/$7/$8.txt

```

Listing 23: Script Execução Testes das Combinações Criando Máquinas Virtuais Novas

Script Testes Automáticos Para Clientes

```
1 #!/bin/bash
2
3 dbUser=$1
4 nrWarehouses=$2
5 backupFile=$3
6
7 ~/scripts/auxiliary_scripts/autorun_clients.sh server-10-cli $dbUser
8     $nrWarehouses $backupFile 10 clients "$nrWarehouses"warehouses_10clients
9
10 ~/scripts/auxiliary_scripts/autorun_clients.sh server-20-cli $dbUser
11     $nrWarehouses $backupFile 20 clients "$nrWarehouses"warehouses_20clients
12
13 ~/scripts/auxiliary_scripts/autorun_clients.sh server-30-cli $dbUser
14     $nrWarehouses $backupFile 30 clients "$nrWarehouses"warehouses_30clients
15
16 ~/scripts/auxiliary_scripts/autorun_clients.sh server-40-cli $dbUser
17     $nrWarehouses $backupFile 40 clients "$nrWarehouses"warehouses_40clients
18
19 ~/scripts/auxiliary_scripts/autorun_clients.sh server-50-cli $dbUser
20     $nrWarehouses $backupFile 50 clients "$nrWarehouses"warehouses_50clients
21
22 ~/scripts/auxiliary_scripts/autorun_clients.sh server-60-cli $dbUser
23     $nrWarehouses $backupFile 60 clients "$nrWarehouses"warehouses_60clients
24
25 ~/scripts/auxiliary_scripts/autorun_clients.sh server-70-cli $dbUser
26     $nrWarehouses $backupFile 70 clients "$nrWarehouses"warehouses_70clients
27
28 ~/scripts/auxiliary_scripts/autorun_clients.sh server-80-cli $dbUser
29     $nrWarehouses $backupFile 80 clients "$nrWarehouses"warehouses_80clients
30
31 ~/scripts/auxiliary_scripts/autorun_clients.sh server-90-cli $dbUser
32     $nrWarehouses $backupFile 90 clients "$nrWarehouses"warehouses_90clients
33
34 ~/scripts/auxiliary_scripts/autorun_clients.sh server-100-cli $dbUser
35     $nrWarehouses $backupFile 100 clients "$nrWarehouses"warehouses_100clients
36
37 ~/scripts/auxiliary_scripts/autorun_clients.sh server-110-cli $dbUser
38     $nrWarehouses $backupFile 110 clients "$nrWarehouses"warehouses_110clients
39
40 ~/scripts/auxiliary_scripts/autorun_clients.sh server-120-cli $dbUser
41     $nrWarehouses $backupFile 120 clients "$nrWarehouses"warehouses_120clients
```

Listing 24: Script Testes Automáticos Para Clientes

Script Testes Automáticos Para *Settings*

```
1 #!/bin/bash
2
3 dbUser=$1
4 nrWarehouses=$2
5 backupFile=$3
6 nrClients=$4
7
8 ######
9 #          FSYNC          #
10#####
11
12 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings-1 $dbUser
13   $nrWarehouses $backupFile $nrClients "#fsync =\" "fsync = off" settings/fsync
14   fsync
15
16#####
17
18 #          SYNCHRONOUS COMMIT          #
19#####
20
21 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings-2 $dbUser
22   $nrWarehouses $backupFile $nrClients "#synchronous_commit =\" "
23   synchronous_commit = off" settings/synchronous_commit synchronous_commit_off
24
25 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings-3 $dbUser
26   $nrWarehouses $backupFile $nrClients "#synchronous_commit =\" "
27   synchronous_commit = remote_write" settings/synchronous_commit
28   synchronous_commit_remote_write
29
30 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings-4 $dbUser
31   $nrWarehouses $backupFile $nrClients "#synchronous_commit =\" "
32   synchronous_commit = local" settings/synchronous_commit
33   synchronous_commit_local
34
35 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings-5 $dbUser
36   $nrWarehouses $backupFile $nrClients "#synchronous_commit =\" "
37   synchronous_commit = remote_apply" settings/synchronous_commit
38   synchronous_commit_remote_apply
39
40#####
41 #          WAL SYNC METHOD          #
42#####
43
44 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings-6 $dbUser
45   $nrWarehouses $backupFile $nrClients "#wal_sync_method =\" "wal_sync_method =
46   fsync" settings/wal_sync_method wal_sync_method_fsync
```

