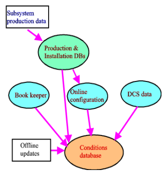
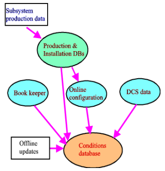


Database solutions to ATLAS specific application requirements (real-life examples)

Gancho Dimitrov (CERN)

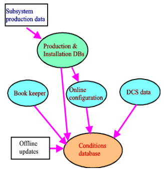
Outline



- ATLAS databases – main roles
- Facts about the ATLAS databases content at CERN
- Case studies and solutions

  Used techniques :
  - Partitioning types – range, automatic interval , reference
  - Data sliding windows
  - Data aggregation
  - Scheduled index maintenance
  - Result set caching
  - Stats gathering settings
  - Use of Active Data Guard (ADG)
  - Others …
- Important metrics for monitoring performance
- Conclusions

ATLAS database servers

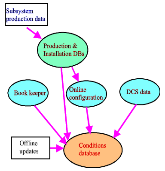


Database / info	ATONR	ATLR	ADCR	ATLARC	INTR	INT8R
Main database role	Online data taking	Post data taking analysis	Grid jobs and file management	Event metadata (TAGS)	SW and replica tests	SW and replica tests
Current Oracle ver.	11.2.0.3	11.2.0.3	11.2.0.3	11.2.0.3	11.2.0.3	11.2.0.3
Number of DB nodes	3	4	4	2	2	2

=> ATONR is of OLTP (Online Transaction Processing) workload type with a sliding window of an year for PVSS data (equivalent to about 4 TB)

=> ATLR and ADCR host many versatile applications. For that reason these databases are the most difficult to deal with.

=> ATLARC is of type data warehouse.



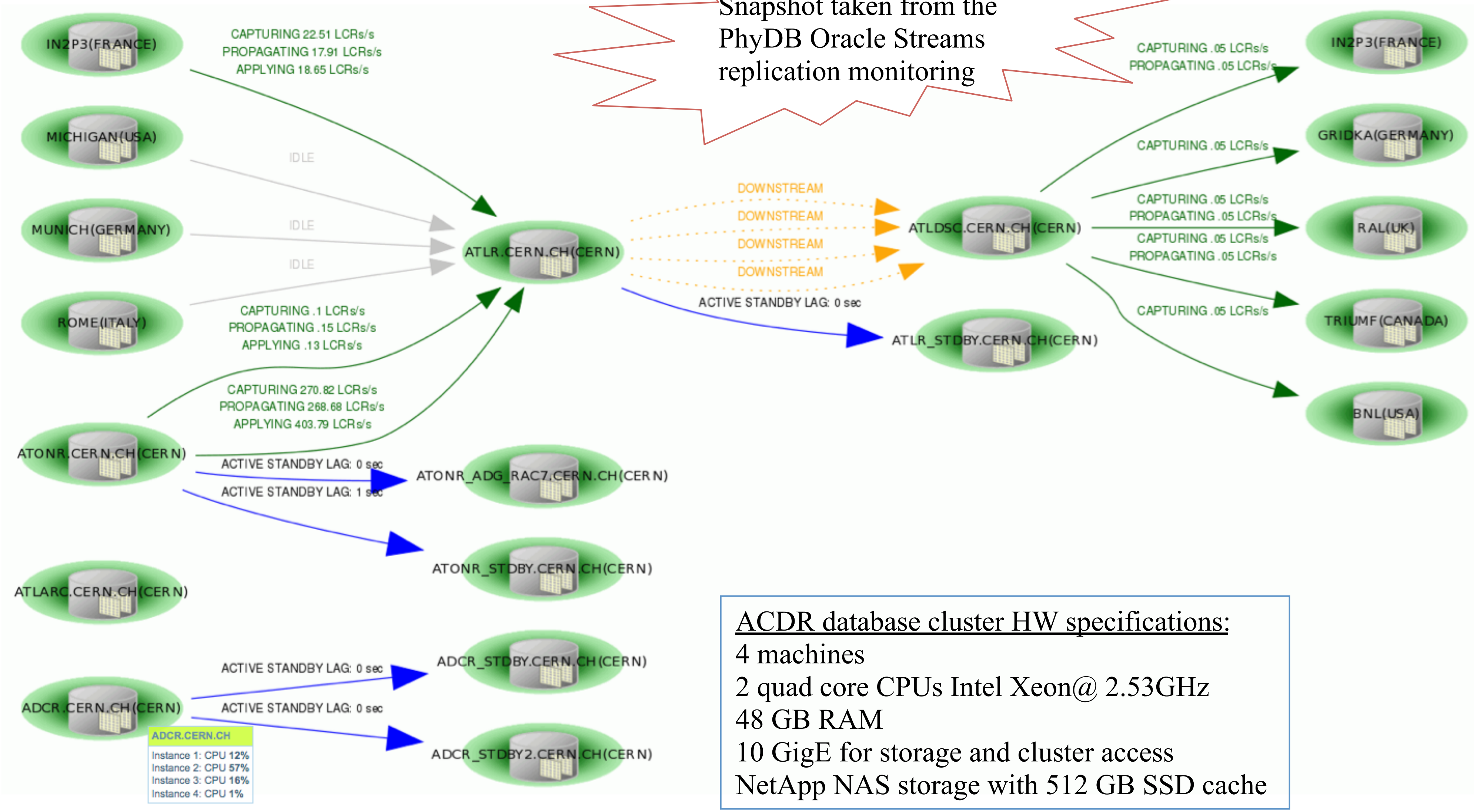
The ATLAS databases at CERN (facts)

The table represents the current state with an impressive number of database schemes. In the common case each schema is related to a dedicated client application except COOL, PVSS and PVSSCONF schemas as each account maps to a sub-detector.

Database / Metric	ATONR	ATLR	ADCR	ATLARC
# DB schemes	69	161	12	62 (incl 23 arch)
# Tables - Non partitioned - Partitioned	37358 459	64960 1051	532 28	15453 10223
Volume - Table segments - Index segments - LOB segments	2,9 TB 2,7 TB 0,8 TB	7,2 TB 10,1 TB 1,1 TB	9,5 TB 5 TB (incl. IOTs) 0,6 TB	9,4 TB 6,3 TB 0 TB
Table with largest number of rows	0.9 billion	2.1 billion	6.2 billion	5.2 billion

* here billion to be considered with the American convention, therefore 9 zeros (milliard in Europe)

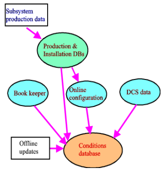
ATLAS databases topology (June 2013)



Snapshot taken from the
PhyDB Oracle Streams
replication monitoring

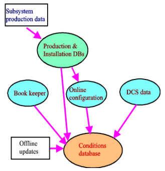
ACDR database cluster HW specifications:
4 machines
2 quad core CPUs Intel Xeon@ 2.53GHz
48 GB RAM
10 GigE for storage and cluster access
NetApp NAS storage with 512 GB SSD cache

ATLAS applications



<u>Short list of ATLAS applications that will be referred often into the following slides</u>

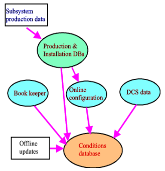
- PanDA – Production and Distributed Analysis system
- JEDI – Job Definition and Execution Interface
- PVSS - Process Visualization and Steering System
- NICOS - Nightly Build System
- DQ2 - File management and distribution system (Don Quixote 2)
- Rucio - File management and distribution system (successor of DQ2, Rucio is name of Sancho Panza's donkey)



Case study 1

PanDA

PanDA system



- The PanDA system is the ATLAS workload management system for production and user analysis jobs
- Originally based on a MySQL database schema, migrated in 2008 to Oracle at CERN.
- Challenges:

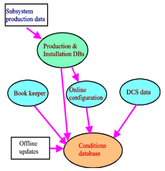
  - PanDA system has to manage millions of grid jobs daily

  - Changes into jobs statuses, sites loads have to be reflected on the database fast enough. Fast data retrievals to the PanDA server and monitor are key requirements

  - Cope with spikes of user’s workload

  - DB system has to deal efficiently with two different workloads: transactional (PanDA server) and data warehouse load (PanDA monitor)

The PANDA ‘operational’ and ‘archive’ data



- All information relevant to a single job is stored in 4 major tables. The most important stats are kept separately from the other space consuming attributes like job parameters, input and output files, ... etc.

- The ‘operational’ data is kept in a separate schema which hosts jobs of the most recent 3 days. Jobs that get status ‘finished’ or ‘failed’ are moved to an archive PANDA schema.

