AlloyDB Omni for PostgreSQL - Transactional (OLTP) Benchmarking Guide

Sep 2023

Disclaimer

This AlloyDB Omni benchmark guide provides best practices for running an Online Transactional Processing (OLTP) benchmark. Your results may vary depending on several factors including, but not limited to the machine specifications of your AlloyDB Omni instance, type of client machine driving the benchmark, region, zone, and network bandwidth at the time of tests. Nothing in this user guide should be construed as a promise or guarantee about the results you'll derive from measuring the OLTP performance of AlloyDB Omni.

Overview

AlloyDB Omni is a downloadable edition of AlloyDB, designed to run anywhere — in your data center, on your laptop, at the edge, and in any cloud. AlloyDB Omni has several components and features, such as state-of-the-art log and transaction management, dynamic memory management, and a built-in columnar engine. As a whole, these features enable high performance for your transactional (OLTP), analytical (OLAP), and hybrid (HTAP) workloads.

Relational database systems typically require a database administrator (DBA) to optimize them for benchmarking, which includes configuring the transaction log settings, establishing the right buffer pool sizes, and tweaking other important database parameters (flags) and characteristics. These settings also vary based on machine hardware.

During installation, AlloyDB Omni chooses settings that are likely to be optimal for the number of CPUs and memory on your system. It requires minimal to no tuning of flags at the database level to achieve high OLTP performance. Users may further adjust the settings to optimize performance for their specific workload.

This document describes step-by-step procedures and best practices to configure AlloyDB Omni, a client machine, and scripts to setup, load and run benchmarks. We will be running HammerDB TPROC-C (derived from TPC-C) with different test parameters.

> NOTE: Since HammerDB's TPROC-C implementation is a close variant of the official TPC-C benchmark, we will use the terms TPC-C and TPROC-C interchangeably throughout this user guide.

Benchmarking Process

We'll go through the following steps to set up and run various OLTP benchmarks.

1. Configure AlloyDB Omni running on a Google Compute Engine (GCE) VM.
2. Setup of a separate benchmark driver client virtual machine running on GCE, where we will install benchmarking tools.
3. Install HammerDB on the client machine.
4. Run TPC-C like benchmark using HammerDB.

Unless otherwise specified, we used the following setup for performance benchmarking:

Component	Value
AlloyDB Omni Machine Type	n2-highmem-16 16vCPU / 128GB / 2048GB Persistent Disk
AlloyDB Omni Version	`15.2.0` (This is the latest version at the time of writing of this document.)
Region	us-central1 (Iowa)
Zone	us-central1-a

Client VM — Machine Type	E2-standard-32 / 128GB / 128 GB persistent disk as boot disk Operating System: Debian linux
Zone of Client VM	us-central1-a [same as AlloyDB Omni instance]
Connectivity	Private IP (same VPC)
Test tools	HammerDB-4.6 Psql

Workloads	TPC-C benchmark on 16 and 64 vCPU machines in following modes:
	    (1) 30% data on cache
	    (2) 100% data on cache
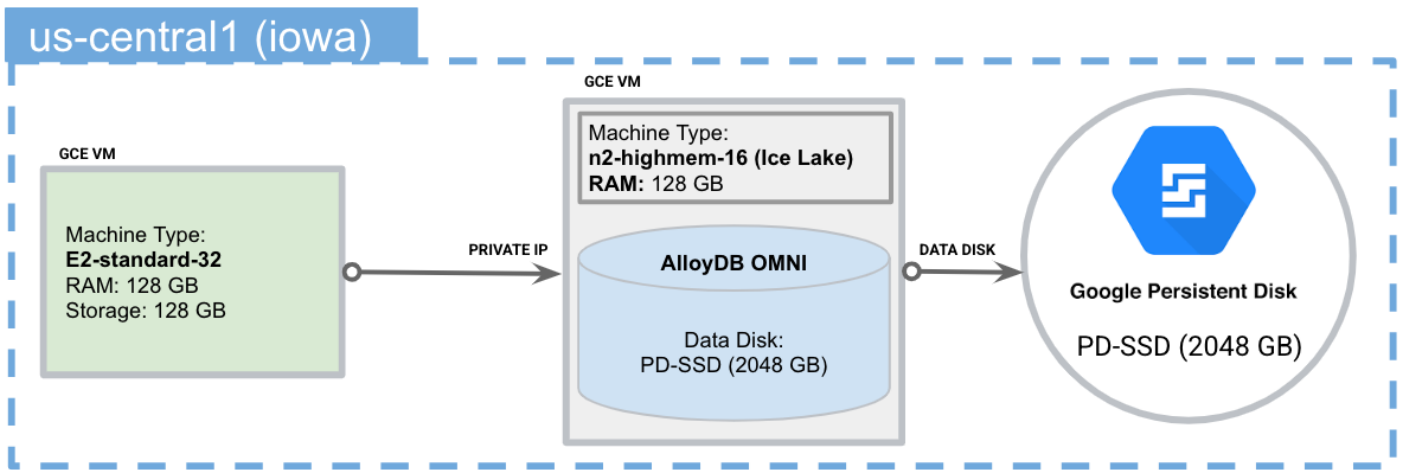
In your own testing, you can also run AlloyDB Omni on other platform configurations (as long as they meet these system requirements). Your benchmarking results will vary based on your specific hardware. Some crucial factors that can affect performance include the CPU model, number of cores/vCPUs, available memory, disk performance (IOPS and throughput), and network performance (latency and bandwidth) between the server and client.