```

32 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -7 $dbUser
    $nrWarehouses $backupFile $nrClients "#wal_sync_method = " "wal_sync_method =
      open_datasync" settings/wal_sync_method wal_sync_method_open_datasync
33
34 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -8 $dbUser
    $nrWarehouses $backupFile $nrClients "#wal_sync_method = " "wal_sync_method =
      open_sync" settings/wal_sync_method wal_sync_method_open_sync
35
36 ######
37 #          FULL PAGE WRITES          #
38 ######
39
40 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -9 $dbUser
    $nrWarehouses $backupFile $nrClients "#full_page_writes = " "full_page_writes
      = off" settings/full_page_writes full_page_writes_off
41
42 ######
43 #          WAL BUFFERS            #
44 ######
45
46 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -10 $dbUser
    $nrWarehouses $backupFile $nrClients "#wal_buffers = " "wal_buffers = 2MB"
      settings/wal_buffers wal_buffers_2MB
47
48 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -11 $dbUser
    $nrWarehouses $backupFile $nrClients "#wal_buffers = " "wal_buffers = 4MB"
      settings/wal_buffers wal_buffers_4MB
49
50 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -12 $dbUser
    $nrWarehouses $backupFile $nrClients "#wal_buffers = " "wal_buffers = 8MB"
      settings/wal_buffers wal_buffers_8MB
51
52 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -13 $dbUser
    $nrWarehouses $backupFile $nrClients "#wal_buffers = " "wal_buffers = 16MB"
      settings/wal_buffers wal_buffers_16MB
53
54 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -14 $dbUser
    $nrWarehouses $backupFile $nrClients "#wal_buffers = " "wal_buffers = 32MB"
      settings/wal_buffers wal_buffers_32MB
55
56 ######
57 #          COMMIT DELAY           #
58 ######
59
60 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -15 $dbUser
    $nrWarehouses $backupFile $nrClients "#commit_delay = " "commit_delay = 10"
      settings/commit_delay commit_delay_10
61

```

```

62 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -16 $dbUser
  $nrWarehouses $backupFile $nrClients "#commit_delay = " "commit_delay = 200"
  settings/commit_delay commit_delay_200
63
64 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -17 $dbUser
  $nrWarehouses $backupFile $nrClients "#commit_delay = " "commit_delay = 500"
  settings/commit_delay commit_delay_500
65
66 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -18 $dbUser
  $nrWarehouses $backupFile $nrClients "#commit_delay = " "commit_delay = 1000"
  settings/commit_delay commit_delay_1000
67
68 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -19 $dbUser
  $nrWarehouses $backupFile $nrClients "#commit_delay = " "commit_delay = 1500"
  settings/commit_delay commit_delay_1500
69
70 ######
71 #          COMMIT SIBLINGS          #
72 ######
73
74 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -20 $dbUser
  $nrWarehouses $backupFile $nrClients "#commit_siblings = " "commit_siblings =
  2" settings/commit_siblings commit_siblings_2
75
76 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -21 $dbUser
  $nrWarehouses $backupFile $nrClients "#commit_siblings = " "commit_siblings =
  4" settings/commit_siblings commit_siblings_4
77
78 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -22 $dbUser
  $nrWarehouses $backupFile $nrClients "#commit_siblings = " "commit_siblings =
  8" settings/commit_siblings commit_siblings_8
79
80 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -23 $dbUser
  $nrWarehouses $backupFile $nrClients "#commit_siblings = " "commit_siblings =
  16" settings/commit_siblings commit_siblings_16
81
82 ######
83 #          WAL LEVEL          #
84 ######
85
86 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -24 $dbUser
  $nrWarehouses $backupFile $nrClients "#wal_level = " "wal_level = minimal"
  settings/wal_level wal_level_minimal
87
88 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -25 $dbUser
  $nrWarehouses $backupFile $nrClients "#wal_level = " "wal_level = logical"
  settings/wal_level wal_level_logical
89
90 #####

```

