# Migration of Oracle 11.2.0.4 database to AWS Aurora PostgreSQL 11.6

Krishan Kumar
Sr Database Engineer
Dynata

## 1. Introduction

I would like to share my experience of migrating one of our largest databases from Oracle 11.2.0.4 Standard Edition to Aurora version 3.1.3 & PostgreSQL 11.6, as our business is growing so the instance size was not sufficient to support the growth and we cannot go higher instance size because of restrictions on number of vCPU for standard edition.
To support the business growth, we were having two options either we migrate to Oracle Enterprise Edition or move to Aurora based solution since migrating to Oracle Enterprise Edition is going to cost us lot more, so we decided to go with Aurora PostgreSQL based solution as we all know that converting Oracle stored procedure or functions to PostgreSQL is much easier than going to Aurora MySQL.
The objective was to migrate the database with minimal downtime and as smooth as possible.

## 2. Size of the database & schemas

| Schema | Prod | | QA | |
|---|---|---|---|---|
| | No of Tables | No of Indexes | No of Tables | No of Indexes |
| Schema 1 | 380,621 | 1,004,282 | 153,347 | 519,197 |
| Schema 2 | 163 | 80,676 | 161 | 2,526 |
| Schema 3 | 28,420 | 347,524 | 269 | 2,185 |
| Schema 4 | 2,044 | 8,125 | 560 | 2,227 |
| Schema 5 | 138 | 374 | 137 | 370 |
| **Total DB Size** | 11TB | | 500GB | |

## 3. Migration Strategy

We started brainstorming on the strategy how to migrate the stored procedures and data to Aurora PostgreSQL, since AWS provides the DMS (Data Migration Service) so migrating the data was made easier with this approach, but challenge was to convert stored procedures, table, indexes & triggers DDL.
We started following the AWS documents for database migration & best practices, so we decided to use AWS SCT (Schema Conversion Tool) to convert stored procedures, triggers, generate DDL for table & indexes.

## 4. Migration pre-work

We setup the SCT and start migrating the stored procedure, tables & indexes, SCT worked for smaller schemas but when it comes to **Schema 1** we could not able to get it working for tables & index conversion hence we have to come up with alternate solution to get the DDL generated.
So, we must come with home grown solution hence we used php script to generate DDL for table, sequence, index, trigger creations and store those DDLs in a table so that we can use/re-

use these DDLs anytime we need them in PostgreSQL. Time to time we were refreshing this table in Postgres for any object created after the last refresh.

We decided to use five DMS task for each schema, this approach worked for **Schema 2, 3, 4 & 5** but DMS task didn't even able to start data migration for **Schema 1** after running for many hours, as we know that there were 290k+ tables still left in this schema after archiving around 85K tables.
Since the number of tables were very huge hence Oracle logminer process were not able to get the information needed by DMS task to start the full load+CDC.

As the tables in **Schema 1** has set of seven different type of tables that means for any new respondent, we create those seven tables to record their activities, so structure of each individual table set is same only name is different as per value of the nnn variable i.e. table_set1_nnnn .. table_set7_nnnn
Also, the Schema 3 & 4 having new table created every time a new respondent comes to the system.

Now what next? remember I mentioned above that **Schema 1** has set of seven different types of tables so now we have created seven different tasks for each set of tables so one problem is solved but now how to manage load on source database for 11 tasks created to migrate all the schemas & data.

Since source instance has only 16vCPU & 128GB RAM and it needs to support daily business too, if I did not execute those 11 tasks in parallel then those tasks were going to take weeks to complete the full load, though all sets were not huge in size, but 2 sets were 1.5TB+ each. So decided to run 3-4 tasks in parallel to complete the full load as soon as possible.

## 5. Migration steps in detail

Please find the details of each step needed for the successful migration.