Infrastructure Setup

Provision the server and client VMs

Note: The next section describes how to provision the VMs through the GCP cloud console. You may skip this section if running on your own hardware.

Provision Server on GCE

1. Create or select your GCP project: Go to https://console.cloud.google.com and select your project from the drop down menu or create a new one.

2. Follow these links on the portal: "Products and Solutions" → "All Products" → "Compute Engine".

3. Click on the following button to create instance to run AlloyDB Omni.

4. Choose a name for your server VM, and select your desired region and zone.

5. Under "Machine Configuration", select "N2" for "Series", and "n2-highmem-16" for the "Machine type".

Machine configuration

✓ **General purpose**   **Compute optimized**   **Memory optimized**   **GPUs**

Machine types for common workloads, optimized for cost and flexibility

> 💡 Try the new C3 machine series. There's no charge for C3 VMs during public preview.

Series
N2   ▼

Powered by Intel Cascade Lake and Ice Lake CPU platforms

Machine type

Choose a machine type with preset amounts of vCPUs and memory that suit most workloads. Or, you can create a custom machine for your workload's particular needs. Learn more

PRESET   **CUSTOM**

---
n2-highmem-16 (16 vCPU, 128 GB memory)   ▼
---

	vCPU	Memory
	16	128 GB

∨ **ADVANCED CONFIGURATIONS**

Display device

Enable to use screen capturing and recording tools.

☐ Enable display device

6. For best performance, expand "Advanced Configurations" and select "Intel Ice Lake or later".

7. Under "Boot disk", ensure you are using a Debian 11 image, and have at least 20 GB provisioned for the boot disk.

8. Under "Observability - Ops Agent", select "Install Ops Agent for Monitoring and Logging". This agent helps gather system metrics during the benchmark run.

9. Next, we create a separate disk which will be used by the database. Under "Advanced options" →
   "Disks", select "Add new disk".

Advanced options ⌃

Networking ⌄
Hostname and network interfaces

Disks ⌃
Additional disks

+ ADD NEW DISK   + ATTACH EXISTING DISK   + ADD LOCAL SSD

Security ⌄
Shielded VM and SSH keys

Management ⌄
Description, deletion protection, reservations, automation, and availability policies

---

Sole-tenancy ⌄
Node affinity labels and CPU overcommit

10. In the sidebar, ensure "Disk type" is set to "SSD persistent disk", and "Size" to "2048" GB. Persistent
    disks have per GB and per instance performance limits for the maximum IOPS and throughput that
    they can sustain, so we recommend a large disk size for better disk performance.

Add new disk ✕

Name *

omni-server-16vcpu-disk ❓

Name is permanent

Description

Source

Create a blank disk, apply a bootable disk image, or restore a snapshot of another disk in this project.

Disk source type *

Blank disk ▾

Disk settings

Disk type *

SSD persistent disk ▾ ❓

COMPARE DISK TYPES

Size *

2048 GB ❓

Provision between 10 and 65,536 GB

11. Finally, towards the bottom of the sidebar, ensure "Mode" = "Read/write", "Deletion rule" = "Delete disk", and use a custom device name "alloydb-disk". Then you may click "Save" to finish the disk setup.

Attachment settings

Mode
Disk attachment mode

- ⦿ Read/write
- ○ Read-only

Deletion rule
When deleting instance

- ○ Keep disk
- ⦿ Delete disk

Device name ❓
Used to reference the device for mounting or resizing.

☑ Use a custom device name

Device name *
alloydb-disk

Custom

You're creating an unformatted disk. Format the disk after you attach it to your VM instance. Formatting and mounting a zonal persistent disk

This new disk will be added once you create the new instance.

SAVE   CANCEL

12. Now click "Create" at the bottom of the create instance page, and a new VM will begin to be provisioned for you. Wait until the VM is fully created, which will be indicated by a green check mark under the "Status" column.

VM instances

≡ Filter omni ⊗ Enter property name or value

	Status	Name ↑	Zone	Creation time	In use by	Internal IP	External IP
☐	✅	omni-server-16vcpu	us-central1-a	███████		██████	███████

Set up filesystem on server

Connect to the server VM using the **"gcloud compute ssh"** command. Refer this documentation for details "https://cloud.google.com/compute/docs/connect/standard-ssh".