  ATLAS_PANDA => ATLAS_PANDAARCH

BUT

In Nov 2010 the whole PanDA system got stuck due to :

- increased user workload on the database

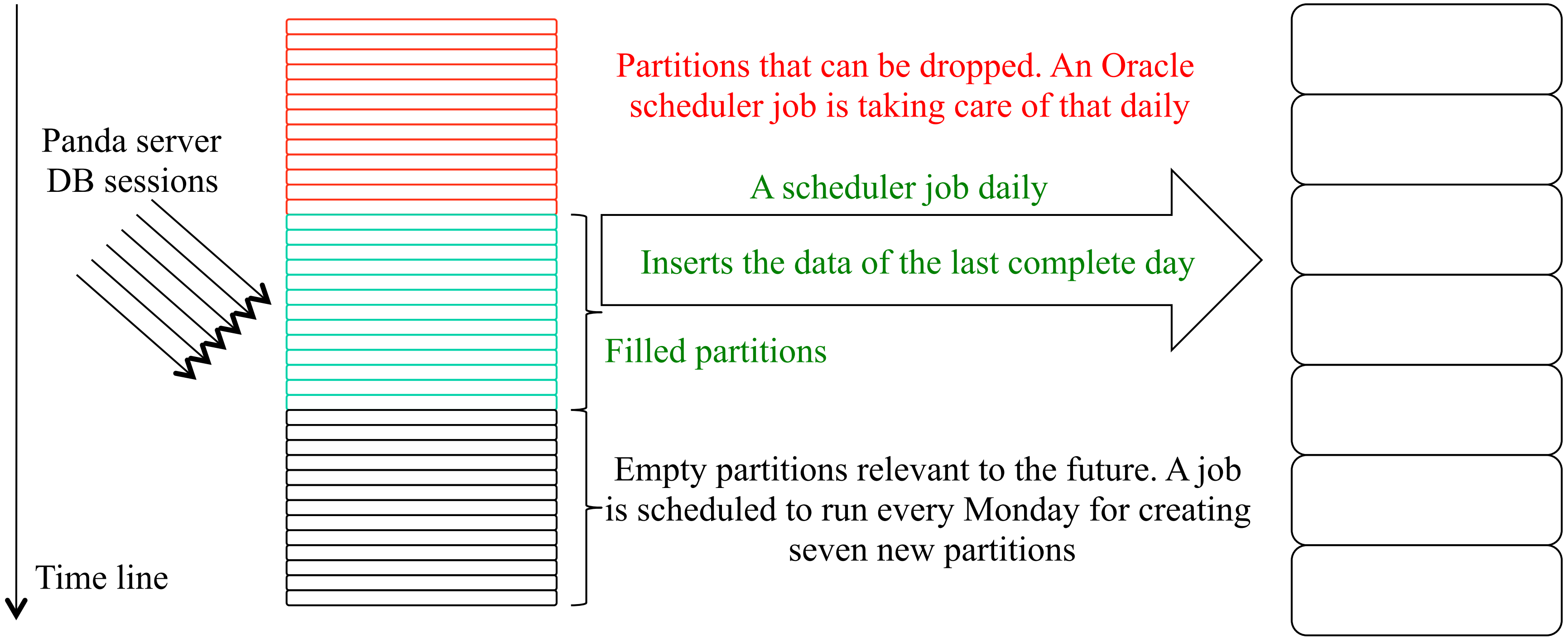
- highly fragmented tables because the PANDA=>PANDAARCH data move was done from the PanDA server with conventional row insertion into the ‘archive’ data and deletion from the ‘operational’ data.

- the hot operational data and indexes could not fit into the buffer pool

New PanDA data segments organization (as of Dec 2010)

ATLAS_PANDA JOB, PARAMS, META and FILES tables: partitioned on a 'modificationtime' column.
Each partition covers a time range of a day

ATLAS_PANDAARCH archive tables partitioned on the 'modificationtime' column.
Some table have defined partitions each covering three days window, others a time range of a month



Partitions that can be dropped. An Oracle scheduler job is taking care of that daily

Panda server DB sessions

A scheduler job daily

Inserts the data of the last complete day

Filled partitions

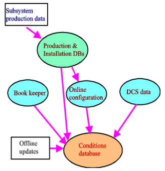
Empty partitions relevant to the future. A job is scheduled to run every Monday for creating seven new partitions

Time line

Certain PanDA tables have defined data sliding window of 3 days, others 30 days.

This natural approach showed to be adequate and not resource demanding !

Important: For being sure that job information will be not deleted without being copied in the PANDAARCH schema, a special verification PLSQL procedure is taking place before the partition drop event!

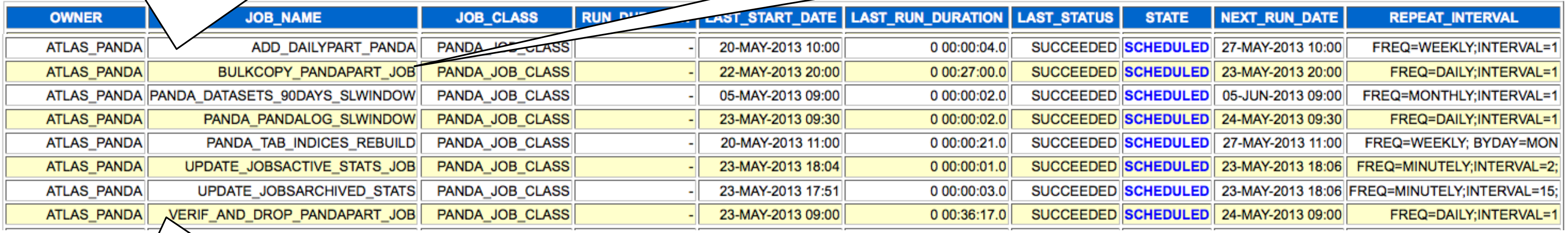


PANDA=>PANDAARCH data flow machinery

- The PANDA => PANDAARCH data flow is sustained by a set of scheduler jobs on the Oracle server which execute a logic encoded in PL/SQL procedures

Weekly job which creates daily partitions relevant to the near future days

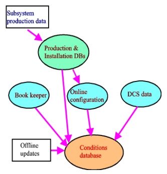
Daily job which copies data of a set of partitions from PANDA to PANDAARCH



OWNER	JOB_NAME	JOB_CLASS	RUN_DURATION	LAST_START_DATE	LAST_RUN_DURATION	LAST_STATUS	STATE	NEXT_RUN_DATE	REPEAT_INTERVAL
ATLAS_PANDA	ADD_DAILYPART_PANDA	PANDA_JOB_CLASS	-	20-MAY-2013 10:00	0 00:00:04.0	SUCCEEDED	SCHEDULED	27-MAY-2013 10:00	FREQ=WEEKLY;INTERVAL=1
ATLAS_PANDA	BULKCOPY_PANDAPART_JOB	PANDA_JOB_CLASS	-	22-MAY-2013 20:00	0 00:27:00.0	SUCCEEDED	SCHEDULED	23-MAY-2013 20:00	FREQ=DAILY;INTERVAL=1
ATLAS_PANDA	PANDA_DATASETS_90DAYS_SLWINDOW	PANDA_JOB_CLASS	-	05-MAY-2013 09:00	0 00:00:02.0	SUCCEEDED	SCHEDULED	05-JUN-2013 09:00	FREQ=MONTHLY;INTERVAL=1
ATLAS_PANDA	PANDA_PANDALOG_SLWINDOW	PANDA_JOB_CLASS	-	23-MAY-2013 09:30	0 00:00:02.0	SUCCEEDED	SCHEDULED	24-MAY-2013 09:30	FREQ=DAILY;INTERVAL=1
ATLAS_PANDA	PANDA_TAB_INDICES_REBUILD	PANDA_JOB_CLASS	-	20-MAY-2013 11:00	0 00:00:21.0	SUCCEEDED	SCHEDULED	27-MAY-2013 11:00	FREQ=WEEKLY; BYDAY=MON
ATLAS_PANDA	UPDATE_JOBSACTIVE_STATS_JOB	PANDA_JOB_CLASS	-	23-MAY-2013 18:04	0 00:00:01.0	SUCCEEDED	SCHEDULED	23-MAY-2013 18:06	FREQ=MINUTELY;INTERVAL=2;
ATLAS_PANDA	UPDATE_JOBSARCHIVED_STATS	PANDA_JOB_CLASS	-	23-MAY-2013 17:51	0 00:00:03.0	SUCCEEDED	SCHEDULED	23-MAY-2013 18:06	FREQ=MINUTELY;INTERVAL=15;
ATLAS_PANDA	VERIF_AND_DROP_PANDAPART_JOB	PANDA_JOB_CLASS	-	23-MAY-2013 09:00	0 00:36:17.0	SUCCEEDED	SCHEDULED	24-MAY-2013 09:00	FREQ=DAILY;INTERVAL=1

Daily job which verifies that all rows of certain partition have been copied to PANDAARCH and drops the PANDA partition if the above is true.

