DOGBERT'S PASSWORD RECOVERY SERVICE FOR MORONS
I DON'T REMEMBER MY PASSWORD.
IS IT "123"?
THAT'S JUST SPOOKY.

scottadams@aol.com

www.dilbert.com

1-17-07

© 2007 Scott Adams, Inc./Dist. by UFS, Inc.

Vzporedno izvajanje operacij s PL/SQL

Boris Oblak

ORACLE CERTIFIED PROFESSIONAL

boris.oblak@abakus.si

SIOUG 2004

Portorož, 19.-22.9.2004

Oracle parallelism

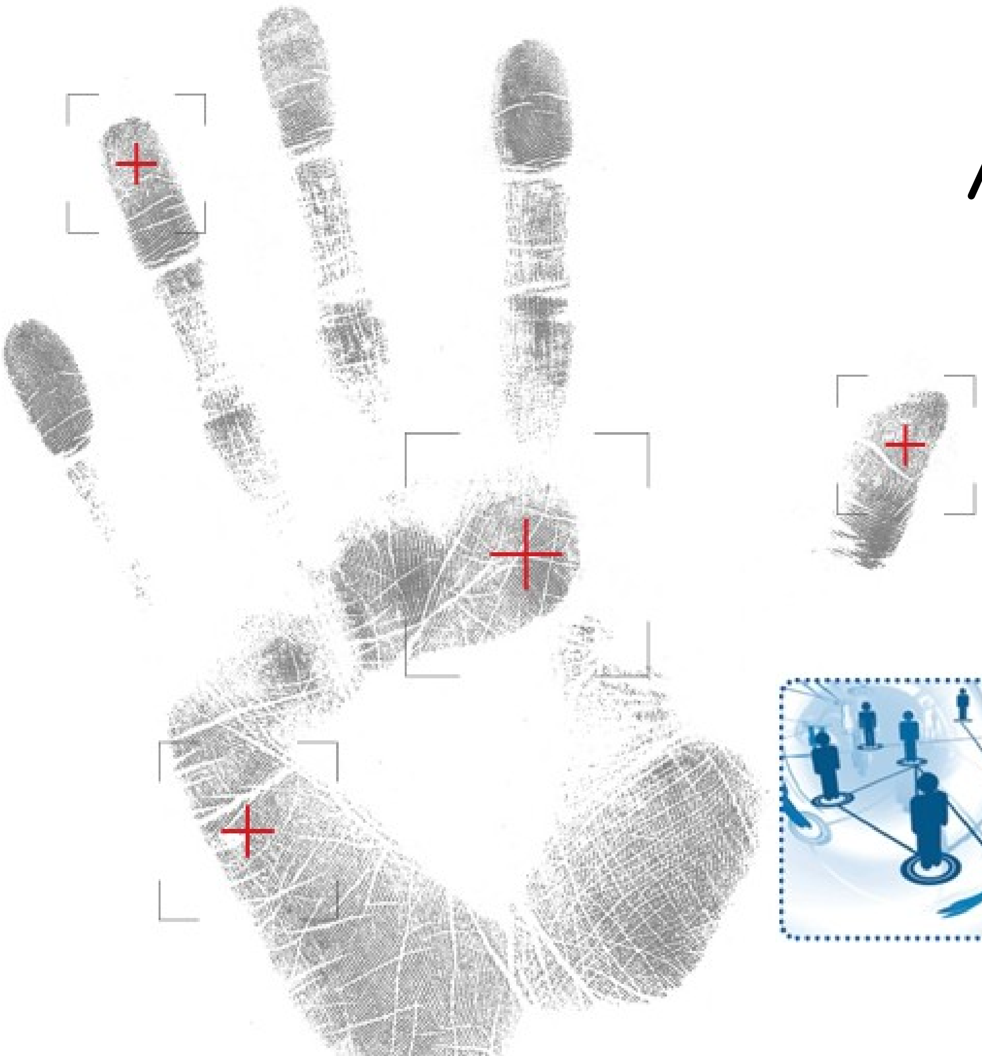
- Do It Yourself: 9*i*, 10g
- Oracle: in 11.2
  - dbms_parallel_execute

Abakus ARBITER

ORACLE Gold Partner

Boris Oblak
Abakus plus d.o.o.