Sample command:

```
gcloud compute ssh --zone "<primary zone>" "<server machine name>"  --project "<google-project>"
```

After connecting to the VM, run the following commands:

```
sudo mkdir -p /home/$USER/alloydb-data
sudo mkfs.ext4 -m 1 -F "/dev/disk/by-id/google-alloydb-disk"
sudo mount --make-shared -o noatime,discard,errors=panic "/dev/disk/by-id/google-alloydb-disk"
"/home/$USER/alloydb-data"
```

You can verify that you have formatted and mounted the PD correctly by running `lsblk -o NAME,MOUNTPOINT,FSTYPE,SIZE /dev/disk/by-id/google-alloydb-disk`:

```
$ lsblk -o NAME,MOUNTPOINT,FSTYPE,SIZE /dev/disk/by-id/google-alloydb-disk


NAME MOUNTPOINT                 FSTYPE SIZE
sdb  /home/$USER/alloydb-data ext4     2T
```

Note the line that says `sdb /home/$USER/alloydb-data ext4 2T`: It means you have successfully formatted the PD with an ext4 filesystem, it is accessible through the path `/home/$USER/alloydb-data`, and it has 2TB capacity.

Provision Client on GCE

To run the OLTP benchmarks, you will require a client machine with enough processing power. The benchmark driver HammerDB runs in a highly parallel fashion and consumes a significant amount of CPU. The client machines' configurations are chosen in a way that they should not be a bottleneck for the experiment.

For HammerDB TPC-C benchmark: An E2-standard-32 machine (32 vCPUS, 128 GB memory) and 128 GB disk as a client for driving TPC-C benchmark.

Important: For this exercise, the client must be provisioned in the same region, zone, and VPC as AlloyDB Omni's primary instance. Benchmarking tools directly access the AlloyDB Omni instance over private IP. This setup reduces network latency between the server and client.

Below is a sample client machine we provisioned to execute the TPC-C benchmark on an AlloyDB Omni primary instance with 16 virtual CPUs.

Basic information

Name	██████████████
Instance Id	
Description	None
Type	Instance
Status	✓ Running
Creation time	May 10, 2023, 8:57:31 PM UTC-07:00
Zone	us-central1-a ← Same as AlloyDB Omni instance's zone
Instance template	None
In use by	None
Reservations	Automatically choose
Labels	None
Tags ❓	████████████████████
	✏️
Deletion protection	Disabled
Confidential VM service ❓	Disabled
Preserved state size	0 GB

Machine configuration

Machine type	e2-standard-32
CPU platform	Intel Broadwell
Architecture	x86/64
vCPUs to core ratio ❓	—
Custom visible cores ❓	—
Display device	Disabled
	Enable to use screen capturing and recording tools
GPUs	None

Networking

Public DNS PTR Record	None
Total egress bandwidth tier	—
NIC type	—

→ VIEW IN NETWORK TOPOLOGY

Install AlloyDB Omni

Follow the steps in "Install AlloyDB Omni on the VM" to install AlloyDB Omni.

Allow access from the client VM

Edit the file `/var/alloydb/config/pg_hba.conf`. This file controls which clients may connect to the database, and we need to add an entry for the client. For example, if your client's IP address is `1.2.3.4`, you will add a line at the end of `pg_hba.conf` like this:

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
# "local" is for Unix domain socket connections only.
# Don't allow any unix socket connections as the alloydbadmin.
local   all             alloydbadmin                            reject
local   all             all                                     md5
# IPv4 local connections:
host    all             all             127.0.0.1/32            trust
# IPv6 local connections:
host    all             all             ::1/128                 trust
# Allow replication connections on localhost, from a user with the replication privilege.
local   replication     all                                     md5
host    replication     all             127.0.0.1/32            trust
host    replication     all             ::1/128                 trust
host    all             all             1.2.3.4/32              trust  # <-- ENTRY FOR CLIENT
```

> NOTE: In this guide, we use the "trust" setting to simplify the benchmarking setup. However, note that "trust" bypasses password protection, and should not be used for a production instance.

Update database configuration

Finally, edit the file `/var/alloydb/config/postgresql.conf`. This file sets the configurations for the database. The default configuration values are already tuned for your system's vCPU and RAM, but in our benchmarking, we need to increase the value of `max_connections=2000` to accommodate more clients in the larger tests.

Start AlloyDB Omni

Now we may restart AlloyDB Omni to pick up the updated configurations:

```
sudo alloydb database-server stop
sudo alloydb database-server start
```

If you do not see any errors, that means AlloyDB Omni is running. Verify by connecting to the database locally:

```
sudo docker exec -it pg-service psql -h localhost -U postgres
```

You should see a psql prompt, which means you have successfully connected:

```
sudo docker exec -it pg-service psql -h localhost -U postgres
psql (15.2)
Type "help" for help.

postgres=#
```

Type `quit` to exit psql.

Setup of Benchmark Driver Machine (Client)

This section will guide you through the steps of configuring the client machine, where we will install benchmarking tools such as HammerDB.

Connect to the client machine using the **"gcloud compute ssh"** command.

Sample command:

```
gcloud compute ssh --zone "<primary zone>" "<client machine name>"  --project "<google-project>"
```

Install PostgreSQL client

You will need a `psql` client application to connect to AlloyDB Omni. Use the following command to install a `postgresql` client that includes a `psql` application and then ensure you are able to connect.

