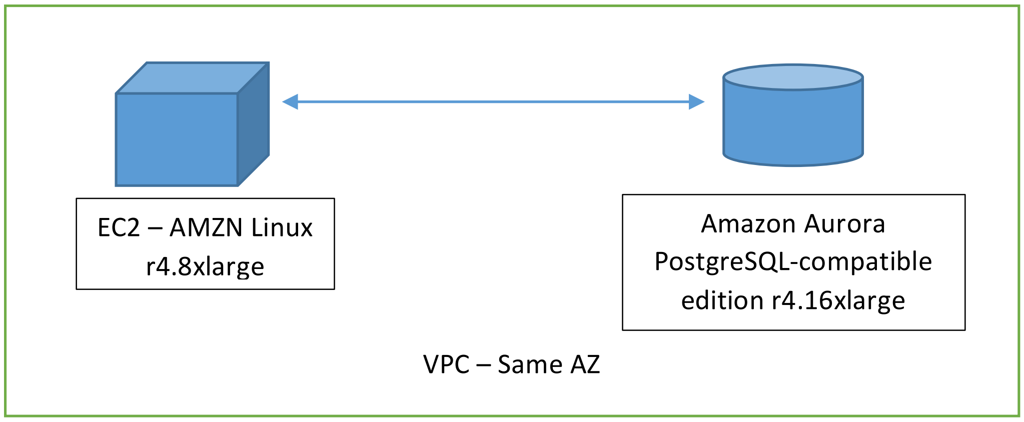
1. Introduction

This document outlines the steps to benchmark the performance of the PostgreSQL-compatible edition of Amazon Aurora using the *pgbench* and *sysbench* benchmarking tools. It describes how to run a number of different workloads to simulate a base load (*pgbench*), a read-write workload (*pgbench*), and a write-heavy workload (*sysbench*) on the PostgreSQL-compatible edition of Amazon Aurora.

Note that the results displayed in this document are only examples. Your actual results might vary, based on environmental dependencies such as instance configuration, region, network performance, etc.

2. Setup

For this document, the reference setup consists of one client machine (an AWS EC2 *r4.8xlarge* instance running *sysbench* or *pgbench*), querying an Amazon Aurora r4.16xlarge database instance in the same Availability Zone. These instances are created in an Amazon VPC with enhanced networking enabled, to ensure that throughput numbers are not constrained by network bandwidth.


EC2 – AMZN Linux r4.8xlarge
Amazon Aurora PostgreSQL-compatible edition r4.16xlarge
VPC – Same AZ

The setup tasks are as follows:

1. Create an Amazon VPC. For the purposes of this test, the EC2 instance (*r4.8xlarge with AMZN Linux*) is in the same Availability Zone as the database instance.
2. Ensure that enhanced networking is enabled on the EC2 instance (see http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/enhanced-networking.html).
3. Install *pgbench* and *sysbench* (version 0.5) on the EC2 instance. For detailed instructions, see Appendix A of this document.
4. Launch an *r4.16xlarge* instance of the PostgreSQL-compatible edition of Amazon Aurora database engine in the same VPC. Amazon RDS automatically enables enhanced networking.

3. Steps for Performance Testing

1. Set the necessary environment variables.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/pgsql/lib
export PATH=$PATH:/usr/local/pgsql/bin/
```

2. Run the *pgbench* initialization on your Amazon Aurora database instance. The following command loads a *pgbench* database using a scale factor of 10000, vacuums the resulting data, and then indexes it:

```
pgbench -i --fillfactor=90 --scale=10000 --host=<rds-aurora-instance-host-name> \
--username=<db-username> postgres
```

3. Run the *pgbench* read/write workload using the following command:

```
pgbench --host=<rds-aurora-instance-host-name> --username=<db-username>  \
--protocol=prepared -P 60 --time=3600 --client=2048 --jobs=2048 postgres
```

4. Prepare the *sysbench* benchmark data in your Amazon Aurora database instance. The following command will create the test data:

```
sysbench --test=/usr/local/share/sysbench/oltp.lua \
--pgsql-host=<rds-aurora-instance-host-name> --pgsql-db=postgres \
--pgsql-user=<db-username> --pgsql-password=<db-password> --pgsql-port=5432 \
--oltp-tables-count=250 --oltp-table-size=450000 prepare
```

5. Run the write workload on the EC2 *sysbench* client, using the following command:

```
sysbench --test=/usr/local/share/sysbench/oltp.lua \
--pgsql-host=<rds-aurora-instance-host-name> --pgsql-db=postgres \
--pgsql-user=<db-username> --pgsql-password=<db-password> --pgsql-port=5432 \
--oltp-tables-count=250 --oltp-table-size=450000 --max-requests=0 --forced-shutdown \
--report-interval=60 --oltp_simple_ranges=0 --oltp-distinct-ranges=0 \
--oltp-sum-ranges=0 --oltp-order-ranges=0 --oltp-point-selects=0 --rand-type=uniform \
--max-time=3600 --num-threads=2048 run
```

4. Viewing the Results

4.1 Data Load: *pgbench* Initial Load

The following *pgbench* output shows the results of the reference data load test performed on the EC2 instance. This test loads the *pgbench* setup (with a scale factor of 10000) into the database, and then performs vacuuming and indexing tasks. This results in a database with approximately 1 billion rows in 3 tables.

In our tests, the initial load required 813 seconds to load the data, 0.4 seconds to commit the data, 310 seconds to vacuum the loaded data, and 463 seconds to index the data. The total time required was 1,586 seconds. All of this activity was performed while maintaing six copies of the data in three different Availability Zones.

```
1000000000 of 1000000000 tuples (100%) done (elapsed 812.60 s, remaining 0.00 s)
vacuum...
set primary keys...
total time: 1586.05 s (insert 812.67 s, commit 0.41 s, vacuum 309.60 s, index 463.36 s)
done.
```

4.2 Read and Write Workload - *pgbench*

The following *pgbench* output shows the results of the read/write default test on the EC2 instance, running *pgbench,* with patches to allow more than 1,000 clients/thread. These patches are described in Appendix A. This read/write workload test performs one SELECT, three UPDATEs and one INSERT for each transaction.

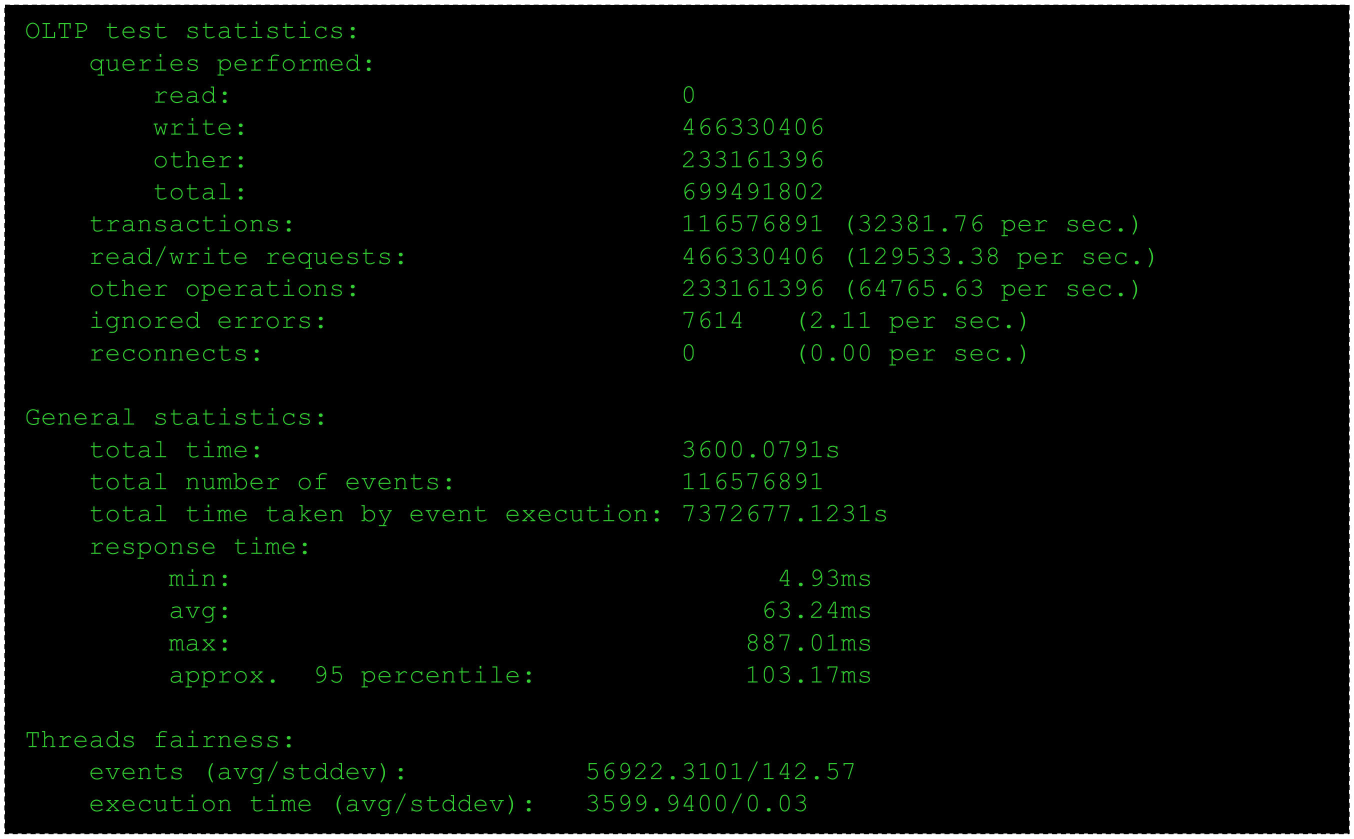
We ran this test for 3,600 seconds, and observed an average performance of 41,498 transactions per second. This resulted in approximately 41,498 reads per second, 124,494 updates per second and 41,498 inserts per second, for a total of 165,992 writes per second on average.

