Database Migration

Oracle 12.2 on Exadata X7
To
PostgreSQL 12.4

By Ujang Jaenudin

Who am I

- Self Employed DBA Since 2011
- Certified Oracle DBA since 2006
- PostgreSQL DBA Since 2014
- 1st Project using Postgres 9.1
- Now Maintain PostgreSQL DB:
  Core financial system, Fintech,
  ATM Switching, Retailer,
  Bank's Middleware System, etc

Objectives

- System Overview
- Migration Objectives
- Database Migration
- Core Function
- Core Function Tuning
- Summary

System Overview

- Critical OLTP System
- Peak Transaction around 400 TPS
- System accessed from million of Mobile devices
- Critical part: Searching available seats on the route

System Overview Cont'd

Exadata X7 1/8:

RAC 2 Nodes

48 CPU cores Total

1.5 TB Ram Total

Xeon(R) Platinum 8160 CPU @ 2.10GHz

L1d cache:	32K
L1i cache:	32K
L2 cache:	1024K
L3 cache:	33792K

VS

Postgres (DEV):

1 Node/Server

4 CPU cores

64 GB Ram

Xeon(R) Gold 6154 CPU @ 3.00GHz

L1d cache:	32K
L1i cache:	32K
L2 cache:	1024K
L3 cache:	25344K

Migration Objectives

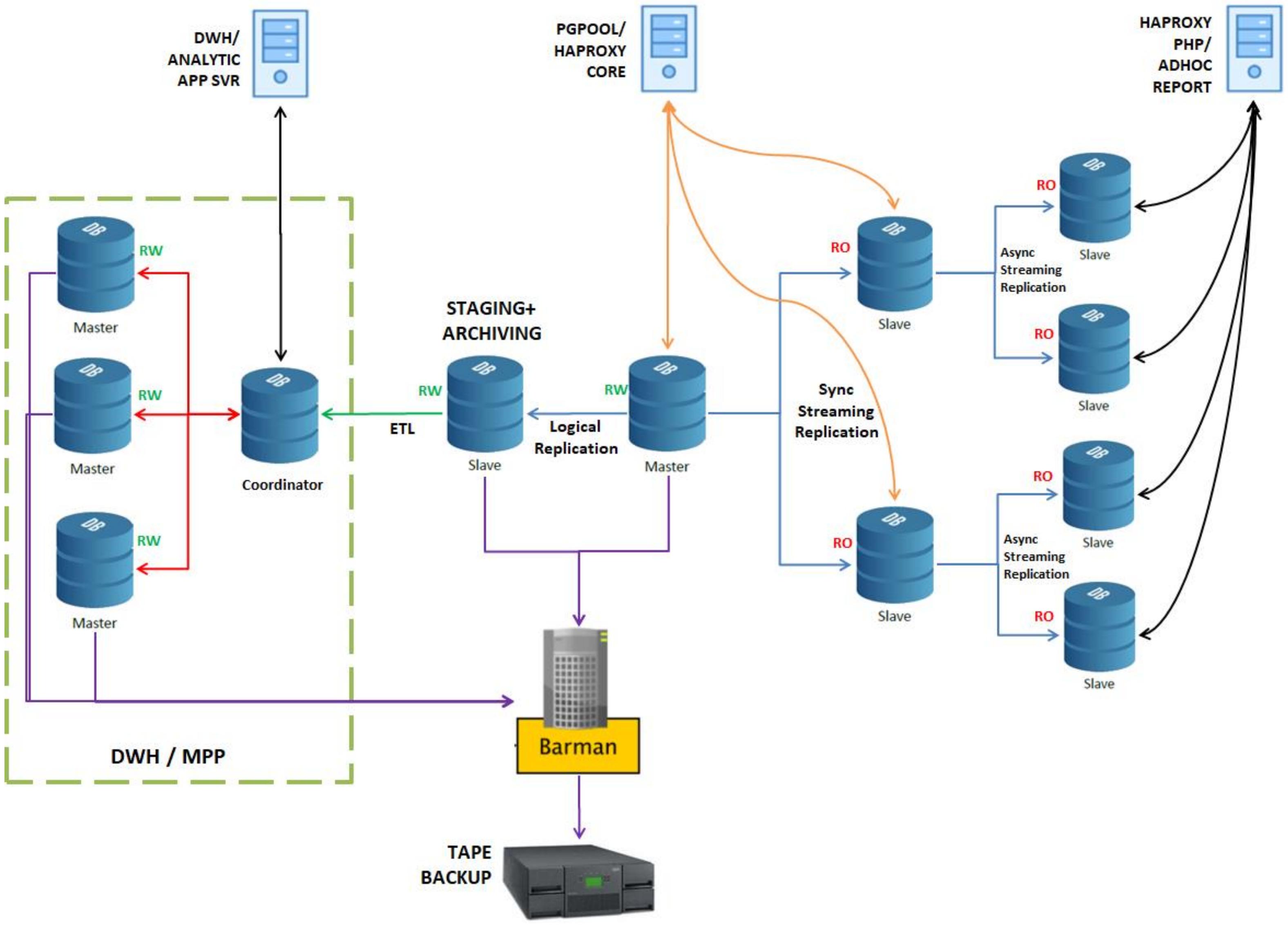
- Cost
- Flexibility, extend/reduce servers on certain Events
- Future business growth
- Scalability

