OLTP Database Optimizations for high-volume and high-velocity use cases

Nov 09, 2017

Samrat Roy, Dheeraj Kondapaneni, Rav Pedpati

PayPal

AGENDA

- Fast-o-meter and weight-o-meter
- Index me
- Extra miles with Trigger
- Descale writes in RAC
- Summary and Q & A
- Fast Data Copy (if time is kind ☺)

Fast-o-meter and weight-o-meter

Read IOPS	Write IOPS	Execs/sec
>75K	>75K	>500K

Read PayLoad/sec	Write PayLoad/sec
>0.5GB	>0.5GB

readTime p95	readTime p99	writeTime p95	writeTime p99
3ms	6ms	10ms	15ms

Index Usecase #1

Problem Statement: Right-hand Index contention on heavy duty tables

Artifacts:

- Surrogate key id based
- Unique index as based on database sequence
- Issue exposed only in high concurrent fast usecases

HEADER_FILE	HEADER_BLOCK	EXTENTS
171	28882	1

Solve#1: Re-create index as reverse

Issues:

- Range scan expensive
- Still a bottleneck as close keys still share same block

Block# 28883

```
row#0[8021]
col 0; len 2; (2):  02 c2
row#1[8010]
col 0; len 2; (2):  03 c2
row#2[7999]
col 0; len 2; (2):  04 c2
row#3[7988]
col 0; len 2; (2):  05 c2
```

tabdemo1

A	B
100	100
200	101
300	102
400	103

Select A,B from tabdemo1
where B between 1 and 10000;

Index Usecase #1

Solve#2: Re-create index as global hash partitioned

Issues:

- Partition management issues
- Issue with adding more partitions

HEADER_FILE	HEADER_BLOCK	EXTENTS
171	28882	1
171	28906	1
171	28898	1
171	28890	1

tabdemo1

A	B
100	100
200	101
300	102
400	103

Block# 28883

row#0[8021]
col 0; len 2; (2): c2 02

Block# 28907

row#1[8016]
col 0; len 2; (2): c2 04

Block# 28899

row#2[8025]
col 0; len 2; (2): c2 02

Block# 28891

row#3[8022]
col 0; len 2; (2): c2 04

Index usecase #1

Solve#3 [FINAL]: Re-structure table as index and keep it local

Issues:

- One time table restructure effort

```
create table demo1 (a number, mod_a number, b
number)
partition by range (a)
subpartition by range(mod_a)
(partition p1 values less than (1000)
(subpartition psub1 values less than (1),
subpartition psub2 values less than (2)),
partition p2 values less than (2000)
(subpartition psub3 values less than (1),
subpartition psub4 values less than (2)));

create unique index demo1_pk on demo1 (mod_a, a)
local;
```

HEADER_FILE	PARTITION_NAME	HEADER_BLOCK
14	PSUB1	2816
14	PSUB2	2944
14	PSUB3	3072
14	PSUB4	3200

```
insert into demo1 values
(543,mod(543,2),1);
insert into demo1 values
(544,mod(544,2),1);
```

A	MOD_A	B	SUBP
543	1	1	PSUB2
544	0	1	PSUB1

Index Usecase #2

Problem Statement: Slow query for finding pending transactions (ordered with latest first) for an account for given time window

Artifacts:

- Query uses existing index on account number and time
- Each record belongs to unique transaction for a given account
- Only less than 5% of transactions are in pending status
- Transaction “status” is frequently updated
- Non-Issue in slow/non-mutating key and small tables

Solve#1: Create index on account, time, status

Only ½ needed to be indexed

Account_No	Time_Created	Status
100	1000	2
200	1000	10
105	1005	2
300	1010	0

But all status keys gets indexed

Issues:

- Still need to scan lot of blocks because of skewed data
- Large index size with non-selective data
- Very high key mutations

Index Usecase #2

Solve#2: Create functional index on account, time, case when (status=2) then status else null end