```
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 10000
query mode: prepared
number of clients: 2048
number of threads: 2048
duration: 3600 s
number of transactions actually processed: 149298601
latency average = 49.338 ms
latency stddev = 22.907 ms
tps = 41458.992207 (including connections establishing)
tps = 41497.938898 (excluding connections establishing)
```

4.3 Write Workload – *sysbench*

The following *sysbench* output shows the results of the write-heavy test running on the EC2 instance running *sysbench*.

We ran this test for 3600 seconds, and observed an average performance of 129,533 write requests per second, while maintaining six copies of data in three different Availability Zones. During this test, the system performed 32,381 transactions per second, involving INSERTs, index and non-index UPDATEs, and DELETEs.



```
OLTP test statistics:
    queries performed:
        read:                            0
        write:                           466330406
        other:                           233161396
        total:                           699491802
    transactions:                        116576891 (32381.76 per sec.)
    read/write requests:                 466330406 (129533.38 per sec.)
    other operations:                    233161396 (64765.63 per sec.)
    ignored errors:                      7614   (2.11 per sec.)
    reconnects:                          0      (0.00 per sec.)

General statistics:
    total time:                          3600.0791s
    total number of events:              116576891
    total time taken by event execution: 7372677.1231s
    response time:
         min:                                  4.93ms
         avg:                                 63.24ms
         max:                                887.01ms
         approx.  95 percentile:             103.17ms

Threads fairness:
    events (avg/stddev):           56922.3101/142.57
    execution time (avg/stddev):   3599.9400/0.03
```

Appendix A – Installation instructions for *pgbench* v9.6 and *sysbench* v0.5

For our testing, we used the following procedure to install and configure *pgbench* and *sysbench*:

1. By default, pgbench is limited to a maximum of 1,000 clients. To increase this limit, modify the */etc/security/limits.conf* file so that it contains the following entries:

```
ec2-user          hard    nofile          65000
ec2-user          soft    nofile          65000
ec2-user          hard    nproc           65000
ec2-user          soft    nproc           65000
```

   Note: To make these settings take effect, you must restart your session.

2. Install the required tools for building *pgbench* and *sysbench*:

```
sudo yum install ant git php gnuplot gcc make readline-devel zlib-devel \
postgresql-jdbc bzr automake libtool patch libevent-devel openssl-devel \
ncurses-devel
```

3. Install the base PostgreSQL libraries and the *pgbench* utility. In this step, you build and install the PostgreSQL libraries and binaries, including the following two *pgbench* patches:

   - *pgbench-init-timing.patch* - prints detailed timing information for the different steps of the initial *pgbench* load.
   - *pgbench-poll.patch* - allows *pgbench* to run more than 1000 sessions.

```
wget https://ftp.postgresql.org/pub/source/v9.6.5/postgresql-9.6.5.tar.gz
tar -xzf postgresql-9.6.5.tar.gz
cd postgresql-9.6.5
wget https://s3.amazonaws.com/aurora-pgbench-patches/pgbench-init-timing.patch
patch -p1 -b  < pgbench-init-timing.patch
wget https://s3.amazonaws.com/aurora-pgbench-patches/pgbench-poll.patch
patch -p1 -b < pgbench-poll.patch
./configure
make -j 4 all
sudo make install
cd ..
```

4. Build and install *sysbench* 0.5:

```
git clone -b 0.5  https://github.com/akopytov/sysbench.git
cd sysbench
./autogen.sh
CFLAGS="-L/usr/local/pgsql/lib/ -I /usr/local/pgsql/include/" | ./configure \
--with-pgsql --without-mysql --with-pgsql-includes=/usr/local/pgsql/include/ \
--with-pgsql-libs=/usr/local/pgsql/lib/
make
sudo make install
cd sysbench/tests
sudo make install
```