```

91          #          WAL WRITER DELAY          #
92          ######
93
94      ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -26 $dbUser
95          $nrWarehouses $backupFile $nrClients "#wal_writer_delay = " "wal_writer_delay
96          = 100ms" settings/wal_writer_delay wal_writer_delay_100
97
98      ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -27 $dbUser
99          $nrWarehouses $backupFile $nrClients "#wal_writer_delay = " "wal_writer_delay
100         = 500ms" settings/wal_writer_delay wal_writer_delay_500
101
102     ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -28 $dbUser
103         $nrWarehouses $backupFile $nrClients "#wal_writer_delay = " "wal_writer_delay
104         = 1000ms" settings/wal_writer_delay wal_writer_delay_1000
105
106     ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -29 $dbUser
107         $nrWarehouses $backupFile $nrClients "#wal_writer_delay = " "wal_writer_delay
108         = 2000ms" settings/wal_writer_delay wal_writer_delay_2000
109
110     ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -30 $dbUser
111         $nrWarehouses $backupFile $nrClients "#wal_writer_delay = " "wal_writer_delay
112         = 5000ms" settings/wal_writer_delay wal_writer_delay_5000
113
114          ######
115          #          WAL WRITER FLUSH AFTER          #
116          #####
117
118      ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -31 $dbUser
119          $nrWarehouses $backupFile $nrClients "#wal_writer_flush_after = "
120          wal_writer_flush_after = 2MB" settings/wal_writer_flush_after
121          wal_writer_flush_after_2MB
122
123      ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -32 $dbUser
124          $nrWarehouses $backupFile $nrClients "#wal_writer_flush_after = "
125          wal_writer_flush_after = 4MB" settings/wal_writer_flush_after
126          wal_writer_flush_after_4MB
127
128      ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -33 $dbUser
129          $nrWarehouses $backupFile $nrClients "#wal_writer_flush_after = "
130          wal_writer_flush_after = 8MB" settings/wal_writer_flush_after
131          wal_writer_flush_after_8MB
132
133      ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -34 $dbUser
134          $nrWarehouses $backupFile $nrClients "#wal_writer_flush_after = "
135          wal_writer_flush_after = 16MB" settings/wal_writer_flush_after
136          wal_writer_flush_after_16MB
137
138      ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -35 $dbUser
139          $nrWarehouses $backupFile $nrClients "#wal_writer_flush_after = "

```

```
117 wal_writer_flush_after = 32MB" settings/wal_writer_flush_after  
118 wal_writer_flush_after_32MB  
119 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-settings -36 $dbUser  
$nrWarehouses $backupFile $nrClients "#wal_writer_flush_after = "  
120 wal_writer_flush_after = 64MB" settings/wal_writer_flush_after  
wal_writer_flush_after_64MB
```

Listing 25: Script Testes Automáticos Para *Settings*

Script Testes Automáticos Para *Checkpoints*