ORACLE | CERTIFIED PROFESSIONAL



18. Strokovno srečanje
SIOUG 2013
14.-16. oktober 2013

Parallel executing

Abakus plus d.o.o.

ORACLE Gold Partner

History

from 1992, ~20 employees

Applications:

special (DB – Newspaper Distribution, FIS – Flight Information System)
ARBITER – the ultimate tool in audit trailing
APPM - Abakus Plus Performance and Monitoring Tool

Services:

DBA, OS administration , programming (MediaWiki, Oracle)
networks (services, VPN, QoS, security)
open source, monitoring (Nagios, OCS, Wiki)

Hardware:

servers, **SAN storage**, firewalls

Infrastructure:

from 1995 GNU/Linux *(18 years of experience !)*
Oracle on GNU/Linux: since RDBMS 7.1.5 & Forms 3.0 *(before Oracle !)*
>20 years of experience with High-Availability !

Sava

Gorenjska Banka
Banka s posluhom

Aerodrom Ljubljana

Mercator

GOODYEAR

BANKA SLOVENIJE
EVROSISTEM

Mestna občina Ljubljana

MESTNA OBČINA KOPER
COMUNE CITTA DI CAPODISTRIA

futuraplus

Iskra
Iskra MIS

DELO PRODAJA

KONTROLA ZRAČNEGA PROMETA SLOVENIJE

GENERALI GROUP

GENERALI
Zavarovalnica

Aerodrom Ljubljana

BANKA
SLOVENIJE
BANK OF SLOVENIA
EUROSYSTEM

DELO PRODAJA

KONTROLA
ZRAČNEGA
PROMETA
SLOVENIJE

Gorenjska Banka
Vse, kar šteje.