Benefits after the data segment changes

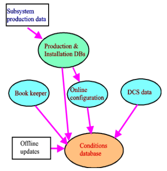


- High scalability: the Panda jobs data copy and deletion is done on **table partition level instead on row level as before**

- Removing the already copied data is not IO demanding (very little redo and does not produce undo) as this is a simple Oracle operation over a table segment and its relevant index segments (alter table … drop partition …)

- Fragmentation in the table segments is avoided. Much better space utilization and caching in the buffer pool

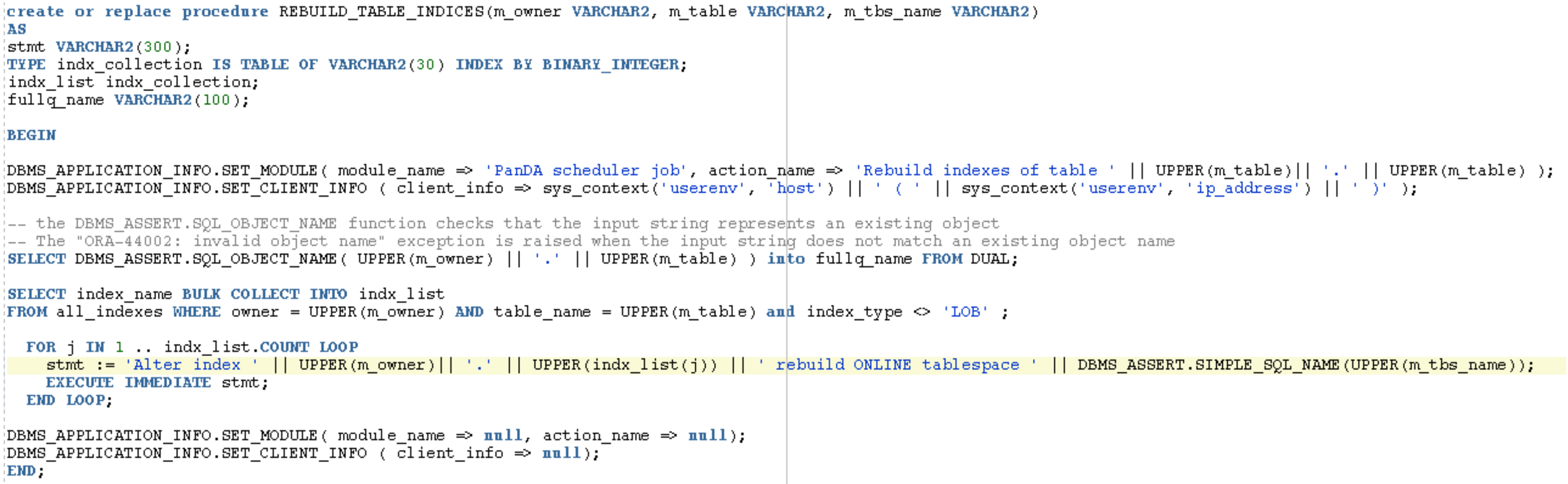
- No need for indexes rebuild or coalesce operations for these partitioned tables.



Other DB techniques used in PanDA (1)

Regular index maintenance (rebuids triggered by a scheduler job) on the tables with high transaction activity

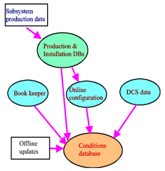
Note: fuctionality of the DBMS_ASSERT package is used for validating the procedure input values – a measure against SQL injection



```
create or replace procedure REBUILD_TABLE_INDICES(m_owner VARCHAR2, m_table VARCHAR2, m_tbs_name VARCHAR2)
AS
stmt VARCHAR2(300);
TYPE indx_collection IS TABLE OF VARCHAR2(30) INDEX BY BINARY_INTEGER;
indx_list indx_collection;
fullq_name VARCHAR2(100);

BEGIN

DBMS_APPLICATION_INFO.SET_MODULE( module_name => 'PanDA scheduler job', action_name => 'Rebuild indexes of table ' || UPPER(m_table)|| '.' || UPPER(m_table) );
DBMS_APPLICATION_INFO.SET_CLIENT_INFO ( client_info => sys_context('userenv', 'host') || ' ( ' || sys_context('userenv', 'ip_address') || ' )' );

-- the DBMS_ASSERT.SQL_OBJECT_NAME function checks that the input string represents an existing object
-- The "ORA-44002: invalid object name" exception is raised when the input string does not match an existing object name
SELECT DBMS_ASSERT.SQL_OBJECT_NAME( UPPER(m_owner) || '.' || UPPER(m_table) ) into fullq_name FROM DUAL;

SELECT index_name BULK COLLECT INTO indx_list
FROM all_indexes WHERE owner = UPPER(m_owner) AND table_name = UPPER(m_table) and index_type <> 'LOB' ;

  FOR j IN 1 .. indx_list.COUNT LOOP
    stmt := 'Alter index ' || UPPER(m_owner)|| '.' || UPPER(indx_list(j)) || ' rebuild ONLINE tablespace ' || DBMS_ASSERT.SIMPLE_SQL_NAME(UPPER(m_tbs_name));
    EXECUTE IMMEDIATE stmt;
  END LOOP;

DBMS_APPLICATION_INFO.SET_MODULE( module_name => null, action_name => null);
DBMS_APPLICATION_INFO.SET_CLIENT_INFO ( client_info => null);
END;
```

Note:

These regular index rebuilds for PanDA is used for well selected tables that host few million rows – NOT on tables with 100s of million rows and above.

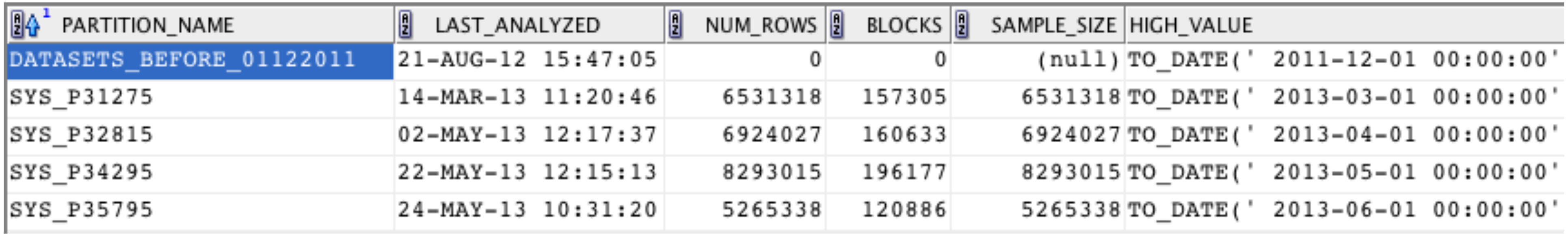


Other DB techniques used in PanDA (2)

Automatic interval parititioning

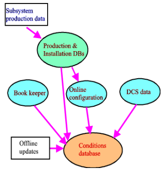
Partition is created automatically when a user transaction imposes a need for it (e.g. user inserts a row with a timestamp for which a partition does not yet exist)

```
CREATE TABLE table_name ( … list of columns …)
PARTITION BY RANGE(my_tstamp) INTERVAL(NUMTODSINTERVAL(1,’MONTH’))
( PARTITION data_before_01122011 VALUES LESS THAN
(TO_DATE('01-12-2011', 'DD-MM-YYYY')) )
```



PARTITION_NAME	LAST_ANALYZED	NUM_ROWS	BLOCKS	SAMPLE_SIZE	HIGH_VALUE
DATASETS_BEFORE_01122011	21-AUG-12 15:47:05	0	0	(null)	TO_DATE(' 2011-12-01 00:00:00'
SYS_P31275	14-MAR-13 11:20:46	6531318	157305	6531318	TO_DATE(' 2013-03-01 00:00:00'
SYS_P32815	02-MAY-13 12:17:37	6924027	160633	6924027	TO_DATE(' 2013-04-01 00:00:00'
SYS_P34295	22-MAY-13 12:15:13	8293015	196177	8293015	TO_DATE(' 2013-05-01 00:00:00'
SYS_P35795	24-MAY-13 10:31:20	5265338	120886	5265338	TO_DATE(' 2013-06-01 00:00:00'

In PanDA and other ATLAS applications interval partitioning is very handy for transient type of data where we impose a policy of agreed DATA SLIDING WINDOW (however partition removal is done via a home made PLSQL code)



Other DB techniques used in PanDA and DQ2 (3)

Result set caching

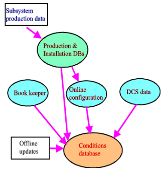
This technique was used on well selected set of PanDA server queries (and on the DQ2 system on PLSQL functions) - useful in cases where data do not change often, but is queried on a frequent basis.