```
1 #!/bin/bash
2
3 dbUser=$1
4 nrWarehouses=$2
5 backupFile=$3
6 nrClients=$4
7
8 ######
9 #           CHECKPOINT TIMEOUT          #
10#####
11
12 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-1 $dbUser
   $nrWarehouses $backupFile $nrClients "#checkpoint_timeout ="
   checkpoint_timeout = 1min" checkpoint/checkpoint_timeout
   checkpoint_timeout_1min
13
14 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-2 $dbUser
   $nrWarehouses $backupFile $nrClients "#checkpoint_timeout ="
   checkpoint_timeout = 2min" checkpoint/checkpoint_timeout
   checkpoint_timeout_2min
15
16 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-3 $dbUser
   $nrWarehouses $backupFile $nrClients "#checkpoint_timeout ="
   checkpoint_timeout = 5min" checkpoint/checkpoint_timeout
   checkpoint_timeout_5min
17
18 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-4 $dbUser
   $nrWarehouses $backupFile $nrClients "#checkpoint_timeout ="
   checkpoint_timeout = 5min" checkpoint/checkpoint_timeout
   checkpoint_timeout_5min
19
20 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-5 $dbUser
   $nrWarehouses $backupFile $nrClients "#checkpoint_timeout ="
   checkpoint_timeout = 5min" checkpoint/checkpoint_timeout
   checkpoint_timeout_5min
21#####
22#           MAX WAL SIZE          #
23#####
24
25
26 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-6 $dbUser
   $nrWarehouses $backupFile $nrClients "max_wal_size =" "max_wal_size = 2GB"
   checkpoint/max_wal_size max_wal_size_2GB
27
28 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-7 $dbUser
   $nrWarehouses $backupFile $nrClients "max_wal_size =" "max_wal_size = 3GB"
```

```

29      checkpoint/max_wal_size max_wal_size_3GB
30      ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-9 $dbUser
31          $nrWarehouses $backupFile $nrClients "max_wal_size =" "max_wal_size = 4GB"
32          checkpoint/max_wal_size max_wal_size_4GB
33      ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-10
34          $dbUser $nrWarehouses $backupFile $nrClients "max_wal_size =" "max_wal_size
35          = 5GB" checkpoint/max_wal_size max_wal_size_5GB
36      ##### MIN WAL SIZE #####
37      #                         #
38      ##### MIN WAL SIZE #####
39
40      ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-12
41          $dbUser $nrWarehouses $backupFile $nrClients "min_wal_size =" "min_wal_size
42          = 80MB" checkpoint/min_wal_size min_wal_size_80MB
43
44      ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-13
45          $dbUser $nrWarehouses $backupFile $nrClients "min_wal_size =" "min_wal_size
46          = 160MB" checkpoint/min_wal_size min_wal_size_160MB
47
48      ##### CHECKPOINT COMPLETION TARGET #####
49      #                         #
50      ##### CHECKPOINT COMPLETION TARGET #####
51
52      ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-16
53          $dbUser $nrWarehouses $backupFile $nrClients "#checkpoint_completion_target
54          =" "checkpoint_completion_target = 0.0" checkpoint/
55          checkpoint_completion_target checkpoint_completion_target_00
56
57      ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-17
58          $dbUser $nrWarehouses $backupFile $nrClients "#checkpoint_completion_target
59          =" "checkpoint_completion_target = 0.2" checkpoint/
60          checkpoint_completion_target checkpoint_completion_target_02

```

```

56 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-18
    $dbUser $nrWarehouses $backupFile $nrClients "#checkpoint_completion_target"
    =" "checkpoint_completion_target = 0.4" checkpoint/
    checkpoint_completion_target checkpoint_completion_target_04

57
58 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-19
    $dbUser $nrWarehouses $backupFile $nrClients "#checkpoint_completion_target"
    =" "checkpoint_completion_target = 0.6" checkpoint/
    checkpoint_completion_target checkpoint_completion_target_06

59
60 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-20
    $dbUser $nrWarehouses $backupFile $nrClients "#checkpoint_completion_target"
    =" "checkpoint_completion_target = 0.8" checkpoint/
    checkpoint_completion_target checkpoint_completion_target_08

61
62 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-21
    $dbUser $nrWarehouses $backupFile $nrClients "#checkpoint_completion_target"
    =" "checkpoint_completion_target = 1.0" checkpoint/
    checkpoint_completion_target checkpoint_completion_target_10