1. Prepare source database (**db.r5.4xlarg 16 vCPU, 128GB Memory**) as per AWS documents for CDC (Change Data Capture). Aws source https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Source.Oracle.html#CHAP_Source.Oracle.Amazon-Managed
    a. Enable supplemental logging

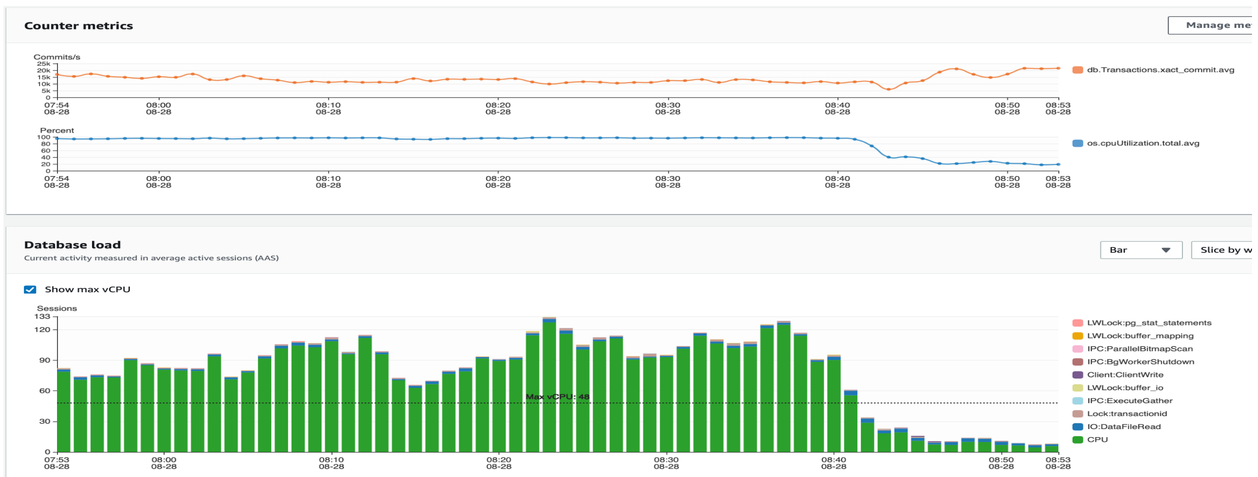
       exec rdsadmin.rdsadmin_util.alter_supplemental_logging('ADD','PRIMARY KEY');

    b. Change the retention for the archive logs so that we have all required logs when CDC process starts after the full load.
        exec rdsadmin.rdsadmin_util.set_configuration('archivelog retention hours',96);
        commit;
    c. Configure CDC using Binary Reader
        exec rdsadmin.rdsadmin_master_util.create_archivelog_dir;
2. Prepare Aurora PostgreSQL 11.6 cluster
    a. Setup networking and security related infrastructure
        i. Create Subnets, Subnet Group
        ii. Create Security Groups
        iii. Create all required information for cross account data transfer
    b. Create Aurora PostgreSQL 11.6 cluster in AWS with sufficient capacity, we went with **db.r5.12xlarge (48 vCPU & 384 GB Memory)**
        i. Created only writer instance
        ii. Disabled auto vacuum
        iii. Disabled synchronous_commit
3. Create only tables no indexes, no foreign key constraint & no sequences on the PostgreSQL DB.
4. Setup DMS
    a. Create Replication instance (**dms.r4.4xlarge 16 vCPU & 128GB Memory with 200GB storage**)
    b. Create Source & Target End points
        i. Source end point with extra connection attributes
            parallelLoadThreads=1;ReplaceChars=EDA88D,3F
        ii. Target end point with extra connection attribute
            executeTimeout=3600;
    c. Setup DMS tasks
        i. Task for Schema 1
            1. Task for table set 1
            2. Task for table set 2
            3. Task for table set 3
            4. Task for table set 4
            5. Task for table set 5
            6. Task for table set 6
            7. Task for table set 7

   ii. Task for Schema 2
   iii. Task for Schema 3
   iv. Task for Schema 4
   v. Task for Schema 5
   vi. Task for Archived Schema
   vii. Task for on-demand activity

