CERN IT Department

an Oracle – Castor story

from simple code to working code

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Sebastien Ponce, Database Developer's Workshop, July 8th 2008

CERN

Outline

- Foreword
  - what is CASTOR
  - why do we use DB
  - how do we use the DB
- A real life coding example
  - debugging and optimization of the selection of migration candidates

CASTOR

- is Cern Advanced STORage
  - handles all physics data at CERN and in 3 Tier 1s (10s of petaBytes)
  - deals with magnetic tapes and a level of cache on disk
- is a DB centric system holding all its state in ORACLE databases (namespace, cache,...)
  - lots of PL/SQL interfaced via OCCI
    - the CASTOR logic is PL/SQL code
  - small DB (few GBs, ~15 main tables)
  - very active DB (100s transactions/s)

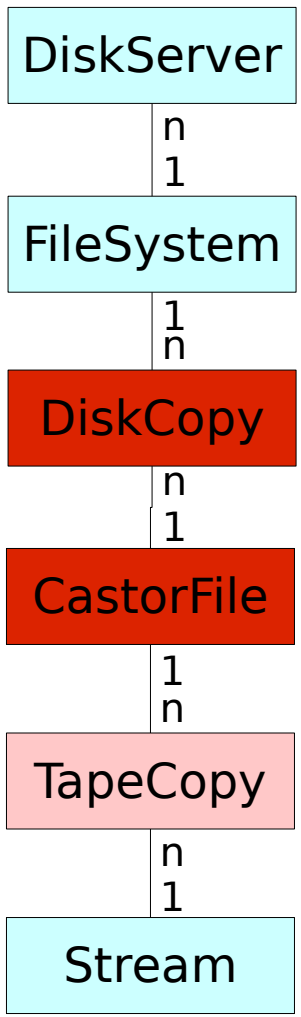
CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Example : migrations

- Problem :
  - find the best file to be migrated (i.e. written to tape) for a given stream (to a tape drive)
- Context :
  - a tape is rotating
  - more data is needed to keep its buffer full (otherwise tape will stop)
  - we need to find a file on the “best” file system possible (load balancing)

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

DB schema



	Size (rows)	Activity (modifs/day)
DiskServer	100s	~1
FileSystem	1000s	~1
DiskCopy	1 000 000s	1 000 000s
CastorFile	1 000 000s	100 000s
TapeCopy	10 000s	100 000s
Stream	10s	100s

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Not so naïve SQL

- SELECT ... FROM
  (SELECT ...
  FROM DiskServer, FileSystem, DiskCopy,
  TapeCopy, Stream
  WHERE ... (primary-foreign keys)
  AND DiskCopy.status = CANBEMIGR
  AND Stream.id = <myinput>
  ORDER BY f(FileSystem))
  WHERE rownum < 2;
- Note that ORDER BY + RowNum forces the use of a nested select in ORACLE

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Concurrency problems

- Our SQL is not “thread safe”
  - 2 streams asking concurrently for the best file to migrate may get the same one
  - on top, the 2 decisions won't “see” each other
    - because each decision modifies the ranking of the selected FileSystem
- We need to take a lock

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Locking granularity

- What is modified when a decision is taken ?
  - the weight of the selected file system
  - the weight of the other file systems on the same disk server
- So we need to lock all file systems of the selected diskserver
  - but this cannot be done atomically

Examples of bad locking

- SELECT ....
  FROM Table1
  WHERE ... <returns multiple rows>
  FOR UPDATE
- SELECT ....
  FROM Table1, Table2
  WHERE ... <returns a single row>
  FOR UPDATE
- In both cases, the locks are not taken atomically
  - so running twice this concurrently will create a dead lock for sure

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Proper locking

- In order to avoid bad locking of multiple rows, we need to serialize the locking code by taking first a “master” lock
- Natural solution : lock the DiskServer first
  - but we need to agree across all the software on the order of the locks
    - otherwise some other code will lock filesystem first and create dead locks
  - this also limits concurrency at the DiskServer level
    - while we just need a master lock, we may prevent other codes to run

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

LockTable

- Best choice is thus to implement a dedicated “LockTable”
- Note that we don't need full serialization when taking locks on the filesystems
  - because we know that the locks will be for those sharing a common disk server
- A master lock per DiskServer reduces the granularity of the locks
  - note that this implies 2 triggers to fill/clean up the LockTable on insertion/deletion of diskServers

Locking SQL

- SELECT * FROM LockTable
  WHERE id =
  (SELECT id FROM
  (SELECT DiskServer.id
  FROM DiskServer, FileSystem, DiskCopy,
  TapeCopy, Stream
  WHERE ... (primary-foreign keys)
  AND DiskCopy.status = CANBEMIGR
  AND Stream.id = <myinput>
  ORDER BY f(FileSystem))
  WHERE rownum < 2)
  FOR UPDATE;
- And then select the FileSystem, and finally the file to be migrated...