```
sudo apt-get update
sudo apt install -y postgresql-client
```

Now ensure that it works and you are able to connect to the AlloyDB Omni. Use the "Private IP" address of your AlloyDB Omni instance.

```
psql -h <Private IP> -U postgres
```

Install HammerDB-4.6 Driver for TPC-C benchmark

For this benchmarking guide, we utilized HammerDB-4.6 driver. Execute the following commands to install HammerDB driver.

```
mkdir hammerdb
pushd hammerdb
curl -OL
https://github.com/TPC-Council/HammerDB/releases/download/v4.6/HammerDB-4.6-Linux.tar.gz
tar zxvf HammerDB-4.6-Linux.tar.gz
```

> NOTE: If your operating system is Non-Debian, perform the following checks to ensure you have all essential libraries to run hammerdb. For debian, the steps below are optional.

Next, run `hammerdb/HammerDB-4.6` directory:

```
cd HammerDB-4.6
sudo ./hammerdbcli
```

This puts you in the HammerDB shell. From there, run `librarycheck`:

```
HammerDB CLI v4.6
Copyright (C) 2003-2022 Steve Shaw
Type "help" for a list of commands
Initialized SQLite on-disk database /tmp/hammer.DB using existing tables (45,056 KB)
hammerdb> librarycheck
```

In the output, look for the section that says `Checking database library for PostgreSQL`, and ensure it succeeds. Otherwise, fix any errors that show up. On some platforms, you may need to install additional packages:

```
Debian-based system: sudo apt-get update && apt-get install -y libpgtcl
Red hat: sudo yum install tcl libpq
```

If the check was successful, you should see output like this:

```
~/hammerdb/HammerDB-4.6$ ./hammerdbcli
HammerDB CLI v4.6
Copyright (C) 2003-2022 Steve Shaw
Type "help" for a list of commands
Initialized new SQLite on-disk database /tmp/hammer.DB
hammerdb>librarycheck
<... snipped ...>
Checking database library for PostgreSQL
Success ... loaded library Pgtcl for PostgreSQL
<... snipped ...>
```

Notes on performance benchmarking

Benchmark Cleanup

This step is important if you are planning to execute multiple benchmarks in succession. Performing a proper cleanup between each benchmark is a critical prerequisite for accurate and reliable benchmarking results. This includes deleting previous benchmark data (i.e. benchmark database), and rebooting the AlloyDB Omni instance (that clears caches at database and operating systems level) before running another benchmark. A proper benchmark cleanup ensures that residual effects from previous benchmarks do not affect the performance measurements of the new benchmark. It also helps to ensure consistency and repeatability of the benchmark results, which is essential for making meaningful comparisons between different systems or identifying areas for optimization in hardware, software, or configuration.

Follow the URL https://cloud.google.com/compute/docs/instances/stop-start-instance to learn more about how to reboot a GCE VM.

To drop the previous benchmark database, you can use the following psql command from the client machine.

```
psql -h <Private IP> -U postgres -c "DROP DATABASE IF EXISTS <database_name>;"
```

Understanding system performance

Since AlloyDB Omni can be run on many different environments, it is important to know that the transaction performance is highly dependent on CPU/Memory/IO/Network latency.

1. When most data fits in memory, it is a CPU bound workload, and more CPUs will get more transaction performance.
2. When most data can not fit in memory, it becomes an IO bound workload, more disk IOPS/throughput will get more transaction performance. IO latency is also important for OLTP workload, when a transaction commits, it needs to flush WAL to disk before commit, so IO latency is directly related to commit latency.
3. Query latency is affected by network latency between client and server communication. It is recommended to have the client and server located in the same local network or same zone for benchmarking purposes.

Before benchmarking, It is useful to be able to characterize system performance of the hardware. In this section, we list down some commands that can be used to measure:

1. Performance of the CPU
2. Performance of the disk
3. Network latency between client and server

CPU performance

CPU performance can be measured by sysbench benchmark, see https://github.com/akopytov/sysbench for installation instructions.
Use the following command to measure cpu performance:

```
sysbench cpu --cpu-max-prime=10000 --threads=<Number of vCPUs> run
```

Disk performance

Fio can be used to measure disk performance.

Use the following commands to measure IOPS, throughput and latency.
IOPS