5. Test everything in QA
6. Run a dry-run from the production data, we selected 20% total tables to get an idea how long it can take for all tables when we do the actual migration.
7. When everything worked as expected in QA & in production dry-run; we started data migration in Production for each schema, not all task at the same time, all mentioned 11 tasks were started in such a way that at any given point of time we are not running more than 3-4 concurrent sessions to avoid overloading of our production system.
8. Once full load is done then create all indexes i.e., 1m+ indexes
   a. Distributed all the indexes in multiple scripts
   b. At any given point of time, we were running 5-7 scripts concurrently
9. Once index creation is done then started the CDC i.e., ongoing replication
10. Run the vacuum on 280K+ tables to collect stats
   a. Distributed vacuum command in multiple scripts
   b. At any given point of time, we were running 10-12 scripts concurrently
11. Added reader instance couple of hours prior to cut over time.
12. Day of cut-over from Oracle to Aurora PostgreSQL with some downtime
   a. Stop the application, lock the application DB user to make sure no more data being written to source database
   b. Stop all DMS tasks making sure all data has been replicated to target DB so put a monitoring in place to make sure.
   c. Enable the Foreign Key Constraints in Schema 2,3, 4 & 5
   d. Create all sequences
   e. Create all triggers
13. Do a sanity check of the db
14. Start the application
15. Challenges faced immediately after traffic is live

   As soon traffic started hitting the database within few minutes CPU is hovering 100% and everything was on-CPU as you can see here



Started looking into it to figure out why this is happening so started searching best practice documents to see if I might have missed something, so I found that parameters **seq_page_cost & random_page_cost** was set to their default values. Once I changed seq_page_cost from 1 (default) to 4 and random_page_cost from 4 (default) to 1.5, the difference in cpu utilization was immediately visible as you can see in the above graph.

16. Post migration issues

   a. Query performance degradation

      - One of the query which was not even visible in top 20 queries in Oracle that started showing as top 5 queries with high CPU utilization so started digging into it, it was found that there was a function based index on the table so I checked it in Oracle, I found that function based index was not being used in Oracle too but Oracle were able to handle the execution time for this query every efficiently whereas Aurora PostgreSQL could not handle so we need to modify query in application so that it can start using the function based index.

      - There were 1-2 more queries which were needs to be tunned though these queries were running fine in Oracle.

      - I have disabled enable_bitmapscan parameter because this was not working out very well for our queries.

      - Changed the value of work_mem from 4BM (default) to 32MB to avoid any temp table creation on disk.

      - Set the value of idle_in_transaction_session_timeout to 10 seconds

      - For one of our processes for which we use only stored procedure, this procedure performance was terrible and to solve it we must make lot of change the way this procedure was written BTW there were no problem for this process in Oracle.

b. Vacuum issues

After migration to PostgreSQL vacuuming has become very big issue for us. The problem was not that vacuum process has issue; we optimized all parameters related to vacuuming process; then where was the issue? The issue was having very high number of tables inside the database plus creating 300-500 tables every day hence auto vacuum process was not able to cope with the number of tables are getting eligible for the vacuum.

As the auto vacuum was not able to cope up and due to that MaximumUsedTransactionIDs started climbing, within few days we started seeing that MaximumUsedTransactionIDs has crossed the autovacuum_freeze_max_age default value of 200m, now PostgreSQL has forced the auto vacuum in wraparound mode due to that currently used tables started having bloating i.e. dead tuples started accumulating causing the query performance degradation.

Another surprise for us was the increase of MaximumUsedTransactionIDs for the tables which are in archive schema i.e. no writing on these tables or any table which does not have any write activity but MaximumUsedTransactionIDs is still increasing for all those tables, and we have 60% our tables in this category. I know this bit odd having these many tables, but it is what it is, and I need to solve this problem.

17.Stabilizing the system

We solved all the Query performance problem within few days, but vacuum process really took a long time to stabilize.

So, this is what I have done for the vacuuming

a. Parameters modified for the vacuum optimization.
   a. Increase maintenance_work_mem
   b. Adjust autovacuum_analyze_scale_factor & autovacuum_vacuum_scale_factor
   c. log_autovacuum_min_duration
   d. rds.force_autovacuum_logging_level
b. Disabled the auto vacuum and run vacuum manually every 5 minutes for all eligible tables which are being used actively for write (DML) operations using the same logic how PostgreSQL selects the table but add some additional conditions to it. For script see **appendix A**

c. For all the static tables i.e. tables in archive schema plus all tables which has no write activity, for these tables run a nightly script so that MaximumUsedTransactionIDs remains under control. For script see **appendix B**

I am attaching both the scripts for the reference.