As you know, the best metric to consider when tuning queries, is the number of Oracle block reads per execution. Queries for which result has been gotten from the result cache shows ‘buffer block reads = 0’

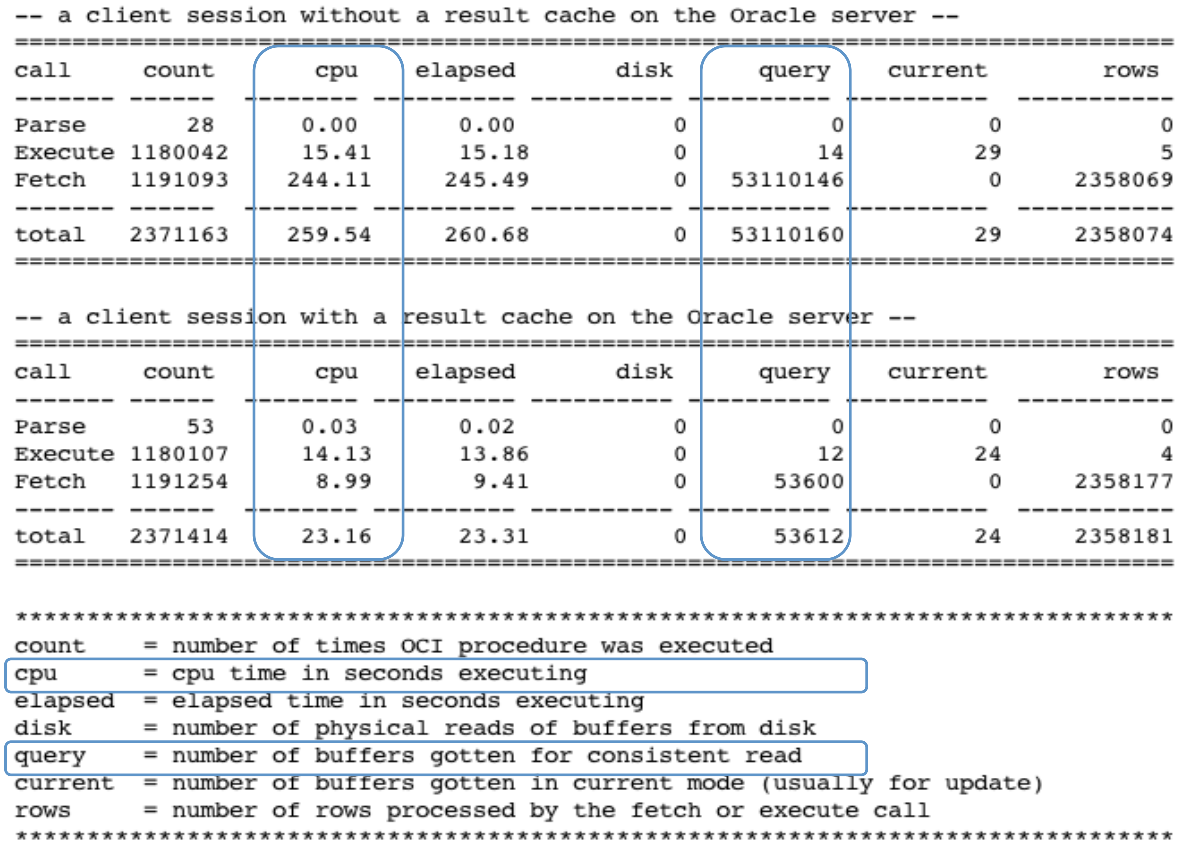
Oracle sends back to the client a cached result if the result has not been changed meanwhile by any transaction, thus improving the performance and scalability

The statistics shows that 95% of the executions of the PanDA server queries (17 distinct queries with this cache option on) were resolved from the result set cache and 99% of the calls to the DDM stored PL/SQL functions (having a rate of 3 billion calls per month).

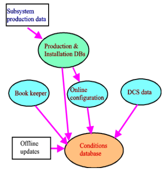
Other DB techniques used in PanDA and DQ2 (3 cont)



Event tracing on client sessions that read 1000 times each of the 1179 rows from a production configuration table showed that in the case of result caching the CPU time is only 23 sec while for the case of disabled result case it is 260 sec). The CPU saving is of factor 9 for that particular case.



```
-- a client session without a result cache on the Oracle server --
===================================================================================
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse       28      0.00       0.00          0          0          0           0
Execute 1180042    15.41      15.18          0         14         29           5
Fetch   1191093   244.11     245.49          0   53110146          0     2358069
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total   2371163   259.54     260.68          0   53110160         29     2358074
===================================================================================

-- a client session with a result cache on the Oracle server --
===================================================================================
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse       53      0.03       0.02          0          0          0           0
Execute 1180107    14.13      13.86          0         12         24           4
Fetch   1191254     8.99       9.41          0      53600          0     2358177
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total   2371414    23.16      23.31          0      53612         24     2358181
===================================================================================

***********************************************************************************
count    = number of times OCI procedure was executed
cpu      = cpu time in seconds executing
elapsed  = elapsed time in seconds executing
disk     = number of physical reads of buffers from disk
query    = number of buffers gotten for consistent read
current  = number of buffers gotten in current mode (usually for update)
rows     = number of rows processed by the fetch or execute call
***********************************************************************************
```



Other DB techniques used in PanDA (4)

Data aggregation for fast result devilery

PanDA server has to be able to get instantaneously details on the current activity at any ATLAS computing site (277 sites) – e.g. number of jobs at any site with particular priority, processing type, status , working group, …etc

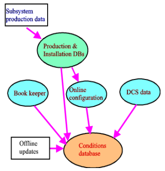
This is vital for justifying on which site the new coming jobs is best to be routed.

Query execution addresing the above requirement showed to be CPU expensive because of high frequency execution.

Solution: A table with aggregated stats is re-populated by a PLSQL procedure on an interval of 2 min by an Oracle scheduler job (usual elapsed time 1-2 sec)

The initial approach relied on a Materialized view (MV), but it showed to be NOT reliable because the MV refresh interval relies on the old DBMS_JOB package

Other DB techniques used in PanDA (5)

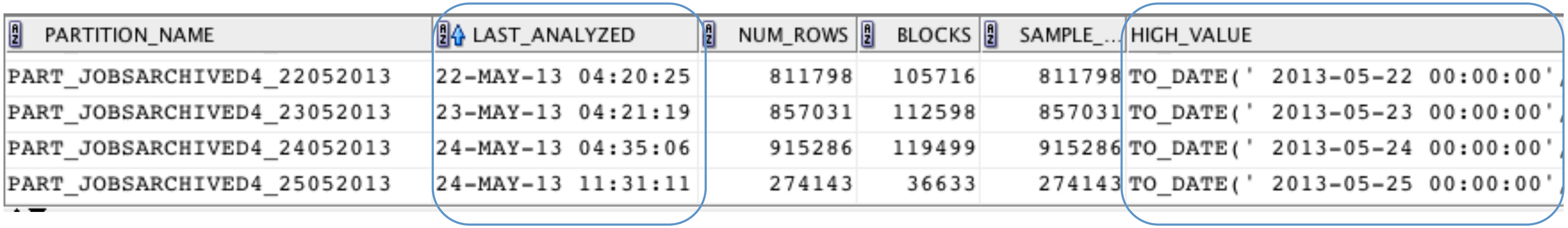


Customized table settings for the Oracle stats gathering

Having up-to-date statistics on tables data is essential for having optimal queries' data access path. We take advantage from statistics collection on partitioned tables called **incremental statistics gathering**.

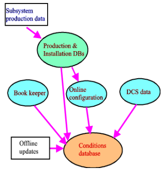
Oracle spends time and resources on collecting statistics only on partitions which are transactional active and computes the global table statistics using the previously ones in an incremental way.

exec DBMS_STATS.SET_TABLE_PREFS ('ATLAS_PANDA', 'JOBSARCHIVED4', 'INCREMENTAL', 'TRUE');



PARTITION_NAME	LAST_ANALYZED	NUM_ROWS	BLOCKS	SAMPLE_...	HIGH_VALUE
PART_JOBSARCHIVED4_22052013	22-MAY-13 04:20:25	811798	105716	811798	TO_DATE(' 2013-05-22 00:00:00'
PART_JOBSARCHIVED4_23052013	23-MAY-13 04:21:19	857031	112598	857031	TO_DATE(' 2013-05-23 00:00:00'
PART_JOBSARCHIVED4_24052013	24-MAY-13 04:35:06	915286	119499	915286	TO_DATE(' 2013-05-24 00:00:00'
PART_JOBSARCHIVED4_25052013	24-MAY-13 11:31:11	274143	36633	274143	TO_DATE(' 2013-05-25 00:00:00'