Migration Objectives Cont'd

- Current workload most likely 80% READ, 20% WRITE
- PostgresQL able to scale well for READS
- We plan to have some slaves (Synchronous SR)
- HA-proxy in front of them

Migration Objectives Cont'd

Dreamed DB Farm Layout



Database Migration

Preparation Phase:

- Define data type mapping
- Install and latest postgreSQL (12.4)
- Install and Configure orafce
- Install and Configure plprofiler
- Install oracle instantclient
- Install ora2pg

Database Migration Cont’d

Migration Phase:

- Create project (work tree)
- Export schema
- Modify some table definitions (data type, partitions)
- Prepare postgres: User, DB, schema
- Load schema into postgres
- Generate kettle transformation file
- Edit kettle file (sed is our best friend)

Database Migration Cont'd

Migration Phase:

- Grouping kettle config file (small and big tables)
- Split big tables's query (hot and cold data)
- Run kettle small tables
- Run Kettle big table's hot data
- Import Rest objects

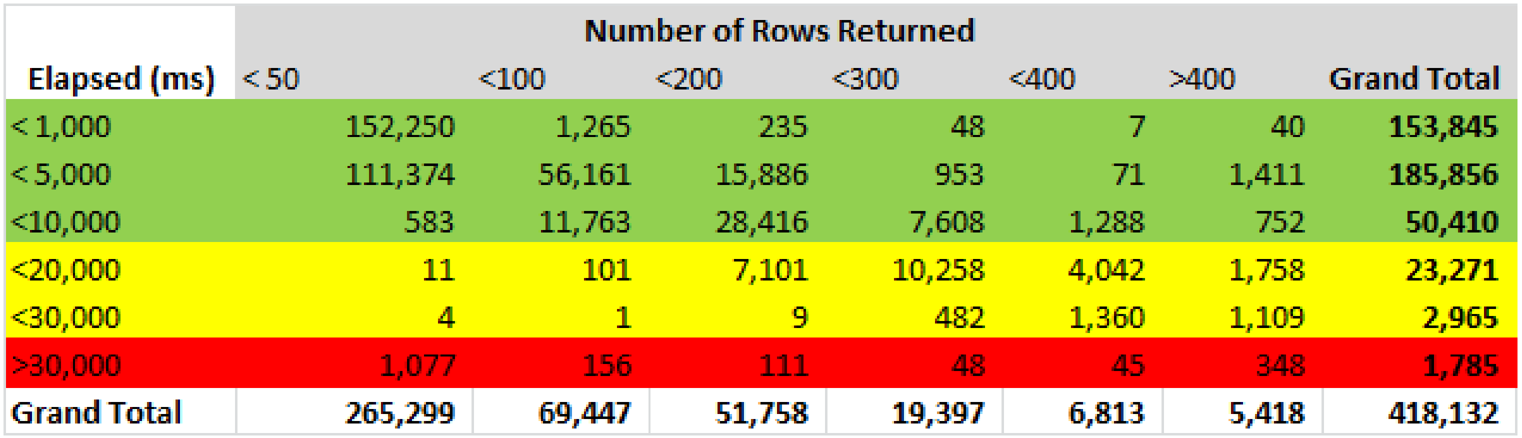
  (type,function,procedure,views,trigger,sequence)
- Create Indexes
- Run Kettle big table's cold data

Core Function

- We must Convert core function from oracle PL/SQL table function to Postgres pl/pgsql function
- Function must return a bunch of records
- On peak hour must survive from atleast 500 concurrent active sessions