## Appendix A:

**Script name: vacuum_script_nightly.sh**

```
#!/usr/bin/bash

SDIR="/var/lib/pgsql/work"
VDAY=`date +%a`
cd ${SDIR}
PNAME=`basename $0`
###Thresholds
VACUUM_THRESHOLD=500
ANALYZE_THRESHOLD=750
##Asign db related variables
HNAME=YOUR DB HOST
UNAME=YOUR DB USER
DBNAME=YOUR DB NAME

if [ -f ${SDIR}/status_${PNAME%%.*}.txt ]; then
 echo -e "Previous process still running\n"
 exit 1
else
 touch ${SDIR}/status_${PNAME%%.*}.txt
 echo -e "Process started at `date`" > ${SDIR}/status_${PNAME%%.*}.txt
fi

echo -e "`date`: Vacuum Start time"
psql -h${HANME} -p5432 -U${UNAME} -w -d${DBNAME} -e << EOF

\t
\o vacuum_script.sql

WITH vac AS (
WITH vbt AS (SELECT setting AS autovacuum_vacuum_threshold FROM pg_settings
WHERE name = 'autovacuum_vacuum_threshold')
    , vsf AS (SELECT setting AS autovacuum_vacuum_scale_factor FROM pg_settings
WHERE name = 'autovacuum_vacuum_scale_factor')
    , fma AS (SELECT setting AS autovacuum_freeze_max_age FROM pg_settings WHERE
name = 'autovacuum_freeze_max_age')
```

```
    , sto AS (select opt_oid, split_part(setting, '=', 1) as param, split_part(setting, '=', 2) as value
from (select oid opt_oid, unnest(reloptions) setting from pg_class) opt)
SELECT 'VACUUM VERBOSE ANALYZE '||ns.nspname||'.'||c.relname||';' as cmd
FROM pg_class c join pg_namespace ns on ns.oid = c.relnamespace
join pg_stat_all_tables stat on stat.relid = c.oid
join vbt on (1=1) join vsf on (1=1) join fma on (1=1)
left join sto cvbt on cvbt.param = 'autovacuum_vacuum_threshold' and c.oid = cvbt.opt_oid
left join sto cvsf on cvsf.param = 'autovacuum_vacuum_scale_factor' and c.oid = cvsf.opt_oid
left join sto cfma on cfma.param = 'autovacuum_freeze_max_age' and c.oid = cfma.opt_oid
WHERE c.relkind = 'r' and nspname <> 'pg_catalog' and ns.nspname not like 'data_archive'
and (stat.n_dead_tup>0 or stat.n_mod_since_analyze>0)
and (age(relfrozenxid) >= coalesce(cfma.value::float, autovacuum_freeze_max_age::float) or
coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
${VACUUM_THRESHOLD} +
coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) * c.reltuples <=
n_dead_tup)
ORDER BY age(relfrozenxid) DESC),
ana AS (
SELECT 'VACUUM VERBOSE ANALYZE '||ns.nspname||'.'||c.relname||';' as cmd
FROM pg_class c join pg_namespace ns on ns.oid = c.relnamespace join pg_stat_all_tables
stat on stat.relid=c.oid
WHERE c.relkind = 'r' and nspname <> 'pg_catalog' and ns.nspname not like 'data_archive'
and
(current_setting('autovacuum_analyze_threshold')::float + ${ANALYZE_THRESHOLD} +
current_setting('autovacuum_analyze_scale_factor')::float * c.reltuples) <=
n_mod_since_analyze
order by n_mod_since_analyze desc limit 500)
SELECT distinct * FROM vac UNION SELECT *FROM ana order by 1 desc;

EOF

echo -e "\nTables to be vacuumed: `wc -l vacuum_script.sql`\n"
psql -h${HANME} -p5432 -U${UNAME} -w -d${DBNAME} -e -f vacuum_script.sql -L
vacuum_script.txt

echo -e "`date`: Vacuum End time"

###Remove process lock file
rm ${SDIR}/status_${PNAME%%.*}.txt
```