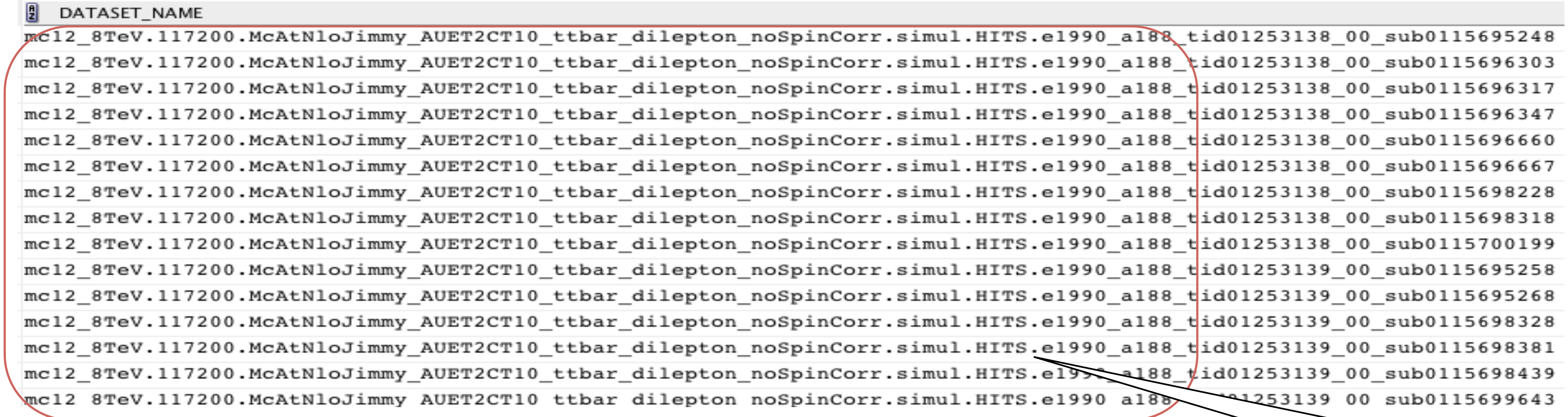
Other DB techniques used in PanDA and DQ2 (5 cont.)



Customized stats gathering settings - histograms

There are cases where histograms for column data distribution lead Oracle optimizer to a sub-optimal query execution plan.

In particular histograms could provide misleading information for columns hosting long strings with common data pattern. When Oracle creates a histogram on a VARCHAR2 column it only considers the first 32 characters in the column and thus wrong cardinality could be considered (especially for wildcard searches)

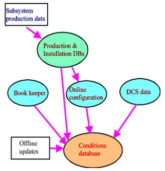


```
DATASET_NAME
mc12_8TeV.117200.McAtNloJimmy_AUET2CT10_ttbar_dilepton_noSpinCorr.simul.HITS.e1990_a188_tid01253138_00_sub0115695248
mc12_8TeV.117200.McAtNloJimmy_AUET2CT10_ttbar_dilepton_noSpinCorr.simul.HITS.e1990_a188_tid01253138_00_sub0115696303
mc12_8TeV.117200.McAtNloJimmy_AUET2CT10_ttbar_dilepton_noSpinCorr.simul.HITS.e1990_a188_tid01253138_00_sub0115696317
mc12_8TeV.117200.McAtNloJimmy_AUET2CT10_ttbar_dilepton_noSpinCorr.simul.HITS.e1990_a188_tid01253138_00_sub0115696347
mc12_8TeV.117200.McAtNloJimmy_AUET2CT10_ttbar_dilepton_noSpinCorr.simul.HITS.e1990_a188_tid01253138_00_sub0115696660
mc12_8TeV.117200.McAtNloJimmy_AUET2CT10_ttbar_dilepton_noSpinCorr.simul.HITS.e1990_a188_tid01253138_00_sub0115696667
mc12_8TeV.117200.McAtNloJimmy_AUET2CT10_ttbar_dilepton_noSpinCorr.simul.HITS.e1990_a188_tid01253138_00_sub0115698228
mc12_8TeV.117200.McAtNloJimmy_AUET2CT10_ttbar_dilepton_noSpinCorr.simul.HITS.e1990_a188_tid01253138_00_sub0115698318
mc12_8TeV.117200.McAtNloJimmy_AUET2CT10_ttbar_dilepton_noSpinCorr.simul.HITS.e1990_a188_tid01253138_00_sub0115700199
mc12_8TeV.117200.McAtNloJimmy_AUET2CT10_ttbar_dilepton_noSpinCorr.simul.HITS.e1990_a188_tid01253139_00_sub0115695258
mc12_8TeV.117200.McAtNloJimmy_AUET2CT10_ttbar_dilepton_noSpinCorr.simul.HITS.e1990_a188_tid01253139_00_sub0115695268
mc12_8TeV.117200.McAtNloJimmy_AUET2CT10_ttbar_dilepton_noSpinCorr.simul.HITS.e1990_a188_tid01253139_00_sub0115698328
mc12_8TeV.117200.McAtNloJimmy_AUET2CT10_ttbar_dilepton_noSpinCorr.simul.HITS.e1990_a188_tid01253139_00_sub0115698381
mc12_8TeV.117200.McAtNloJimmy_AUET2CT10_ttbar_dilepton_noSpinCorr.simul.HITS.e1990_a188_tid01253139_00_sub0115698439
mc12_8TeV.117200.McAtNloJimmy_AUET2CT10_ttbar_dilepton_noSpinCorr.simul.HITS.e1990_a188_tid01253139_00_sub0115699643
```

Common data pattern in the dataset names

e.g. forbid histograms for all columns of a particular table.

exec DBMS_STATS.SET_TABLE_PREFS('ATLAS_PANDA', 'FILESTABLE4','METHOD_OPT', 'FOR ALL COLUMNS SIZE 1');



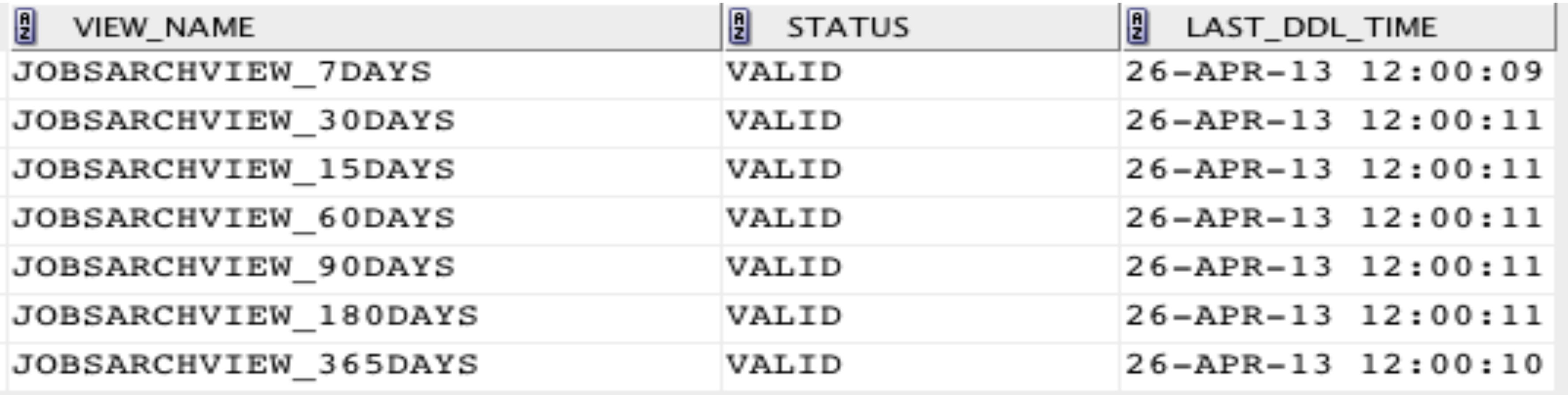
PanDA complete archive

- PanDA full archive now hosts information of 900 million jobs – all jobs since the job system start in 2006
- Studies on the WHERE clauses of the PanDA monitor queries resulted in:

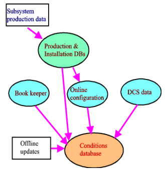
- revision of the indexes: replacement with more approriate multi-column ones so that less tables blocks get accessed

- dynamic re-definition of different time range views in order to protect the large PanDA archive tables from time range unconstrained queries and **to avoid time comparison on row level.**

The views comprise set of partitions based on ranges of 7, 15, 30, 60, 90, 180 and 365 days. These views are shifting windows with defined ranges and the underlying partition list is updated daily automatically via a scheduler job.