GOODYEAR

HRANILNICA LON
Bančništvo na Ljubezniv Oseben Način

Mestna občina
Ljubljana

Mercator

NLB Vita
Življenjska zavarovalnica

MESTNA OBČINA KOPER
COMUNE CITTÀ DI CAPODISTRIA

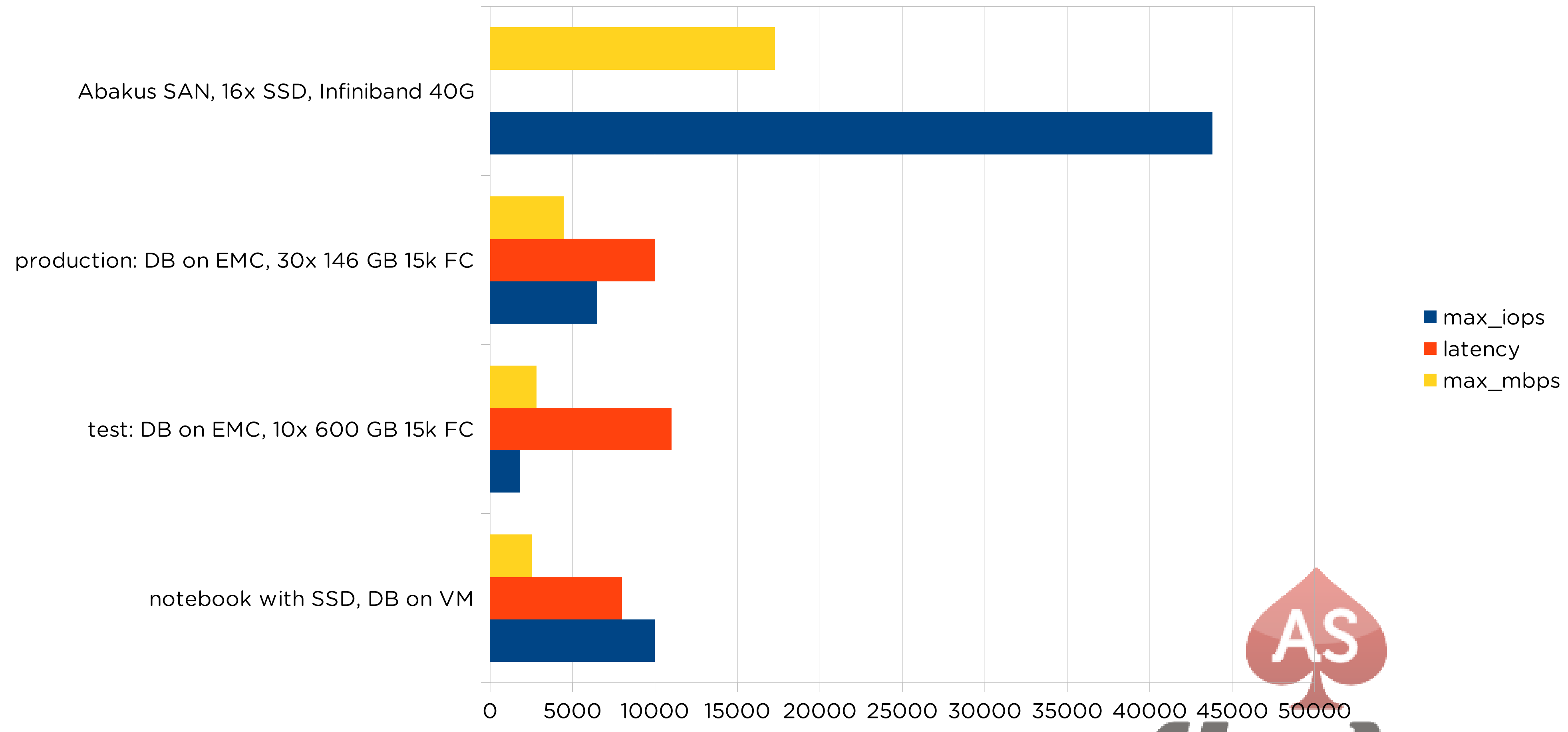
Sava

AS
Abakus
As na disku.

Performance (1)

- test 1 (notebook with SSD, DB on VM):
  max_iops = **9.983**, latency = **8**, max_mbps = **251**
- test 2 (test DB on EMC, 10x 600 GB 15k FC):
  max_iops = **1.824**, latency = **11**, max_mbps = **280**
- test 3 (production DB on EMC, 30x 146 GB 15k FC):
  max_iops = **6.498**, latency = **10**, max_mbps = **455**
- test 4 (Abakus SAN, 16x SSD, Infiniband 40G):
  max_iops = 43.782, latency = 0, max_mbps = 1.727

Performance (2)



Challenge

- batch task: updating balance for each customer
- batch task lasts for 4 hours on a single instance
  - 2-quad core CPU
- after transition to RAC execution time varries between 4 and 7 hours
  - 2-sixteen core CPU

Problem analysis

- all batch tasks use the same log table for reporting
  - insert
  - update
- index and data block contention
- > 200.000 customers, each from 10 to 700.000 records

Phase 1: block contention (1)

- create log table for each instance
  - two tables in our case, apl_log_1 and apl_log_2
- create view
  - apl_log as select * from apl_log_1 union all select * from apl_log_2
- modified logging procedure
  - depends on instance insert/update into instance table
- increase sequence cache size

Phase 1: block contention (2)

```
CASE sys_context ('userenv', 'instance')
   WHEN 1 THEN
      UPDATE apl_log_1 ...;
      INSERT INTO apl_log_1 ...;
   WHEN 2 THEN
      UPDATE apl_log_2 ...;
      INSERT INTO apl_log_2 ...;
   ELSE
      ...
      NULL;
END CASE;
```

Phase 1: fixed execution time

- We fixed the execution time to less than 4 hours.
- But … can we do it better?