```
fio --time_based --runtime=60s --ramp_time=2s --ioengine=libaio --direct=1 --name=iops_test
--filename=/mnt/disks/pgsql/fio_test --bs=8k --iodepth=256 --size=4G --readwrite=randrw
--rwmixread=25 --verify=0 --group_reporting=1
```

Write Throughput

```
fio --name=write_throughput --filename=/mnt/disks/pgsql/fio_test --numjobs=16 --size=4G
--time_based --runtime=60s --ramp_time=2s --ioengine=libaio --direct=1 --verify=0 --bs=256k
--iodepth=256 --rw=randwrite --group_reporting=1
```

Latency

```
fio --time_based --runtime=60s --ramp_time=2s --ioengine=libaio --direct=1 --name=latency_test
--filename=/mnt/disks/pgsql/fio_test --bs=256k --iodepth=1 --size=4G --readwrite=randwrite
--verify=0
```

Network latency

Ping can be used to measure network latency.

```
ping <IP address> -c 100
```

TPC-C Benchmark

[HammerDB](#) is a popular benchmarking tool that includes a [TPC-C](#) benchmark implementation for evaluating the performance of OLTP systems. HammerDB's TPC-C implementation allows users to simulate a workload similar to the TPC-C benchmark, including a mix of transactions that mimic the behavior of a wholesale supplier environment. HammerDB measures the system's performance in terms of transactions per minute (TPM) and generates reports that include detailed statistics and performance metrics. Additionally, HammerDB supports customization of the benchmark parameters, allowing users to adjust the database size, the number of warehouses, and other workload characteristics to simulate different scenarios.

This section provides a comprehensive guide on how to execute the HammerDB TPC-C benchmark to gauge the performance of the AlloyDB Omni database system.

Prerequisites

A. You need to run the following steps from a client (driver) machine. Ensure that you have completed the setup steps listed in the "[Setup of Benchmark Driver Machine (Client)](#)" section (especially installation of the HammerDB utility).

B. **Cleanup**: If you are running multiple benchmarks in succession, ensure you follow the "[Benchmark Cleanup](#)" section before doing your subsequent run.

Initial Setup on Client Machine

Execute all commands from `hammerdb/HammerDB-4.6` directory.

```
cd hammerdb/HammerDB-4.6
```

Then create `setup.env` file as follows:

```
cat << EOF > setup.env

# Private IP of the AlloyDB primary instance
export PGHOST=111.222.333.444

# Postgres default port address. You do not need to change it unless you use non-default port
address.
export PGPORT=5432    # default port to connect with postgres

# Number of TPC-C warehouses to load. This determines the overall database size.
export NUM_WAREHOUSE=576

# Number of users for running the benchmark.
export NUM_USERS=256
EOF
```

Edit the generated `setup.env` file and change all the <mark>highlighted</mark> parameter values to those that are suitable to your environment setup.

For the purpose of this benchmarking guide, we evaluate the performance in the following two crucial scenarios:

1. **Partially (~30%) cached mode:** In this mode, we generate a large TPC-C database which can only partially fit in the buffer cache. The transactions in this mode will not be always served from memory and will incur IO to the underlying storage subsystems. This scenario is more realistic to the OLTP needs of the majority of customers with large data set.

   To test this scenario, change **NUM_WAREHOUSE** as **3200** in the setup.env file.

2. **Fully (100%) cached mode,** where the TPC-C database fully fits in the buffer cache. AlloyDB Omni utilizes approximately 90% of the available 128 GB RAM including buffer cache. Since TPC-C transactions perform minimal IO's (as reads are mostly served from buffer cache) in this mode, higher TPM is expected compared to partially-cached runs.

   To test this scenario, change **NUM_WAREHOUSE** as **576** in the setup.env file.

NOTE: The number of users (or clients) is set to 256 for this test. This number of users has been tuned to provide the best throughput with acceptable latency on both of these configurations.

Script to load TPC-C data

In the context of the TPC-C benchmark, a "**load step**" refers to the process of populating the benchmark database with initial data before running the actual performance test.

During this step, the database is populated with a specified number of warehouses, customers, and other entities according to the TPC-C specifications. The purpose of the load step is to create a realistic workload for the performance test, and to ensure that the test results are comparable across different systems.

After the load step is completed, the database is pre-populated with a defined set of initial data, and ready to be used for the TPC-C benchmark test.

Follow the steps below to load the TPC-C database:

1. Switch to the benchmark home directory.