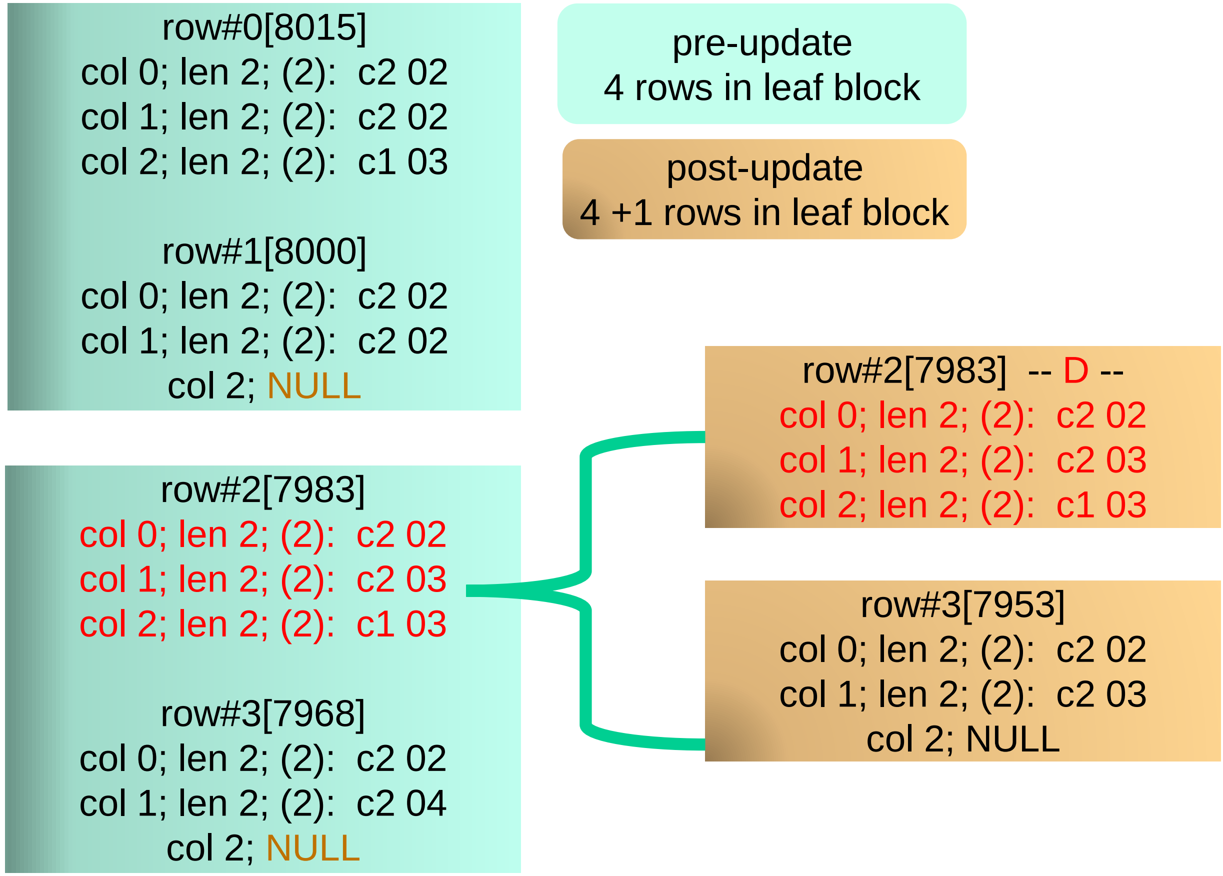
Issues:

- ➢ Still large index size as it indexes status<>2 as well with NULL entries
- ➢ Still need to scan lot of blocks because of leaf blocks with NOT NULL account no and time keys and NULL status

row#2 update:

update testacc
set status=8
where account_no=105 and time_created=1005
and status=2;

Account_No	Time_Created	Status
100	1000	2
200	1000	10
105	1005	2
300	1010	0



```
row#0[8015]
col 0; len 2; (2):  c2 02
col 1; len 2; (2):  c2 02
col 2; len 2; (2):  c1 03

row#1[8000]
col 0; len 2; (2):  c2 02
col 1; len 2; (2):  c2 02
col 2; NULL

row#2[7983]
col 0; len 2; (2):  c2 02
col 1; len 2; (2):  c2 03
col 2; len 2; (2):  c1 03

row#3[7968]
col 0; len 2; (2):  c2 02
col 1; len 2; (2):  c2 04
col 2; NULL
```

pre-update
4 rows in leaf block

post-update
4 +1 rows in leaf block

```
row#2[7983]  -- D --
col 0; len 2; (2):  c2 02
col 1; len 2; (2):  c2 03
col 2; len 2; (2):  c1 03

row#3[7953]
col 0; len 2; (2):  c2 02
col 1; len 2; (2):  c2 03
col 2; NULL
```