Phase 1: fixed execution time

- We fixed the execution time to less than 4 hours.
- But ... can we do it better?
- Yes we can :)

Phase 1: fixed execution time

- We fixed the execution time to less than 4 hours.
- But … can we do it better?
- Yes we can :)
- 2 CPU x 16 core
- Parallel processing

Oracle parallelism

- Database parallelism
  - parallel operations requires EE licence (http://docs.oracle.com/cd/E16655_01/license.121/e17614/editions.htm)

    ```
    ALTER SESSION ENABLE PARALLEL DML;
    UPDATE /*+ parallel(t,64) */ t SET col = expr;
    ```

  - PL/SQL parallelism?
- With Oracle Standard Edition:
  - Do It Yourself: 9*i*, 10g
    - 9*i*: dbms_job
    - >=10g: dbms_scheduler
  - Oracle >= 11.2
    - dbms_parallel_execute

Parallelism in Oracle SE

- dbms_parallel_execute
- :-) We can use all CPU cores (CPU licence is per socket).

dbms_parallel_execute (1)

- »cut« data of an updated table into fragments (chunks)
- apply update statement to every fragment
- chunks have borders
  - start and end value
- Oracle can calculate borders

```
UPDATE t SET col=<value>
 WHERE t.rowid BETWEEN :start_id AND :end_id
```

dbms_parallel_execute (2)

- Oracle can »cut« table's data using:
  - rowid
  - value of any number column in table
  - custom condition
- statistics?
- CREATE JOB privilege

dbms_parallel_execute (3)

```
CREATE TABLE test_table AS
SELECT LEVEL AS id
     , 'Value ' || TO_CHAR(LEVEL) AS name
     , ROUND(DBMS_RANDOM.VALUE(1, 10)) AS RANK
  FROM dual
CONNECT BY LEVEL <= 1000000;
COMMIT;
EXECUTE dbms_stats.gather_table_stats (user,
'TEST_TABLE');
```

- our goal is to update full table

```
UPDATE test_table
   SET name =
   TO_CHAR(RANK) || ' ' || name;
```

dbms_parallel_execute (4)

- required steps
  - create task
  - generate chunks in way you wish
  - run task
  - drop task
- USER_PARALLEL_EXECUTE_TASKS
- USER_PARALLEL_EXECUTE_CHUNKS

dbms_parallel_execute (5)

- modified UPDATE statement

```
CREATE PROCEDURE update_test_table(
   p_start_id IN ROWID, p_end_id IN ROWID) AS
BEGIN
   UPDATE test_table
      SET NAME = to_char(rank) || ' ' || NAME
    WHERE ROWID BETWEEN p_start_id AND p_end_id;
END;
```

dbms_parallel_execute (6)

```
DECLARE
  c_task_name  CONSTANT VARCHAR2(128) := 'TEST TASK. BY ROWID';
BEGIN
  dbms_parallel_execute.create_task(c_task_name);

  dbms_parallel_execute.create_chunks_by_rowid (
       task_name   => c_task_name
     , table_owner => USER
     , table_name  => 'TEST_TABLE'
     , by_row      => TRUE
     , chunk_size  => 50000);

  dbms_parallel_execute.run_task (
       task_name      => c_task_name
     , sql_stmt       =>
          q'$ BEGIN update_test_table (:start_id, :end_id); END; $'
     , language_flag  => DBMS_SQL.native
     , parallel_level => 32);

  dbms_parallel_execute.drop_task(c_task_name);
END;
```

dbms_parallel_execute (7)

- create_chunks_by_rowid
  - by_row
    - TRUE: rows
    - FALSE: blocks
  - chunk_size
- create_chunks_by_number_col
  - table_column
- create_chunks_by_sql
  - sql_statement
  - by_rowid (TRUE/FALSE)

dbms_parallel_execute (7)

- create_chunks_by_rowid
  - by_row
    - TRUE: rows

Note: Keep in mind that Oracle performs COMMIT after finishing every chunk's process.