```
cd hammerdb/HammerDB-4.6
```

2. Create **build-tpcc.sh** file as follows:

```
#!/bin/bash -x

source ./setup.env

./hammerdbcli << EOF

# CONFIGURE PARAMETERS FOR TPCC BENCHMARK
# --------------------------------------
dbset db pg
dbset bm tpc-c

# CONFIGURE POSTGRES HOST AND PORT
# --------------------------------------
diset connection pg_host $PGHOST
diset connection pg_port $PGPORT

# CONFIGURE TPCC
# --------------------------------------
diset tpcc pg_superuser postgres
diset tpcc pg_user tpcc
diset tpcc pg_dbase tpcc

# SET NUMBER OF WAREHOUSES AND USERS TO MANAGE EACH WAREHOUSE
# THIS IMPORTANT METRIC ESTABLISHES THE DATABASE SCALE/SIZE
# --------------------------------------
diset tpcc pg_count_ware $NUM_WAREHOUSE
diset tpcc pg_num_vu 10

# LOG OUTPUT AND CONFIGURATION DETAILS
# --------------------------------------
vuset logtotemp 1
print dict

# CREATE AND POPULATE DATABASE SCHEMA
# --------------------------------------
buildschema

vudestroy
```

```
quit

EOF
```

3. Execute the load command as shown below and wait for the command to finish. During this
   command, you may view the contents of `results/build-tpcc.out` in a second terminal to see its
   progress.
   - For the 100% cached test, this command should return in under an hour.
   - For the 30% cached test, this command could take several hours.

```
chmod +x ./build-tpcc.sh
mkdir -p results
ulimit -n 32768
sudo nohup ./build-tpcc.sh > results/build-tpcc.out 2>&1
```

4. **Validate Load**: After the aforementioned script completes, it would be advisable to confirm that the
   database load was successful. The database's size can be quickly verified by doing as follows:

```
$ psql -h $PGHOST -p 5432 -U postgres
postgres=> \l+ tpcc
                                                                 List of
databases
     Name      |       Owner       | Encoding | Collate |  Ctype  |             Access
privileges               |  Size   | Tablespace |               Description

-------------+------------------+----------+---------+---------+--------------------
----------------+---------+-----------+-----------------------------------------
 tpcc         | tpcc             | UTF8     | C.UTF-8 | C.UTF-8 |
             | --- GB  | pg_default |
```

In 30% cached TPC-C configuration (with 3200 warehouses), expect the size of the TPC-C database to
be around **300** GB.

In 100% cached TPC-C configuration (with 576 warehouses), expect the size of the TPC-C database to
be around **55** GB.

Running the TPC-C benchmark

In this step, we will initiate the actual TPC-C performance test. The TPC-C benchmark will be executed
using the populated database from the load step. The benchmark generates a series of transactions that
simulate a typical business environment, including order entry, payment processing, and inventory
management. The workload is measured in "transactions per minute" (**TPM**), which represents the number of
complete business transactions that the system can handle in one minute.

The run step is designed to stress the database system under realistic conditions and provide a standard way of measuring performance that can be compared across different database systems. Vendors and customers widely use the results of the TPC-C benchmark to evaluate the performance of different database systems and hardware configurations.

The following script will run the TPC-C benchmark for about 1 hour after approximately 10 minutes of warm up.

1. Switch to benchmark home directory:

```
cd hammerdb/HammerDB-4.6
```

2. Create **run-tpcc.sh** script as follows:

```
#!/bin/bash -x

source ./setup.env

./hammerdbcli << EOF
dbset db pg
dbset bm tpc-c

# CONFIGURE PG HOST and PORT
# --------------------------
diset connection pg_host $PGHOST
diset connection pg_port $PGPORT

# CONFIGURE TPCC DB
# --------------------------
diset tpcc pg_superuser postgres
diset tpcc pg_user postgres
diset tpcc pg_dbase tpcc

# BENCHMARKING PARAMETERS
# --------------------------
diset tpcc pg_driver timed
diset tpcc pg_rampup 10
diset tpcc pg_duration 60
diset tpcc pg_vacuum false
diset tpcc pg_partition false
diset tpcc pg_allwarehouse true
diset tpcc pg_timeprofile true
diset tpcc pg_connect_pool false
diset tpcc pg_dritasnap false
diset tpcc pg_count_ware $NUM_WAREHOUSE
diset tpcc pg_num_vu 1

loadscript
print dict
```

```
vuset logtotemp 1
vuset vu $NUM_USERS
vucreate
vurun
quit
EOF
```

3. Run the script as follows:

```
chmod +x run-tpcc.sh
mkdir -p results
ulimit -n 32768
sudo nohup ./run-tpcc.sh > results/run-tpcc.out 2>&1
```

Now wait for the `run-tpcc.sh` script to finish. The script will take approximately 1 hour and 10 minutes to complete.

Analyzing TPC-C Results

In the context of the TPC-C benchmark, **NOPM** and **TPM** are performance metrics used to measure the performance of a database system. **NOPM** stands for "**New Orders Per Minute**" and measures the number of new order transactions that the system can handle in one minute. The New Order transaction is one of the most important transactions in the TPC-C benchmark and involves creating a new order for a customer.

TPM stands for "**Transactions Per Minute**" and measures the total number of completed business transactions that the system can handle in one minute. This includes not only *New Order* transactions but also *Payment*, *Delivery*, *Order Status*, and other types of transactions defined in the TPC-C benchmark.

In general, TPM is considered to be the primary performance metric for the TPC-C benchmark, as it provides an overall measure of the system's ability to handle a realistic workload. However, NOPM can also be a useful metric for systems that are heavily focused on processing new orders, such as e-commerce or retail systems.

Measured Results With AlloyDB Omni

With 30% cached TPC-C database on 16 vCPU machine (i.e. NUM_WAREHOUSE=3200 and NUM_USERS=256), we observed **106804** tpm-C (New Order Per Minute) from a cumulative **245666** TPM. These performance numbers can be extracted using following command:

```
$ grep NOPM results/run-tpcc.out
Vuser 1:TEST RESULT : System achieved 106804 NOPM from 245666 PostgreSQL TPM
```

On a 100% cached TPC-C database on 16 vCPU machine (i.e. NUM_WAREHOUSE=576 and NUM_USERS=256), we got 436429 tpm-C (New Order Per Minute) from a cumulative 1011352 TPM :

```
$ grep NOPM results/tpcc-run.out
Vuser 1:TEST RESULT : System achieved 436429 NOPM from 1011352 PostgreSQL TPM
```

Summary of performance results on 16 vCPU.

TPC-C Scenario	NUM_WAREHOUSE	NUM_USERS	New Order Per Minute (NOPM)	Cumulative TPM
30% cached	3200	256	106893	245878
100% cached	576	256	414917	958875

Observability

To further understand the behavior of the database system, you can use GCE monitoring page to monitor important system metrics, such as CPU usage, memory usage, etc. These monitoring information can be found by navigating to the "Compute Engine -> VM instances -> Instance" page and/or navigating to the **Observability** page on https://console.cloud.google.com.

For instance, the below picture shows the CPU/Memory/Disk/Network metrics of a GCE instance during the TPC-C run.
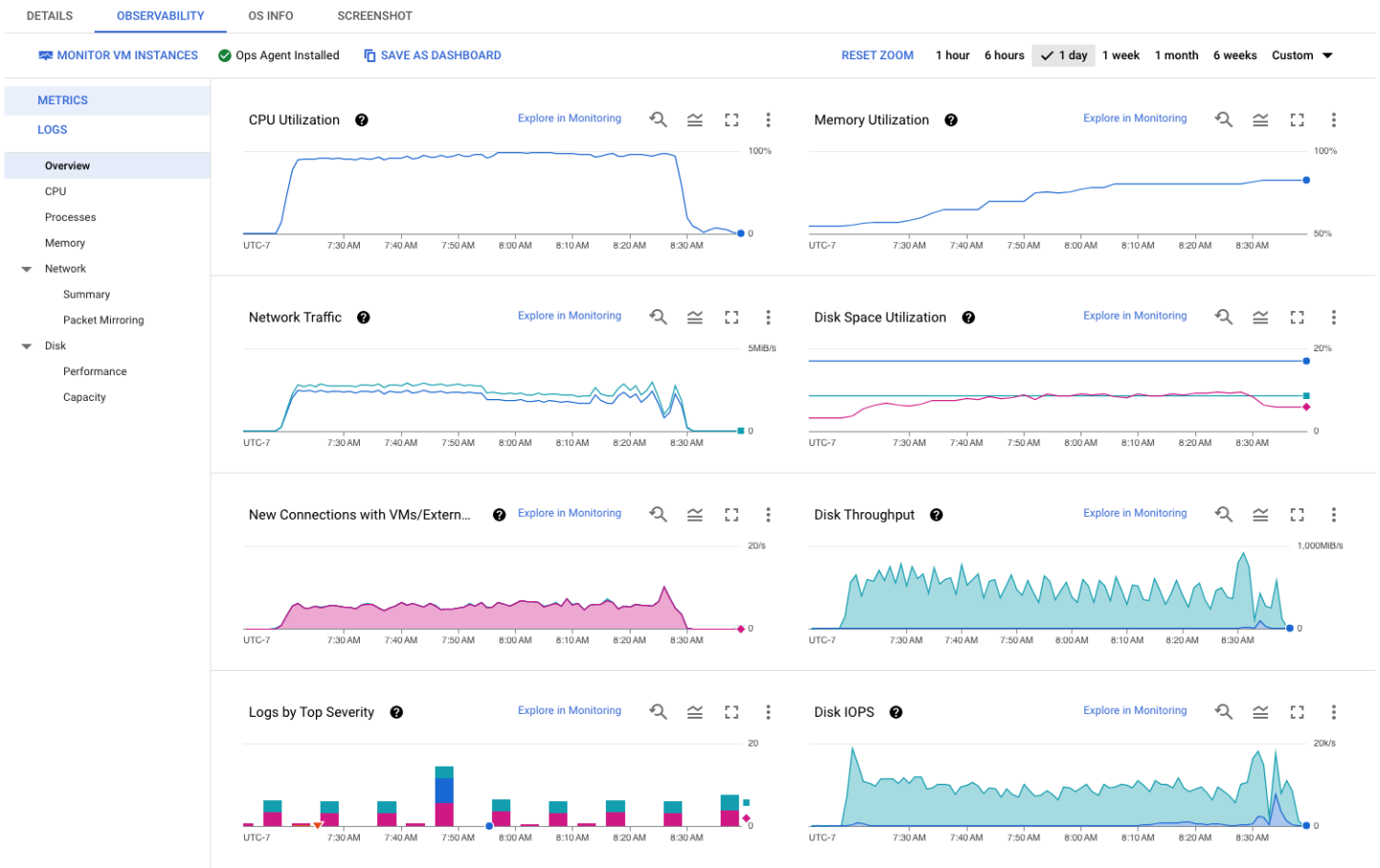
If you are running AlloyDB Omni on other hardware, you can use the `iostats` program to check real time CPU/IO stats.