63 #####
64 #          CHECKPOINT WARNING          #
65 #####
66

67
68 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-22
    $dbUser $nrWarehouses $backupFile $nrClients "#checkpoint_warning ="
    "checkpoint_warning = 0" checkpoint/checkpoint_warning checkpoint_warning_0

69
70 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-23
    $dbUser $nrWarehouses $backupFile $nrClients "#checkpoint_warning ="
    "checkpoint_warning = 10s" checkpoint/checkpoint_warning
    checkpoint_warning_10

71
72 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-24
    $dbUser $nrWarehouses $backupFile $nrClients "#checkpoint_warning ="
    "checkpoint_warning = 30s" checkpoint/checkpoint_warning
    checkpoint_warning_30s

73
74 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-25
    $dbUser $nrWarehouses $backupFile $nrClients "#checkpoint_warning ="
    "checkpoint_warning = 60s" checkpoint/checkpoint_warning
    checkpoint_warning_60s

75
76 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-checkpoint-26
    $dbUser $nrWarehouses $backupFile $nrClients "#checkpoint_warning ="
    "checkpoint_warning = 120s" checkpoint/checkpoint_warning
    checkpoint_warning_120s

```

Listing 26: Script Testes Automáticos Para *Checkpoints*

Script Testes Automáticos Para *Archiving*

```
1 #!/bin/bash
2
3 dbUser=$1
4 nrWarehouses=$2
5 backupFile=$3
6 nrClients=$4
7
8 ##### ARCHIVE MODE #####
9 #                         #
10 #####
11 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-archiving-1 $dbUser
12   $nrWarehouses $backupFile $nrClients "#archive_mode =\" "archive_mode = off"
13   archiving/archive_mode archive_mode_off
14 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-archiving-2 $dbUser
15   $nrWarehouses $backupFile $nrClients "#archive_mode =\" "archive_mode = on"
16   archiving/archive_mode archive_mode_on
17 ~/scripts/auxiliary_scripts/autorun_optimizations.sh server-archiving-3 $dbUser
18   $nrWarehouses $backupFile $nrClients "#archive_mode =\" "archive_mode =
19   always" archiving/archive_mode archive_mode_always
```

Listing 27: Script Testes Automáticos Para *Archiving*

Script Testes Automáticos Para Combinações

```
1 #!/bin/bash
2
3 dbUser=$1
4 nrWarehouses=$2
5 backupFile=$3
6 nrClients=$4
7
8 ##### SETTINGS COMBINATION #####
9 #           SETTINGS COMBINATION      #
10 #####
11
12 # Run 1
13 ~/scripts/auxiliary_scripts/autorun_combinations.sh server-combinations-1
14     $dbUser $nrWarehouses $backupFile $nrClients "settings_combination.conf"
15         settings/combination settings_combination_1
16
17 # Run 2
18 ~/scripts/auxiliary_scripts/autorun_combinations.sh server-combinations-2
19     $dbUser $nrWarehouses $backupFile $nrClients "settings_combination.conf"
20         settings/combination settings_combination_2
21
22 # Run 3
23 ~/scripts/auxiliary_scripts/autorun_combinations.sh server-combinations-3
24     $dbUser $nrWarehouses $backupFile $nrClients "settings_combination.conf"
25         settings/combination settings_combination_3
26
27 ##### CHECKPOINT COMBINATION #####
28 #           CHECKPOINT COMBINATION      #
29 #####
30
31 # Run 1
32 ~/scripts/auxiliary_scripts/autorun_combinations.sh server-combinations-4
33     $dbUser $nrWarehouses $backupFile $nrClients "checkpoints_combination.conf"
34         checkpoint/combination checkpoint_combination_1
35
36 # Run 2
37 ~/scripts/auxiliary_scripts/autorun_combinations.sh server-combinations-5
38     $dbUser $nrWarehouses $backupFile $nrClients "checkpoints_combination.conf"
39         checkpoint/combination checkpoint_combination_2
40
41 # Run 3
42 ~/scripts/auxiliary_scripts/autorun_combinations.sh server-combinations-6
43     $dbUser $nrWarehouses $backupFile $nrClients "checkpoints_combination.conf"
44         checkpoint/combination checkpoint_combination_3
45
46 #####
```