Index Usecase #2

Solve#3 [FINAL]: Create functional index on
case when (status=2) then account else null end,
case when (status=2) then status else null end,
case when (status=2) then time else null end

Account_No	Time_Created	Status
100	1000	2
200	1000	10
105	1005	8
300	1010	0

row#0 update:
update testacc set status=80 where account_no=100 and time_created=1000 and status=2;

```
row#0[8015]
col 0; len 2; (2):  c2 02
col 1; len 2; (2):  c2 02
col 2; len 2; (2):  c1 03
```

pre-update
1 row in leaf block

post-update
0+1 row in leaf block
to-be-deleted

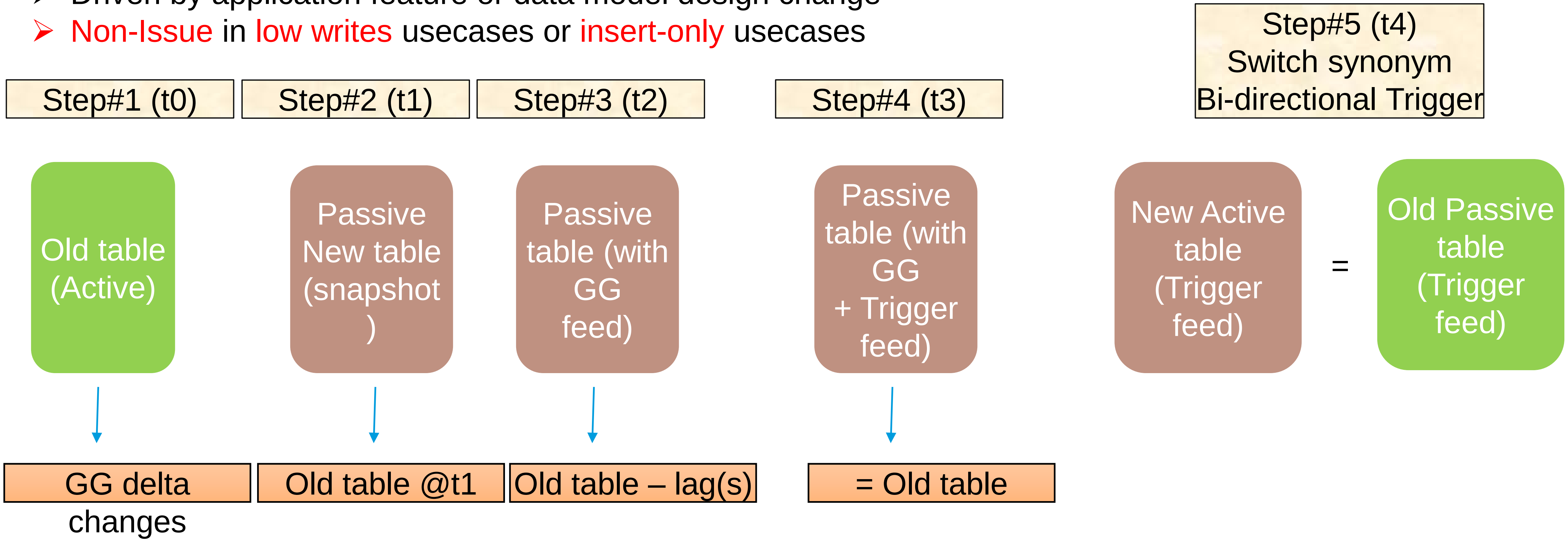
```
row#0[8015] flag: ---D--
col 0; len 2; (2):  c2 02
col 1; len 2; (2):  c2 02
col 2; len 2; (2):  c1 03
```

Trigger Usecase #1

Problem Statement: Zero downtime table-restructure online

Artifacts:

- Critical heavy duty table needs major restructure
- Driven by application feature or data model design change
- Non-Issue in low writes usecases or insert-only usecases



Step#1 (t0)

Old table (Active)

GG delta changes

Step#2 (t1)

Passive New table (snapshot)

Old table @t1

Step#3 (t2)

Passive table (with GG feed)

Old table – lag(s)

Step#4 (t3)

Passive table (with GG + Trigger feed)