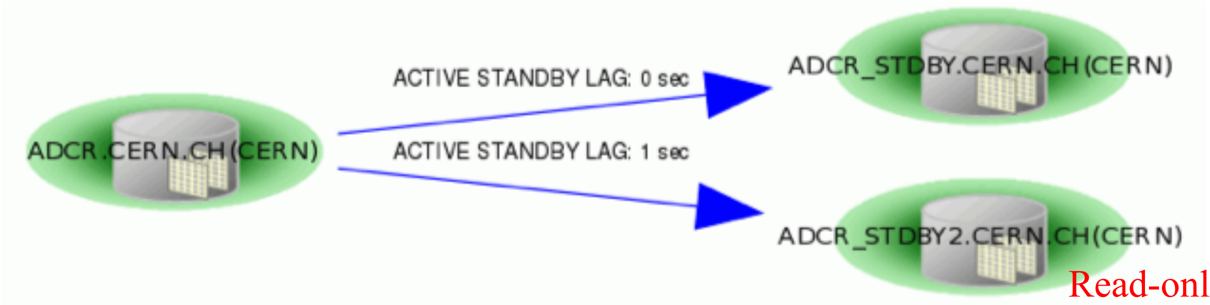


VIEW_NAME	STATUS	LAST_DDL_TIME
JOBSARCHVIEW_7DAYS	VALID	26-APR-13 12:00:09
JOBSARCHVIEW_30DAYS	VALID	26-APR-13 12:00:11
JOBSARCHVIEW_15DAYS	VALID	26-APR-13 12:00:11
JOBSARCHVIEW_60DAYS	VALID	26-APR-13 12:00:11
JOBSARCHVIEW_90DAYS	VALID	26-APR-13 12:00:11
JOBSARCHVIEW_180DAYS	VALID	26-APR-13 12:00:11
JOBSARCHVIEW_365DAYS	VALID	26-APR-13 12:00:10



Potential use of ADG from PanDA monitor

- ADCR database has two standby databases:
  - Data Guard for disaster recovery and backup offloading
  - Active Data Guard (ADCR_ADG) for read-only replica


ADCR.CERN.CH(CERN)
ACTIVE STANDBY LAG: 0 sec
ACTIVE STANDBY LAG: 1 sec
ADCR_STDBY.CERN.CH(CERN)
ADCR_STDBY2.CERN.CH(CERN)

Read-only replica

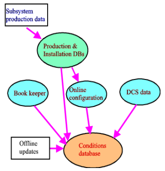
- PanDA monitor can benefit from the Active Data Guard (ADG) resources

  It is planned that PanDA monitor sustains two connection pools:

  - to the primary database ADCR

  - to ADCR's ADG

  The idea is queries that span on time ranges larger than certain threshold to be resolved from the ADG where we can afford several paralell slave processes per user query.

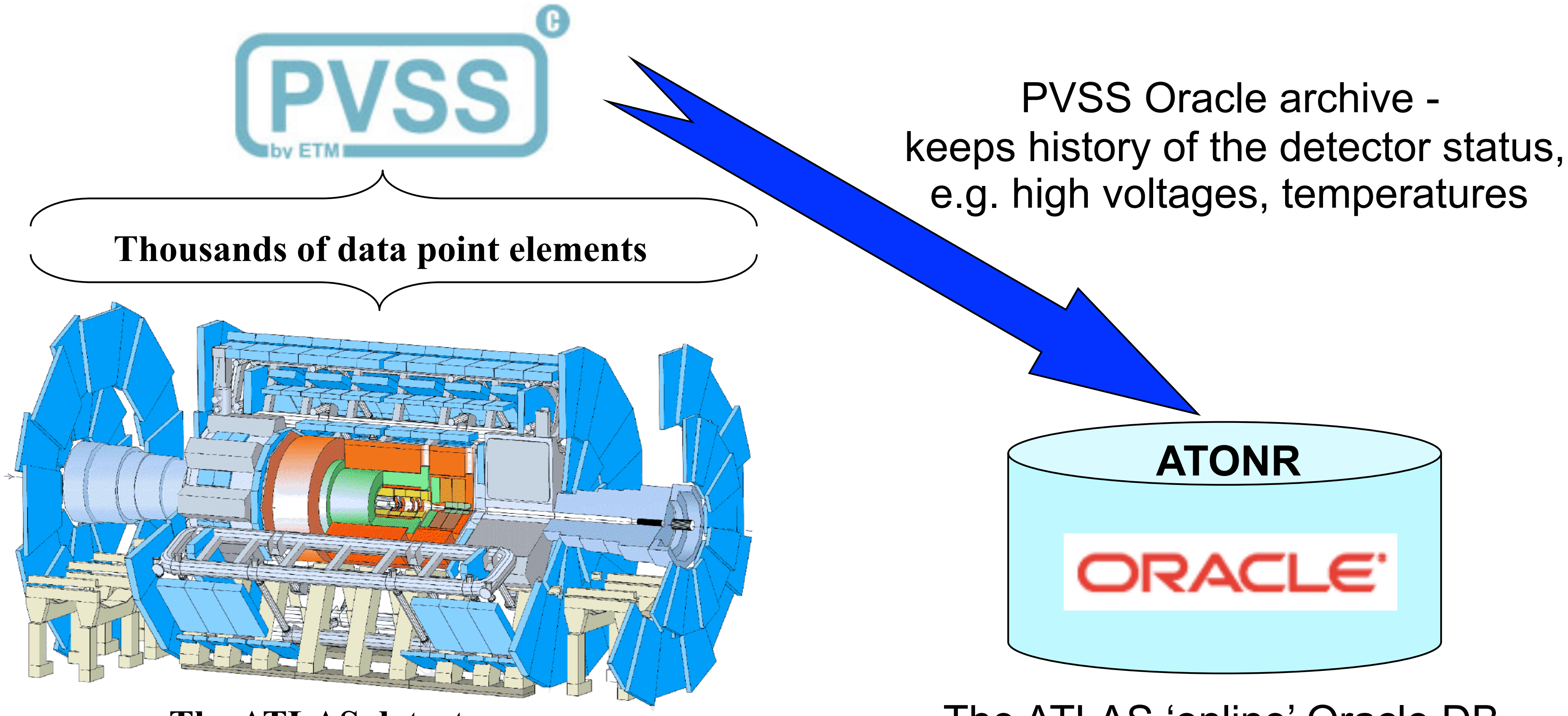


Case study 2

PVSS

The PVSS system and its use in ATLAS

PVSS (Process Visualization and Steering System) is a control and data acquisition system being in use in the LHC experiments since year 2000.



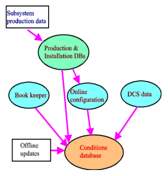
Thousands of data point elements

PVSS Oracle archive -
keeps history of the detector status,
e.g. high voltages, temperatures

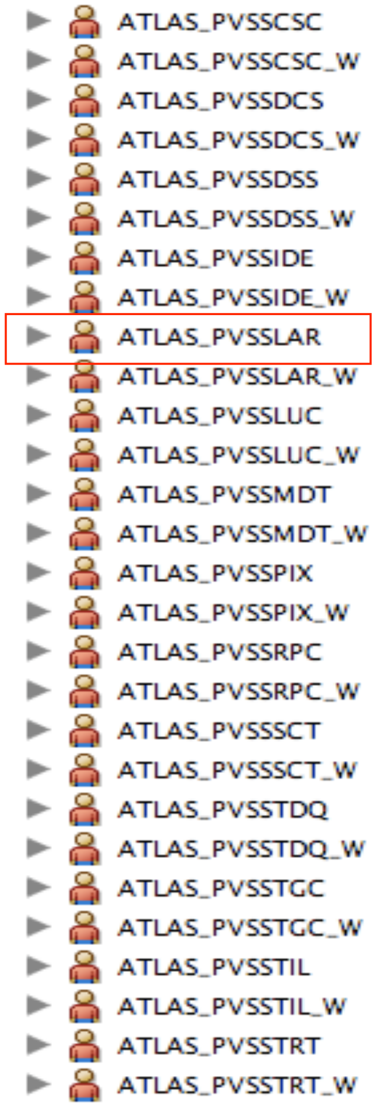
The ATLAS detector

The ATLAS ‘online’ Oracle DB

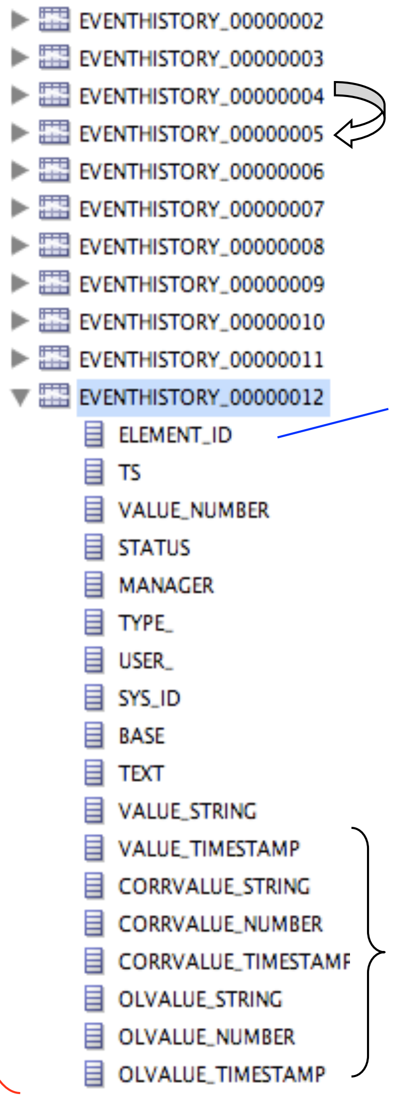
The ATLAS PVSS DB accounts and table desc.