  - table_column
- create_chunks_by_sql
  - sql_statement
  - by_rowid (TRUE/FALSE)

Phase 2: parallelism (1)

- > 200.000 customers
- from 10 to 700.000 records per customer
- uneven distribution
    - cannot use ranges of customers IDs
- each thread process one customer
- 2 CPU, 16 core each
- 2 threads per CPU
- parallelism: 64
- custom SQL

Phase 2: parallelism (2)

- update procedure uses more parameters
- keep history of executions
  - create table for additional parameters and history

```
CREATE TABLE cust_task_master (
   master_task_id INT NOT NULL,
   execution_date DATE NOT NULL,
   CONSTRAINT cust_task_master_pk
      PRIMARY KEY (master_task_id)
);
```

Phase 2: parallelism (3)

```
CREATE TABLE cust_task_slaves (
   master_task_id INT NOT NULL,
   cust_id        INT NOT NULL,
   balance_date   DATE NOT NULL,
   account_id     INT NOT NULL,
   start_time     TIMESTAMP,
   end_time       TIMESTAMP,
   rows_processed INT,
   CONSTRAINT cust_task_slaves_pk
      PRIMARY KEY (master_task_id, cust_id),
   CONSTRAINT cust_task_slaves_fk
      FOREIGN KEY (master_task_id)
         REFERENCES cust_task_master (master_task_id)
);
```

Phase 2: parallelism (4)

```
CREATE OR REPLACE PACKAGE BODY cust_parallel AS

   c_task_name CONSTANT VARCHAR2(128) := 'UPDATE_CUST_PARALLEL';

   FUNCTION prepare_cust RETURN cust_task_master.master_task_id%TYPE IS
      cts_rec cust_task_slaves%ROWTYPE;
      ctm_rec cust_task_master%ROWTYPE;
   BEGIN
      SELECT cust_task_master_sq.nextval INTO ctm_rec.master_task_id FROM dual;
      ctm_rec.execution_date := SYSDATE;
      INSERT INTO cust_task_master VALUES ctm_rec;

      -- populate slaves recs with actual parameters
      INSERT INTO cust_task_slaves
         (cust_id, master_task_id, balance_date, account_id)
         SELECT /* actual SELECT here ... */;
      -- commit must be done!
      COMMIT;
      RETURN(ctm_rec.master_task_id);
   END;
```

Phase 2: parallelism (5)

```
PROCEDURE update_cust(p_master_id IN cust_task_master.master_task_id%TYPE,
                      p_cust_id   IN cust_task_slaves.cust_id%TYPE) IS
   l_rows INT := 0;
BEGIN
   log_me(p_master_id, 'cust_id:' || p_cust_id);
   UPDATE cust_task_slaves s
      SET s.start_time = systimestamp
    WHERE s.master_task_id = p_master_id
      AND s.cust_id = p_cust_id;
   COMMIT;

   -- actual processing
   FOR x_rec IN (SELECT *
                   FROM <my_tables>      mt,
                        cust_task_slaves cts
                  WHERE mt.cust_id = cts.cust_id
                    AND mt.balance_date = cts.balance_date
                    AND mt.account_id = cts.account_id
                    AND cts.master_task_id = p_master_id
                    AND cts.cust_id = p_cust_id)
   LOOP
      -- actual update of one customer here;
      l_rows := l_rows + 1;
   END LOOP;
   -- write end execution time
   UPDATE cust_task_slaves s
      SET s.end_time       = systimestamp,
          s.rows_processed = l_rows
    WHERE s.master_task_id = p_master_id
      AND s.cust_id = p_cust_id;
   COMMIT;
END;
```

Phase 2: parallelism (6)

```
PROCEDURE run_parallel(p_parallel_level IN INT := 64) IS
   l_master_id cust_task_master.master_task_id%TYPE;
BEGIN
   l_master_id := prepare_cust;
   dbms_parallel_execute.create_task(c_task_name);

   BEGIN
      -- create chunks (see USER_PARALLEL_EXECUTE_CHUNKS
      dbms_parallel_execute.create_chunks_by_sql(
         task_name => c_task_name,
         sql_stmt  =>
              'select master_task_id, cust_id from cust_task_slaves where master_task_id ='
           || l_master_id,
         by_rowid  => FALSE);

      -- run tasks (run actual update with cust_parallel.update_cust procedure=
      dbms_parallel_execute.run_task(
         task_name      => c_task_name,
         sql_stmt       => q'$ BEGIN cust_parallel.update_cust (:start_id, :end_id); END; $',
         language_flag  => dbms_sql.native,
         parallel_level => p_parallel_level);

      -- drop task after processing
      dbms_parallel_execute.drop_task(c_task_name);
   EXCEPTION
      WHEN OTHERS THEN
         dbms_parallel_execute.drop_task(c_task_name);
         raise_application_error(-20001, SQLERRM);
   END;
END;
```