## Appendix B:

**Script name: vacuum_script_nightly.sh**

```
#!/usr/bin/bash
###
### Script to vacuum all those tables which has greatest(age(c.relfrozenxid),age(t.relfrozenxid))>300m to avoid vacuum running in wraparound mode.
### Also vacuum pg_* tables from postgres and template1 databases
### Written by Krishan

SDIR="/var/lib/pgsql/work"
VDAY=`date +%a`
cd ${SDIR}
PNAME=`basename $0`
##Asign db related variables
HNAME=YOUR DB HOST
UNAME=YOUR DB USER
DBNAME=YOUR DB NAME

if [ -f ${SDIR}/status_${PNAME%%.*}.txt ]; then
 echo -e "Previous process still running\n"
 exit 1
else
 touch ${SDIR}/status_${PNAME%%.*}.txt
 echo -e "Process started at `date`" > ${SDIR}/status_${PNAME%%.*}.txt
fi

echo -e "`date`: ${PNAME} Start time"

psql -${HANME} -p5432 -U${UNAME} -w -d${DBNAME} -e << EOF
\t
\o ${PNAME%%.*}.sql
select '--'||to_char(current_date,'Dy');
WITH cms_data_archive AS (SELECT 'VACUUM FREEZE '||c.oid::regclass||';' as cmd,greatest(age(c.relfrozenxid),age(t.relfrozenxid)) as greatest_age,stat.n_dead_tup,c.reltuples
FROM pg_class c LEFT JOIN pg_class t ON c.reltoastrelid = t.oid JOIN pg_stat_all_tables stat on stat.relid=c.oid
WHERE stat.schemaname='cms_data_archive' and c.relkind IN ('r', 'm') and greatest(age(c.relfrozenxid),age(t.relfrozenxid))>260000000),
zero AS (
SELECT 'VACUUM FREEZE '||c.oid::regclass||';' as cmd,greatest(age(c.relfrozenxid),age(t.relfrozenxid)) as greatest_age,stat.n_dead_tup,c.reltuples
FROM pg_class c LEFT JOIN pg_class t ON c.reltoastrelid = t.oid JOIN pg_stat_all_tables stat on stat.relid=c.oid
WHERE stat.schemaname<>'cms_data_archive' and c.relkind IN ('r', 'm') and greatest(age(c.relfrozenxid),age(t.relfrozenxid))>260000000 and (stat.n_dead_tup+stat.n_mod_since_analyze)=0
),
nonzero AS (
SELECT 'VACUUM VERBOSE ANALYZE '||c.oid::regclass||';' as cmd,greatest(age(c.relfrozenxid),age(t.relfrozenxid)) as greatest_age,stat.n_dead_tup,c.reltuples
FROM pg_class c LEFT JOIN pg_class t ON c.reltoastrelid = t.oid JOIN pg_stat_all_tables stat on stat.relid=c.oid
WHERE stat.schemaname NOT IN ('cms_data_archive','cms_master') and c.relkind IN ('r', 'm') and greatest(age(c.relfrozenxid),age(t.relfrozenxid))>260000000 and (stat.n_dead_tup+stat.n_mod_since_analyze)>0
),
cms_master AS (
SELECT 'VACUUM VERBOSE ANALYZE '||c.oid::regclass||';' as cmd,greatest(age(c.relfrozenxid),age(t.relfrozenxid)) as greatest_age,stat.n_dead_tup,c.reltuples
FROM pg_class c LEFT JOIN pg_class t ON c.reltoastrelid = t.oid JOIN pg_stat_all_tables stat on stat.relid=c.oid
WHERE stat.schemaname='cms_master' and c.relkind IN ('r') and (stat.n_dead_tup+stat.n_mod_since_analyze)>0 and to_char(current_date,'Dy') IN ('Fri','Sat')
),
ordlist AS (
SELECT cmd,greatest_age,n_dead_tup,reltuples FROM cms_data_archive
union
SELECT cmd,greatest_age,n_dead_tup,reltuples FROM zero
union
SELECT cmd,greatest_age,n_dead_tup,reltuples FROM nonzero
union
SELECT cmd,greatest_age,n_dead_tup,reltuples FROM cms_master)
SELECT cmd FROM ordlist ORDER BY greatest_age DESC;

EOF

cp ${PNAME%%.*}.sql ${PNAME%%.*}.sql.`date +%a`

wc -l ${PNAME%%.*}.sql

rm x??
rm x??.txt
split -l 5000 ${PNAME%%.*}.sql

###Run the script in parallel 12 sessions, everyday there are around 50k-60k tables getting eligible for vacuuming based on age of transaction id

for fname in `ls x??|head -n 12`;do
 echo -e "Processing file: $fname"
 time psql -h${HANME} -p5432 -U${UNAME} -w -d${DBNAME} -b -f ${fname} -L ${fname}.txt &
done

##Vacuum tables from template1 & postgres schema's
psql -h${HANME} -p5432 -U${UNAME} -w -d${DBNAME} -e << EOF
\t
\o pg_tables_vacuum_c.sql
select 'vacuum verbose analyze '||c.relname||'; --'||age(relfrozenxid) as cmd
from pg_class c JOIN pg_stat_all_tables stat on stat.relid=c.oid where relkind in ('r','m') and c.relname like 'pg_%' and age(relfrozenxid)>60000000
order by age(relfrozenxid) desc;

\c postgres
\o pg_tables_vacuum_p.sql
select 'vacuum verbose analyze '||c.relname||'; --'||age(relfrozenxid) as cmd
from pg_class c JOIN pg_stat_all_tables stat on stat.relid=c.oid where relkind in ('r','m') and c.relname like 'pg_%' and age(relfrozenxid)>60000000
order by age(relfrozenxid) desc;

\c template1
\o pg_tables_vacuum_t.sql
```

```
select 'vacuum verbose analyze '||c.relname||'; --'||age(relfrozenxid) as cmd
from pg_class c JOIN pg_stat_all_tables stat on stat.relid=c.oid where relkind in ('r','m') and
c.relname like 'pg_%' and age(relfrozenxid)>60000000
order by age(relfrozenxid) desc;

EOF

psql -h${HANME} -p5432 -U${UNAME} -w -d${DBNAME} -e << EOF

SELECT datname, age(datfrozenxid) FROM pg_database;

\i pg_tables_vacuum_c.sql

\c postgres
\i pg_tables_vacuum_p.sql

\c template1
\i pg_tables_vacuum_t.sql

SELECT datname, age(datfrozenxid) FROM pg_database;
EOF

echo -e "`date`: ${PNAME} End time"

###Remove process lock file
rm ${SDIR}/status_${PNAME%%.*}.txt
```