```

35      #          FINAL COMBINATION          #
36      ##########
37
38 # Run 1
39 ~/scripts/auxiliary_scripts/autorun_combinations.sh server-combinations-7
  $dbUser $nrWarehouses $backupFile $nrClients "final_combination.conf" final/
    combination final_combination_1
40
41 # Run 2
42 ~/scripts/auxiliary_scripts/autorun_combinations.sh server-combinations-8
  $dbUser $nrWarehouses $backupFile $nrClients "final_combination.conf" final/
    combination final_combination_2
43
44 # Run 3
45 ~/scripts/auxiliary_scripts/autorun_combinations.sh server-combinations-9
  $dbUser $nrWarehouses $backupFile $nrClients "final_combination.conf" final/
    combination final_combination_3

```

Listing 28: Script Testes Automáticos Para Combinações

Ficheiro Com Exemplos de Utilização dos Scripts

```
1  #
# ######
#          COMANDOS SCRIPTS
#      #
# ######
4
5 # Copiar pasta scripts para servidor na google cloud
6 gcloud compute scp --recurse ./scripts/ server-joel:
7 gcloud compute scp --recurse ./scripts/ bench-joel:
8
9 ----- Testes Automáticos (Bench Server Side)
10
11 # Teste automático para os clientes (80 warehouses), dump.file precisa de ser o
12   path completo
13 ./scripts/autotest_clients.sh Joel 80 dump.file
14
15 # Teste automático para as settings (80 warehouses, 100 cli), dump.file precisa
16   de ser o path completo
17 ./scripts/autotest_settings.sh Joel 80 dump.file 100
18
19 # Teste automático para o archiving (80 warehouses, 100 cli), dump.file precisa
20   de ser o path completo
21 ./scripts/autotest_archiving.sh Joel 80 dump.file 100
22
23 # Teste automático para o checkpoint (80 warehouses, 100 cli), dump.file precisa
24   de ser o path completo
25 ./scripts/autotest_checkpoint.sh Joel 80 dump.file 100
26
27 ----- Database Server Side Scripts
28
29 # Opção -i para configurar de inicio (clean install), -d para usar
30   discos locais da Google
31 ./scripts/dbserverconfiguration.sh -a server-joel -n 10.128.0.0 -d -i
32
33 # Recupera o depois de dar Restart ao servidor na Google Cloud (só no tem
34   a opção -i)
35 ./scripts/dbserverconfiguration.sh -a server-joel -n 10.128.0.0 -d
```

```

34 # Apagar pasta data do PostgreSQL e criar uma nova e lan ar server , caso se
35     usem discos locais usar op o -d
36 ./ scripts/dbservercleanpostgresdata.sh -a server-joel -n 10.128.0.0 -d
37
38 # Fazer clean install do servidor de benchmark , EXECUTA o load.sh
39 ./ scripts/benchmarkconfiguration.sh -s server-joel -u Joel -w 80
40
41 # Limpar base de dados previamente configurada (drop tpcc) e EXECUTA o load.sh
42 ./ scripts/cleancreatedb.sh -s server-joel -u Joel -w 80
43
44 # Restaurar a base de dados a partir de um ficheiro de backup
45 ./ scripts/restoredb.sh -s server-joel -w 80 -b tpcc.dump
46
47 # Correr o script de transa es (run.sh) dado servidor , user da base de dados
48     e o numero de warehouses e clientes
49 # Vai alterar o database.properties do tpcc, antes n o o fazia e quando
50     quer amos testar num server diferente do inicial foi necess rio isto.
51 ./ scripts/runclients.sh -s server-joel -u Joel -w 80 -c 100
52
53 # ENF OF COMANDOS SCRIPTS
54 #

```

Listing 29: Ficheiro Com Exemplos de Utilização dos Scripts