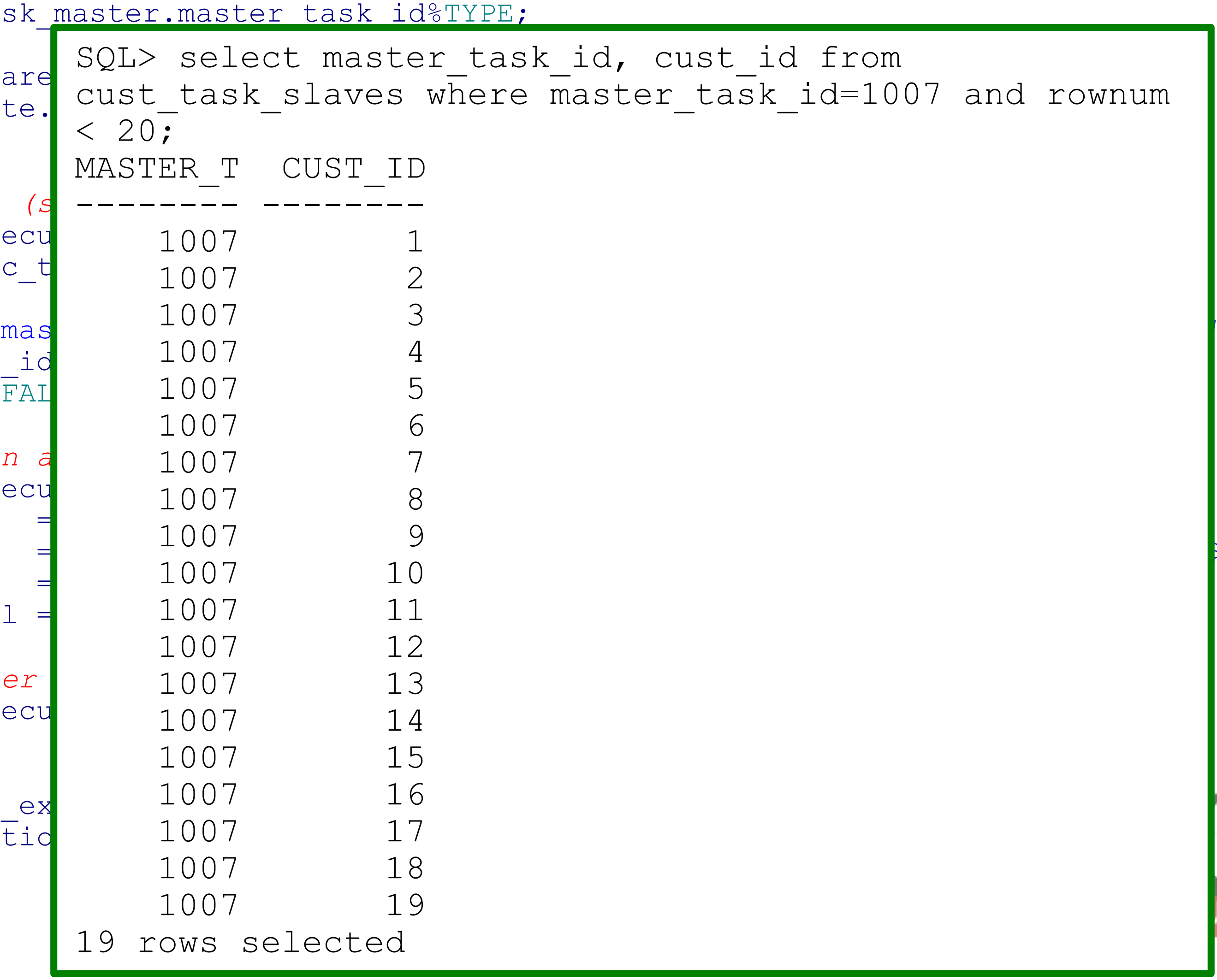
Phase 2: parallelism (6)

```
PROCEDURE run_parallel(p_parallel_level IN INT := 64) IS
   l_master_id cust_task_master.master_task_id%TYPE;
BEGIN
   l_master_id := prepare
   dbms_parallel_execute.

   BEGIN
      -- create chunks (s
      dbms_parallel_execu
         task_name => c_t
         sql_stmt  =>
              'select mas
           || l_master_id
         by_rowid  => FAL                                                   ,

      -- run tasks (run a
      dbms_parallel_execu
         task_name      =
         sql_stmt       =                                                  s',
         language_flag  =
         parallel_level =

      -- drop task after
      dbms_parallel_execu
   EXCEPTION
      WHEN OTHERS THEN
         dbms_parallel_ex
         raise_applicatio
   END;
END;
```



```
SQL> select master_task_id, cust_id from
cust_task_slaves where master_task_id=1007 and rownum
< 20;
MASTER_T  CUST_ID
-------- --------
    1007        1
    1007        2
    1007        3
    1007        4
    1007        5
    1007        6
    1007        7
    1007        8
    1007        9
    1007       10
    1007       11
    1007       12
    1007       13
    1007       14
    1007       15
    1007       16
    1007       17
    1007       18
    1007       19
19 rows selected
```