## Appendix C:

Reference for the resource used

Using an Oracle database as a source for AWS DMS - AWS Database Migration Service (amazon.com)
https://aws.amazon.com/blogs/database/best-practices-for-migrating-an-oracle-database-to-amazon-rds-postgresql-or-amazon-aurora-postgresql-migration-process-and-infrastructure-considerations/
https://www.youtube.com/watch?v=n5K_FfrFjLo&t=13s
https://d1.awsstatic.com/whitepapers/Migration/oracle-database-amazon-aurora-postgresql-migration-playbook.pdf
Presentation from AWS re-Invent 2019
https://www.youtube.com/watch?v=UsNmIo4ymUo&t=2455s
https://aws.amazon.com/blogs/database/how-to-solve-some-common-challenges-faced-while-migrating-from-oracle-to-postgresql/
https://aws.amazon.com/blogs/database/challenges-when-migrating-from-oracle-to-postgresql-and-how-to-overcome-them/
https://aws.amazon.com/blogs/database/introduction-to-aurora-postgresql-cluster-cache-management/
https://aws.amazon.com/blogs/database/best-practices-for-migrating-an-oracle-database-to-amazon-rds-postgresql-or-amazon-aurora-postgresql-target-database-considerations-for-the-postgresql-environment/
https://severalnines.com/blog/tuning-io-operations-postgresql