= Old table

Step#5 (t4)
Switch synonym
Bi-directional Trigger

New Active table (Trigger feed) = Old Passive table (Trigger feed)

Trigger Usecase #1

Bi-directional Trigger

```
create trigger tabtrigdemo1 after update or insert on tabdemo1 for each row disable
begin
      if ((sys_context('USERENV','SESSION_USER') = ‘DEMOAPP') and
          sys_context('USERENV','SERVER_HOST') <> sys_context('USERENV','HOST') and
          nvl(sys_context('USERENV','CLIENT_INFO'),'a')='a')
      then
            dbms_application_info.SET_CLIENT_INFO('USED');
            merge into tabdemo2
            using dual on (COL1 = :new.COL1)
            when matched then update set COL2 = :new.COL2,..;
            when not matched then insert (COL1,COL2,..) values (:NEW.COL1,:NEW.COL2,..);
            dbms_application_info.SET_CLIENT_INFO(NULL);
      end if;
end;
/
```

Prevents loop by excluding “old” write from “new” writes

Label write as “not-to-be-replicated-back/old” before replication

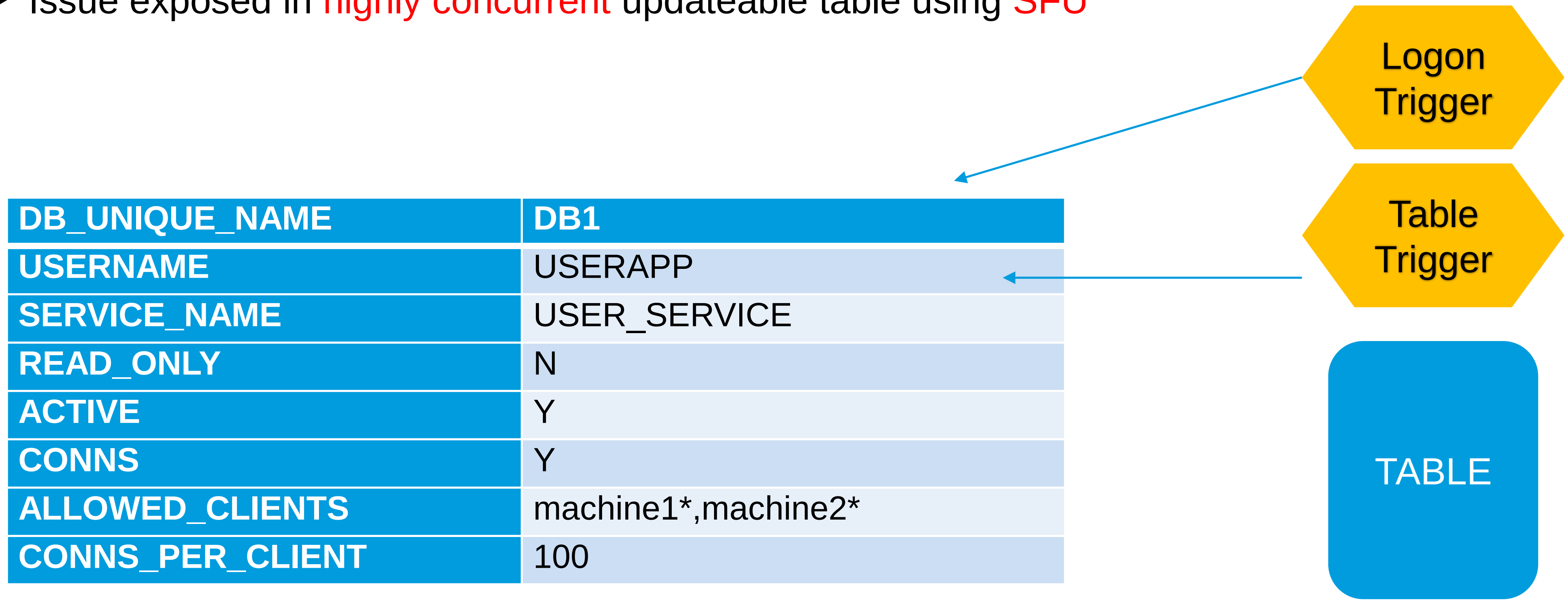
Resets Label for “new” incoming writes in same session

Trigger Usecase #2

Problem Statement: Throttle traffic for database or table(s)

Artifacts:

- SFU (Select For Update) is database issue first before it is application issue
- Login storm commonly known issue due to bad app box or conns config
- Issue exposed in highly concurrent updateable table using SFU



DB_UNIQUE_NAME	DB1
USERNAME	USERAPP
SERVICE_NAME	USER_SERVICE
READ_ONLY	N
ACTIVE	Y
CONNS	Y
ALLOWED_CLIENTS	machine1*,machine2*
CONNS_PER_CLIENT	100

Trigger Usecase #2

Logon Trigger

```
CREATE trigger sys.tabtrigdemo1 before INSERT OR UPDATE OR DELETE ON demo1 FOR
EACH ROW DISABLE

Declare
connFlag char(1);

Begin
select conns into connFlag from conns_map
where username=sys_context(‘userenv’,’session_user’) and
Service_name=sys_Context(‘userenv’,’service_name’) and
Db_unique_name=sys_context(‘userenv’,’db_unique_name’);

if (connFlag='N’) then
raise_application_error(….);
end if;
```

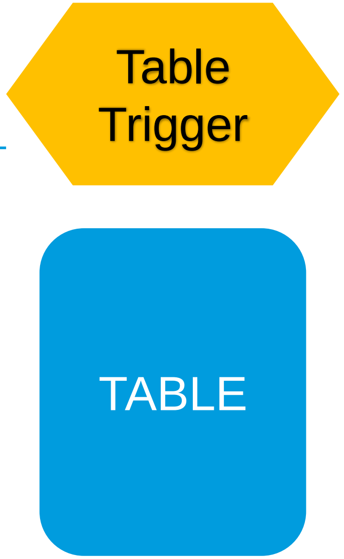
Trigger Usecase #3

Problem Statement: Reduce database downtime for maintenance

Artifacts:

- Maintenance around specific table or few tables have adverse collateral damage
- Table not in critical application flow
- Application does retry <n> times upon failure
- Issue in usecases with highly busy/concurrent table

Table Name	
USERNAME	DEMOAPP
SERVICE_NAME	DEMO_SERVICE
READ_ONLY	Y



Trigger Usecase #4

Problem Statement: Ordered sequence dependency of application in RAC database

Artifacts:

- Sequence if NOORDER across active/active RAC nodes will break application
- Sequence if ORDER across active/active RAC nodes will have contention/slowness

Solve:

- Database Logon Trigger to monitor services (being used for sequence) active/active status
- Table Trigger to monitor and reject writes if concerned services being active/active
- Sequence in steady state is always NOORDER
- During service failover (instance/node crash), there is a window where writes will start using new instance sequence cache which is smaller value than current
- Logon Trigger detects service failover, blocks login till it converts sequence to ORDER
- Once Sequence converted to ORDER, it allows new logins
- Once incident is over, after nth login its converted back to NOORDER

Writes Usecase #1

Problem Statement: Right-hand Index contention with writes in active/active RAC nodes

Artifacts:

- Writes in multiple nodes to scale
- High concurrent writes with indexes on time
- Issue exposed in highly busy systems with high write concurrency

Solve#1:

- “same” Reads are horizontal-scale friendlier than “same” writes
- Logical Partitioning of services with respective applications
- Logical Partitioning of “same” writes in few nodes (<=2)
- Have 2 active “same” write nodes for write availability

Issues:

- Scale issue if ”same” reads/writes are restricted to 2 nodes

Writes Usecase #1

Solve#2 [FINAL]:

- Decouple reads from writes by using simple sqltext parsing in application/connection pool
- Use separate read and write app pool to take advantage of read/write split
- Use OCIAttrGet to check connection is in transaction or not
- Have separate database service for reads and write
- Scale reads in N nodes by running read database service in N nodes
- Writes remain in 2 nodes