```
iostat -m 10
```

This will print statistics about the I/O devices every 10 seconds, e.g.:

```
$ iostat -m 10
Linux 5.10.0-25-cloud-amd64 (omni-server-16vcpu)        09/13/23        _x86_64_        (16 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.77    1.75    1.11    0.26    0.00   96.12

Device             tps    MB_read/s    MB_wrtn/s    MB_dscd/s    MB_read    MB_wrtn    MB_dscd
sda              53.04         0.94         4.81         0.58        803       4104        496
sdb            1535.29         0.04       186.46      2466.31         31     159054    2103788


avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           1.69    6.17    3.22    1.81    0.00   87.10

Device             tps    MB_read/s    MB_wrtn/s    MB_dscd/s    MB_read    MB_wrtn    MB_dscd
sda               0.50         0.00         0.01         0.00          0          0          0
sdb            8297.00         0.00      1060.20       121.73          0      10602       1217

...
```

For details about the output of iostat, please refer to its [documentation](#).

TPC-C Benchmark on 64 vCPU AlloyDB Omni Instance

Infrastructure Setup using 64 vCPU Machine Type

AlloyDB Omni Setup

The overall instructions for the setup of AlloyDB Omni with 64 vCPU machine type are similar to the steps outlined in section "Provision the server and client VM". The only changes you need:
- In step 5: The `Machine Type` should be changed to `n2-highmem-64`.
- In step 9: The size of the disk should be increased to 5120GB, for best performance.

Client Machine Setup

To setup a client machine, you need to follow the steps outlined in "Provision Client Machine" except the `Machine Type` parameter that changes to `n2-standard-64` machine. Ensure that the client machine is located in the zone of AlloyDB Omni primary instance. Below is the screenshot of our client machine configuration.

Machine configuration

Machine type	n2-standard-64
CPU platform	Intel Cascade Lake
Architecture	x86/64
vCPUs to core ratio ❓	—
Custom visible cores ❓	—
Display device	Disabled
	Enable to use screen capturing and recording tools
GPUs	None

Networking

Public DNS PTR Record	None
Total egress bandwidth tier	—
NIC type	—

Then follow the instructions outlined in the section "Setup of Benchmark Driver Machine (Client)".

Running the benchmark

Follow the steps outlined below to run the benchmark:

1. Follow the "Prerequisites" section.

2. Then follow "Initial Setup on Client Machine" and use following parameter values:
   - Set `PGHOST` to the "Private IP" of your new 64 vCPU AlloyDB Omni instance.
   - For 30% Cached TPC-C scenario, set `NUM_WAREHOUSE=12800` and `NUM_USERS=1024`.
   - For 100% Cached TPC-C scenario, set `NUM_WAREHOUSE=2304` and `NUM_USERS=1024`.

3. To setup and load a TPC-C database, follow the "Load TPC-C script" section.
   NOTE: In order to speed-up the load, change the value of `pg_num_vu` to 64 in `build-tpcc.sh` as `diset tpcc pg_num_vu 64.`

4. Then follow the exact steps in "Running the TPC-C benchmark".

Results Observed

Benchmark Mode	NUM_WAREHOUSE	NUM_USERS	New Order Per Minute (NOPM)	Cumulative TPM
30% cached	12800	1024	227849	518515

| 100% cached | 2304 | 1024 | 1003868 | 2293709 |

Results Summary

This section is intended to provide a summary of our observations based on the benchmarks explained in this document.

HammerDB TPC-C Performance Summary

AlloyDB Omni Machine Type	TPC-C Workload Scenario	NUM_WAREHOUSE	NUM_USERS	New Order Per Minute (NOPM)	Cumulative TPM	Converted to TPS
16 vCPU	30% cached	3200	256	106893	245878	4098
16 vCPU	100% cached	576	256	414917	958875	15981
64 vCPU	30% cached	12800	1024	227849	518515	8642
64 vCPU	100% cached	2304	1024	1003868	2293709	38228