- A database schema per subdetector (as total 18)



ATLAS_PVSSCSC
ATLAS_PVSSCSC_W
ATLAS_PVSSDCS
ATLAS_PVSSDCS_W
ATLAS_PVSSDSS
ATLAS_PVSSDSS_W
ATLAS_PVSSIDE
ATLAS_PVSSIDE_W
ATLAS_PVSSLAR
ATLAS_PVSSLAR_W
ATLAS_PVSSLUC
ATLAS_PVSSLUC_W
ATLAS_PVSSMDT
ATLAS_PVSSMDT_W
ATLAS_PVSSPIX
ATLAS_PVSSPIX_W
ATLAS_PVSSRPC
ATLAS_PVSSRPC_W
ATLAS_PVSSSCT
ATLAS_PVSSSCT_W
ATLAS_PVSSTDQ
ATLAS_PVSSTDQ_W
ATLAS_PVSSTGC
ATLAS_PVSSTGC_W
ATLAS_PVSSTIL
ATLAS_PVSSTIL_W
ATLAS_PVSSTRT
ATLAS_PVSSTRT_W



EVENTHISTORY_00000002
EVENTHISTORY_00000003
EVENTHISTORY_00000004
EVENTHISTORY_00000005
EVENTHISTORY_00000006
EVENTHISTORY_00000007
EVENTHISTORY_00000008
EVENTHISTORY_00000009
EVENTHISTORY_00000010
EVENTHISTORY_00000011
EVENTHISTORY_00000012
ELEMENT_ID
TS
VALUE_NUMBER
STATUS
MANAGER
TYPE_
USER_
SYS_ID
BASE
TEXT
VALUE_STRING
VALUE_TIMESTAMP
CORRVALUE_STRING
CORRVALUE_NUMBER
CORRVALUE_TIMESTAMP
OLVALUE_STRING
OLVALUE_NUMBER
OLVALUE_TIMESTAMP

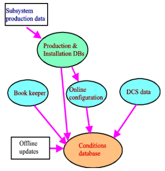
Table is 'switched' when it reaches
a certain time limit (months) and a view
is updated to keep them together
for the application to access the data
(the EVENTHISTORY view)

Data point elements

Not used from ATLAS,
get NULL values, thus
do not take occupy space

The row length
is in the range
55-60 bytes

Sliding window for the PVSS archive on the ATONR



- An idea of keeping only the data of the most recent 12 months on the ATONR database (sliding window) popped up naturally.

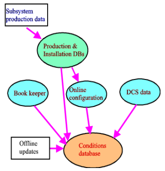
The reasons are:

- the operators in the ATLAS control room do NOT need to look furhter than 12 months in the past.

- the complete archive is already on the ATLAS ‘offline’ database

- the ‘online’ DB is vital for the datataking and is wise to be kept smaller in case of a need of recovery operation.

The PVSS data (all tables and index segments) of 12 months occupy less than 4 TB



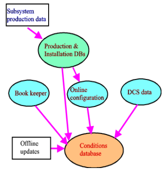
A PVSS query performance problem

- For queries that are interested in data of the most recent hours, often get non-optimal execution plan and consume a lot of resources.

  e.g. For the ‘WHERE modiftime > SYSDATE - 1/2’ the Optimizer considers that there are only few rows relevant to that condition even if the statistics are very recent (computed from the last night).

  In reality, for a ½ day in several different PVSS schemas we could get tens or hundreds of thousands rows. With the wrong statistics Oracle produces non-optimal execution plans.

  A real PVSS case is where more than two indices exist on the EVENTHISTORY table and Oracle decides for the inappropriate one or when a join of two tables is needed Oracle chooses NESTED LOOPs with an index range scan instead of a single HASH JOIN. That leads to much more buffer reads (respectively IO and CPU)

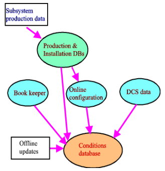


A fix to the PVSS query performance issue

- The fix of the issue described in the previous slide is explicit regular update of the timestamp column max value in the Oracle's data dictionary.

  A PLSQL procedure and a daily scheduler job are changing (advancing with 24 hours) the TS column max value of the latest EVENTHISTORY table for total 18 PVSS schemes.

```
…
hv_date := SYSDATE + m_increase_by;
datevals := DBMS_STATS.DATEARRAY(lv_date, hv_date);
DBMS_STATS.PREPARE_COLUMN_VALUES(new_stat_rec, datevals);
DBMS_STATS.SET_COLUMN_STATS
( UPPER(m_owner), UPPER(m_table), UPPER(m_column),
distcnt => n_distcnt, density => n_density,
nullcnt => n_nullcnt, srec => new_stat_rec,
avgclen => n_avgclen  );
…
```



A fix to the query performance issue (cont.)

Logged output from the scheduler job which was executed on 24-MAY-2013 at 08:00:01 a.m. for DB schema ATLAS_PVSSCSC

Before the change:

ATLAS_PVSSCSC.EVENTHISTORY_00000007 table

⇒ the TS column

⇒ lowest_value: 20-JAN-2013 11:36:37

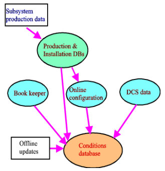
the highest_value: **24-MAY-2013 08:00:01**

After the change:

ATLAS_PVSSCSC.EVENTHISTORY_00000007 table

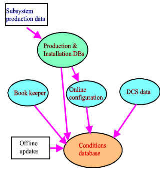
⇒ the TS column lowest_value: 20-JAN-2013 11:36:37

⇒ the NEW highest_value**: 25-MAY-2013 08:00:01**



Case study 3

NICOS

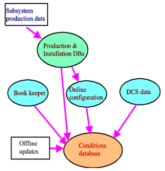


NICOS (recently deployed system)

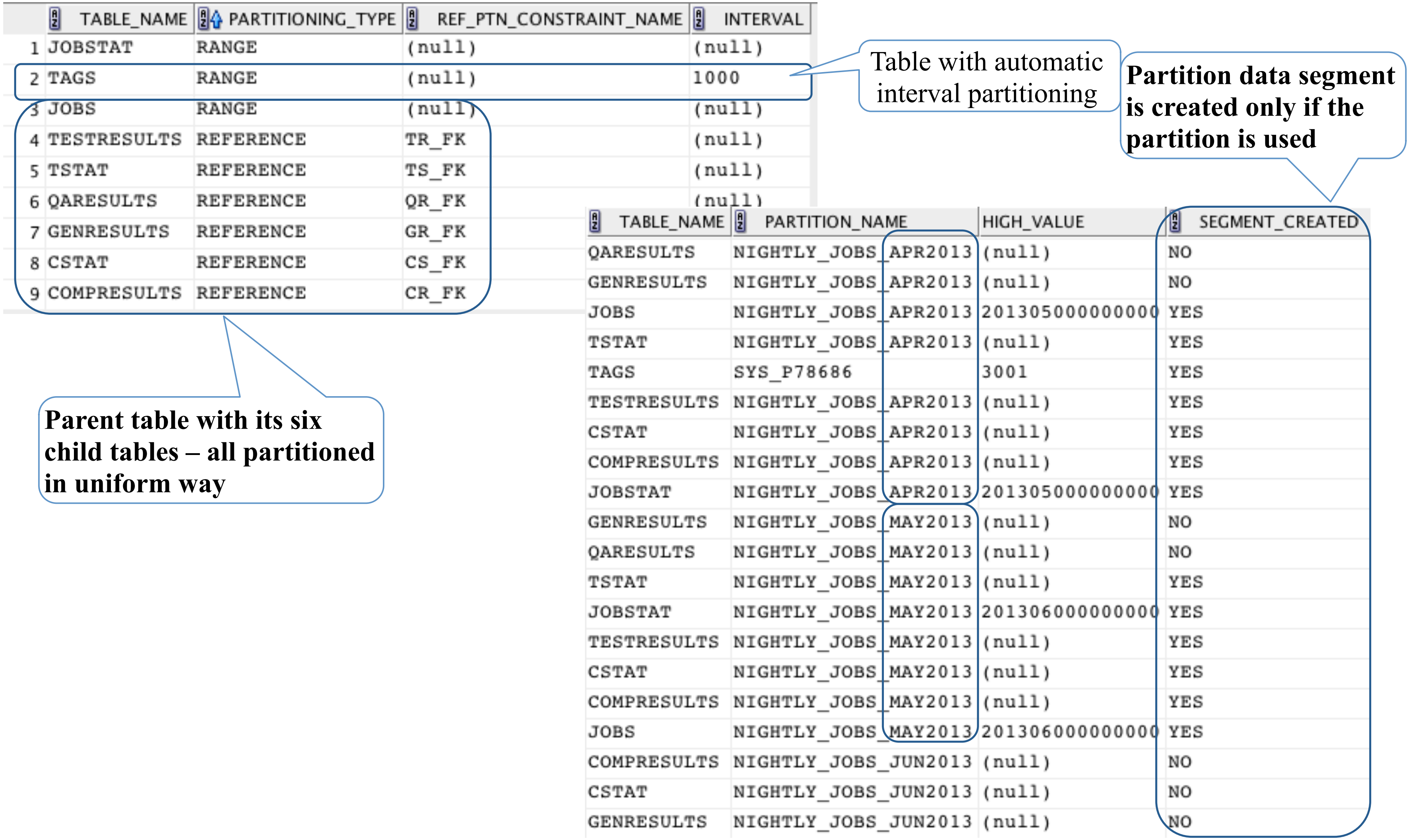
- The new ATLAS Nightly build system (NICOS) database schema hosts the package tags, nightly jobs timestamps, status information and other important parameters.
- The data volume is not huge (few 10s GB/year), but the application and the availability of the data in the database are considered critical
- The retention of the fact data is agreed to be 12 months.