Core Function Cont’d

Exadata X5 1/8
Stress Test Result 500 Concurrent Sessions

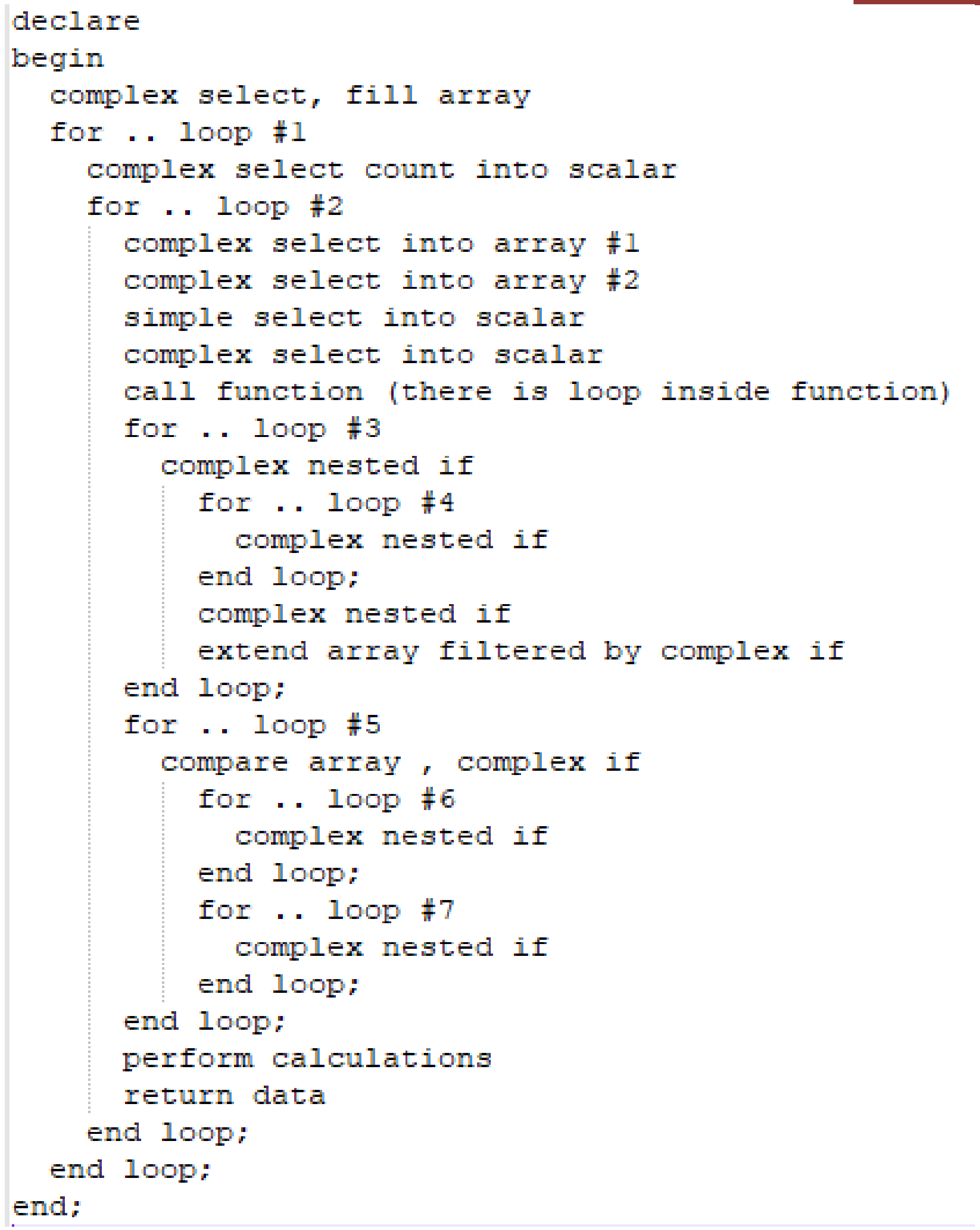


	Number of Rows Returned						
Elapsed (ms)	< 50	<100	<200	<300	<400	>400	Grand Total
< 1,000	152,250	1,265	235	48	7	40	**153,845**
< 5,000	111,374	56,161	15,886	953	71	1,411	**185,856**
<10,000	583	11,763	28,416	7,608	1,288	752	**50,410**
<20,000	11	101	7,101	10,258	4,042	1,758	**23,271**
<30,000	4	1	9	482	1,360	1,109	**2,965**
>30,000	1,077	156	111	48	45	348	**1,785**
Grand Total	**265,299**	**69,447**	**51,758**	**19,397**	**6,813**	**5,418**	**418,132**

- Total 500,000 hit to Function by 500 active sessions
- Total Time taken 3138 seconds
- There are 81,868 queries without result
- 99.57% Success rate (all queries completed below timeout)
- 0.43% Failure Rate due to timeout (App set 30s as timeout)

Core Function Cont’d

Core Function Pseudo



```
declare
begin
  complex select, fill array
  for .. loop #1
    complex select count into scalar
    for .. loop #2
      complex select into array #1
      complex select into array #2
      simple select into scalar
      complex select into scalar
      call function (there is loop inside function)
      for .. loop #3
        complex nested if
          for .. loop #4
            complex nested if
          end loop;
          complex nested if
          extend array filtered by complex if
      end loop;
      for .. loop #5
        compare array , complex if
          for .. loop #6
            complex nested if
          end loop;
          for .. loop #7
            complex nested if
          end loop;
      end loop;
      perform calculations
      return data
    end loop;
  end loop;
end;
```

Core Function Tuning

- Without Tuning, 1 hit = 64 sec
- Disable JIT
- PL/PGSQL For loop = slow
- Array, unnest array = slow
- We are big fan of cursor, open-fetch-close
- After tuning, able to reach 700ms per hit
- Exadata: 70ms per hit
- We still unable to beat exadata speed ☹

Summary

- Ora2pg really cool migration tool
- Ora2pg perl data migration very slow
- Use Pentaho kettle to speedup data migration
- Play attention with data type mapping
- Postgres PL/PGSQL slower than oracle PL/SQL
- Cost effective and flexibility still wins the game ☺

Thank You

Terima Kasih