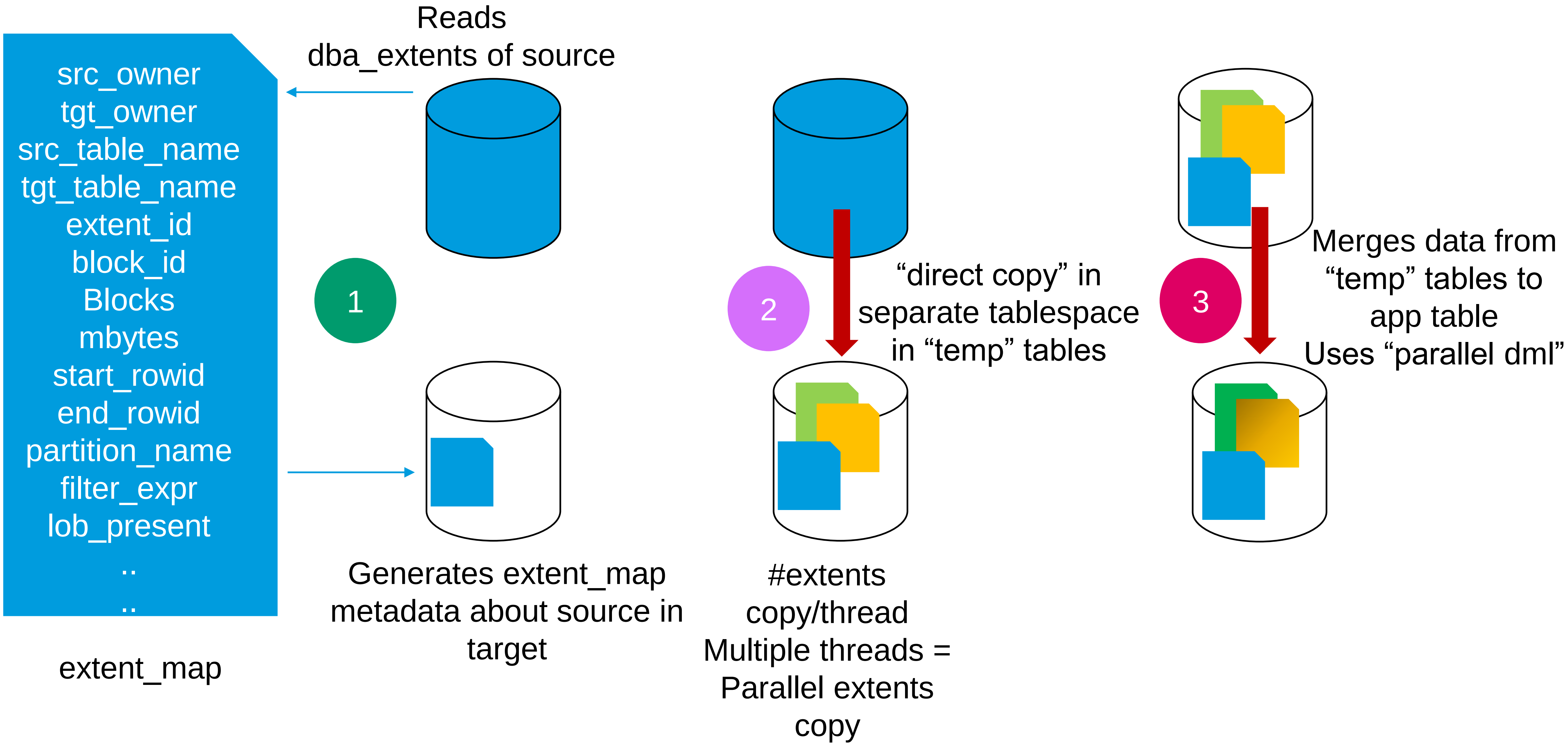
```
OCIAttrGet(
authp,
OCI_HTYPE_SESSION,
&txnInProgress,
(ub4 *)0,
OCI_ATTR_TRANSACTION_IN_PROGRESS,
errhp);

https://docs.oracle.com/cd/E11882_01/appdev.112/e10646/ociaahan.htm#LNOCI17835
```

Summary

- Knowing speed and payload for each table and application helps a long way to scale systems
- Common DBA activities in high-speed environment needs more planning and testing
- Application partnership with data architects along with DBAs critical
- Test the solve, iterate and evolve your solve as 1st solve more often is NOT the last solve!

Q & A

Fast Data Copy



src_owner
tgt_owner
src_table_name
tgt_table_name
extent_id
block_id
Blocks
mbytes
start_rowid
end_rowid
partition_name
filter_expr
lob_present
..
..

extent_map

Reads
dba_extents of source

Generates extent_map
metadata about source in
target

“direct copy” in
separate tablespace
in “temp” tables

#extents
copy/thread
Multiple threads =
Parallel extents
copy

Merges data from
“temp” tables to
app table
Uses “parallel dml”