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Comments

- We have to do the full join for the selection of the diskserver because we need one with a file on it in the proper stream
- This join has to be executed very cleverly in order to not kill the DB
  - discussion about this point is coming
- We have to use 2 levels of nested selects
  - This actually triggers and ORACLE bug :-((

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

ORACLE bug

- Take this code :

```
SELECT id INTO tid FROM Table1 WHERE id =
  (SELECT * FROM
        (SELECT id FROM Table1
         WHERE status = 1 ORDER BY ...)
    WHERE RowNum < 2)
FOR UPDATE;
UPDATE Table1 SET status = 2 WHERE id = tid;
```

- Oracle will execute the nested selects first and potentially in parallel for several queries
- The top select will not be parallelized (lock)
  - but on commit, Oracle should revalidate the nested selects before restarting the second query

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

ORACLE bug (2)

- Oracle does revalidate the first nested select
  - but forgets about the second level one (probably because it is inside a pure select and thus does not need revalidation if you don't look upward)
- This allows to return the same result twice if you run this query twice concurrently !
  - the locking is still serialized however
- Hopefully for us, an update will work properly
  - so SELECT FROM LockTable FOR UPDATE becomes UPDATE LockTable SET id = id

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Back to optimization

- Take

```
SELECT max(gcWeight)
  FROM DiskCopy d, CastorFile c
WHERE d.castorfile = c.id;
```

- DiskCopy is indexed by gcWeight and the proper foreign key is defined
- Here is the execution plan :

#	Operation	Options	Object name	Mode	Cost	Bytes	Cardinality
DML	SELECT max (gcweight ...						
0	SELECT STATEMENT			ALL_ROWS	36941	36	1
1	SORT	AGGREGATE				36	1
2	HASH JOIN				36941	196981200	5471700
3	INDEX	FAST FULL SCAN	I_CASTORFILE_ID	ANALYZED	3084	45580080	6511440
4	TABLE ACCESS	FULL	DISKCOPY	ANALYZED	25410	158679300	5471700

What is happening ?

- Aggregate operations (max, order) are only applied after the selection is done
  - the selection part includes joins
    - which become full joins....
  - simple SELECT MAX(...) FROM Table does the same, even with a dedicated index !
- We found no way in Oracle to select the “best” candidate without full table scan
  - So we need to do it manually... and on a much more complex query...

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Efficient query

- order filesystems
  - that only implies FileSystem and DiskServer tables (~1000 rows)
- loop on best filesystems
  - find a diskcopy in proper status
    - use index on status & filesystem
  - and check the link to the stream
    - 2 index lookups in Tapecopy & Stream
  - probability to be linked to right stream is ~20-50% -> very efficient

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Implementation

- the loop on filesystem is actually a problem
  - it would lead to lock several of them... causing dead locks (first bad locking case)
- So we need to select straight the best filesystem having candidates for migration without doing the full join !
  - denormalization is used
    - the NbTapeCopiesInFS table is defined
    - holding the number of candidates for a given FileSystem and Stream
    - we keep it up to date via 7 triggers...
      - yes, a real pain, took years to debug !

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Final SQL ?

- UPDATE LockTable SET id = id
  WHERE id =
   (SELECT id FROM
     (SELECT DiskServer.id
       FROM DiskServer, FileSystem,
            NbTapeCopiesInFS n
      WHERE ... (primary-foreign keys)
        AND n.Stream = <myinput>
      ORDER BY f(FileSystem))
    WHERE rownum < 2)
  FOR UPDATE;
- And then select the FileSystem, and finally the file to be migrated...
- And have an extra 9 triggers...

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

And there was more

- The “final” statement worked for some time... and then changed execution plan
- This is something very usual in CASTOR
  - typical change is to not use anymore an index and go for full table scan
  - this kills the DB and CASTOR in general
- We detect it using AWR reports and add hints to our statements
- Here we have

```
SELECT /*+ FIRST_ROWS(1)  LEADING(D T ST) */ ...
FROM DiskCopy D, TapeCopy T,
     Stream2TapeCopy ST
```

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

And there was more(2)

- Optimization of the scanning of DiskCopy table was necessary as very few rows are in the proper status (~.5%)
  - function based indexes were used
- Some table could grow on particular situations (exceptional load) and did not shrink automatically
  - regular table shrinking was added
  - this collides with function based indexes
  - So we switched to partitioning on status and went back to regular indexes

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Lessons

- even Oracle may have simple bugs...
- concurrent queries need careful locking
  - no atomicity when taking several locks
  - lock ordering and “master” locks may help
- many queries need “manual” optimization
  - most of the time using simple hints
  - or denormalization and triggers (cumbersome)
- In our case, shrinking tables is mandatory
- At the end, ORACLE is extremely efficient !
  - it is just not as simple to use as one may think

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it