Phase 2: parallelism (7)

```
SQL> exec print_rec ('SELECT t.task_name, t.chunk_type, t.status, t.SQL_STMT,
t.LANGUAGE_FLAG, t.PARALLEL_LEVEL FROM user_parallel_execute_tasks t');

==============================================================================
______TASK_NAME:<UPDATE_CUST_PARALLEL>
_____CHUNK_TYPE:<NUMBER_RANGE>
_________STATUS:<PROCESSING>
_______SQL_STMT:< BEGIN cust_parallel.update_cust (:start_id, :end_id); END; >
__LANGUAGE_FLAG:<1>
_PARALLEL_LEVEL:<64>
==============================================================================
```

As na disku.

Phase 2: parallelism (8)

```
SQL> exec print_rec ('select c.status, c.START_ID, c.END_ID, c.START_TS,
c.END_TS, c.ERROR_MESSAGE from user_parallel_execute_chunks c where rownum < 2');

==============================================================================
______________STATUS:<PROCESSED>
____________START_ID:<1007>
______________END_ID:<4219>
____________START_TS:<29.09.13 09:16:13,628631>
______________END_TS:<29.09.13 09:16:14,645001>
_______ERROR_MESSAGE:<NULL>
==============================================================================
PL/SQL procedure successfully completed
```

Phase 2: parallelism (9)

```
SQL> exec print_rec ('select min (s.start_time) start_time, max (s.end_time)
end_time, sum (s.rows_processed) rows_processed from cust_task_slaves s where
s.master_task_id = 1007');

==============================================================================
__________START_TIME:<29.09.13 09:16:13,636550>
____________END_TIME:<29.09.13 09:20:32,188363>
______ROWS_PROCESSED:<2.478.054.065>
==============================================================================
PL/SQL procedure successfully completed
```

Conclusion

- many developers are unaware of dbms_parallel_execute
- works with Oracle SE (for now)
- better performance compared to parallel DML when chunks are based on ROWID or blocks
- less undo space required and minimizing the chance of ORA-01555
- better uniform distribution across parallel processes
- PL/SQL parallelism

ORA-03113: end-of-file on communication channel

Boris Oblak

Abakus plus d.o.o.

ORACLE CERTIFIED PROFESSIONAL

ORACLE Gold Partner

18. Strokovno srečanje

SIOUG 2013

14.-16. oktober 2013

Parallel executing

AS

Abakus

As na disku.