Having the above requirements was choicen an approach of using **reference partitioning** (first time in ATLAS) for the child tables that host the nightly job attribues.

The parent table is configured to use **range partitioning**



Reference partitions organization in NICOS



	TABLE_NAME	PARTITIONING_TYPE	REF_PTN_CONSTRAINT_NAME	INTERVAL
1	JOBSTAT	RANGE	(null)	(null)
2	TAGS	RANGE	(null)	1000
3	JOBS	RANGE	(null)	(null)
4	TESTRESULTS	REFERENCE	TR_FK	(null)
5	TSTAT	REFERENCE	TS_FK	(null)
6	QARESULTS	REFERENCE	QR_FK	(null)
7	GENRESULTS	REFERENCE	GR_FK	
8	CSTAT	REFERENCE	CS_FK	
9	COMPRESULTS	REFERENCE	CR_FK	

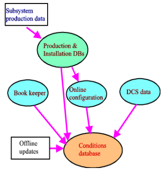
Table with automatic interval partitioning

Parent table with its six child tables – all partitioned in uniform way

Partition data segment is created only if the partition is used

TABLE_NAME	PARTITION_NAME	HIGH_VALUE	SEGMENT_CREATED
QARESULTS	NIGHTLY_JOBS_APR2013	(null)	NO
GENRESULTS	NIGHTLY_JOBS_APR2013	(null)	NO
JOBS	NIGHTLY_JOBS_APR2013	201305000000000	YES
TSTAT	NIGHTLY_JOBS_APR2013	(null)	YES
TAGS	SYS_P78686	3001	YES
TESTRESULTS	NIGHTLY_JOBS_APR2013	(null)	YES
CSTAT	NIGHTLY_JOBS_APR2013	(null)	YES
COMPRESULTS	NIGHTLY_JOBS_APR2013	(null)	YES
JOBSTAT	NIGHTLY_JOBS_APR2013	201305000000000	YES
GENRESULTS	NIGHTLY_JOBS_MAY2013	(null)	NO
QARESULTS	NIGHTLY_JOBS_MAY2013	(null)	NO
TSTAT	NIGHTLY_JOBS_MAY2013	(null)	YES
JOBSTAT	NIGHTLY_JOBS_MAY2013	201306000000000	YES
TESTRESULTS	NIGHTLY_JOBS_MAY2013	(null)	YES
CSTAT	NIGHTLY_JOBS_MAY2013	(null)	YES
COMPRESULTS	NIGHTLY_JOBS_MAY2013	(null)	YES
JOBS	NIGHTLY_JOBS_MAY2013	201306000000000	YES
COMPRESULTS	NIGHTLY_JOBS_JUN2013	(null)	NO
CSTAT	NIGHTLY_JOBS_JUN2013	(null)	NO
GENRESULTS	NIGHTLY_JOBS_JUN2013	(null)	NO

Reference partitioning : pros and cons



- Pros:

  => Tables are partitioned in uniform way

  => On event of partition creation in the parent table Oracle automatically creates relevant partitions into the child tables

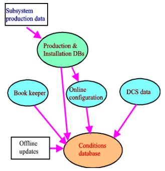
  => Provides flexibility for maintaining any data sliding window (indexes are partitoned as well)

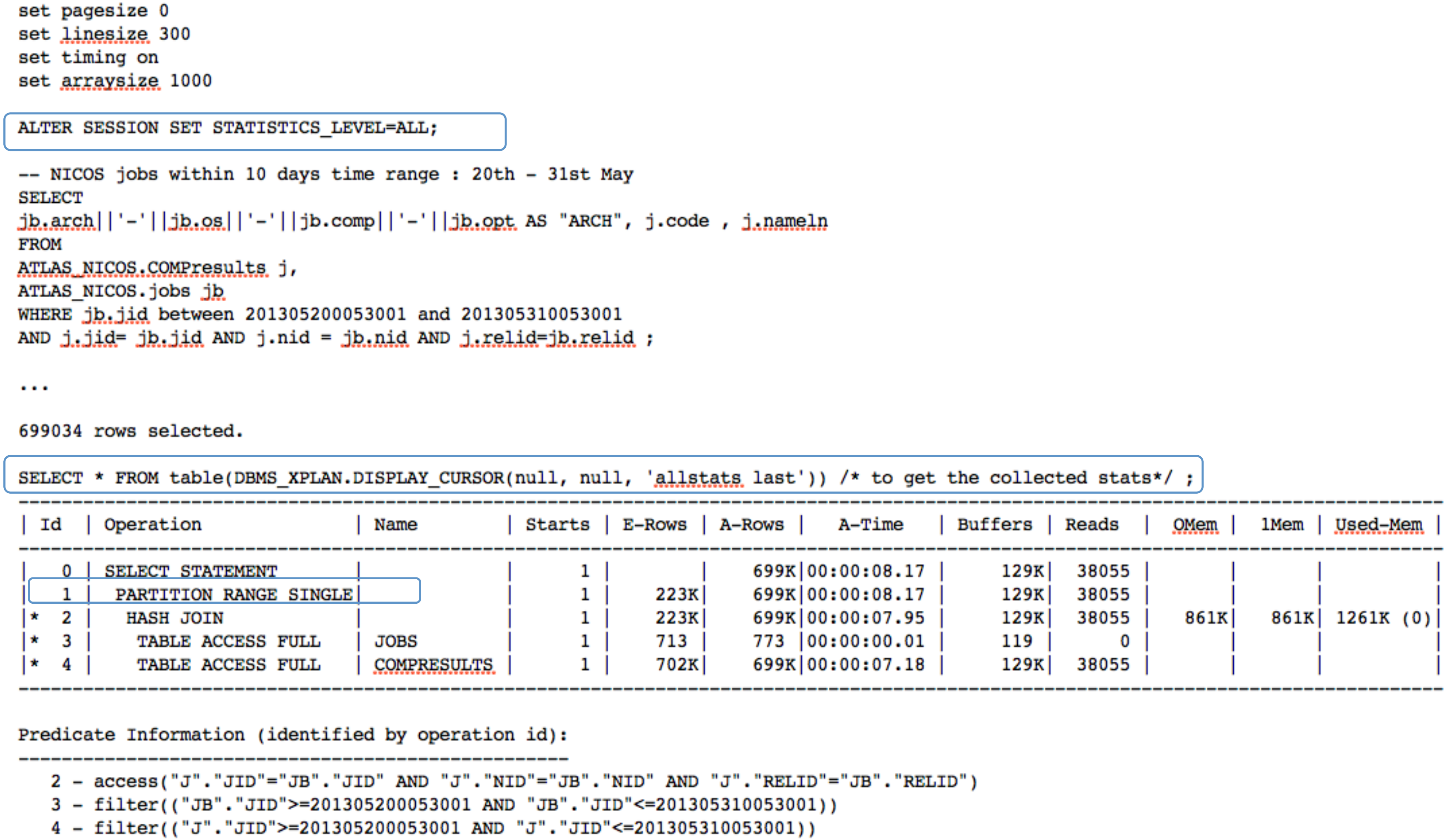
  => Partition pruning takes place when retrieving data

- Cons

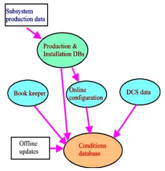
  => There is an inconvenience that Oracle does not yet provide an automatic INTERVAL partitioning on the parent table combined with REFERENCE partitioning for the child tables

Query with partition pruning taking place



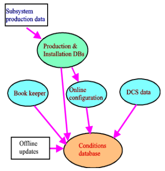


```
set pagesize 0
set linesize 300
set timing on
set arraysize 1000

ALTER SESSION SET STATISTICS_LEVEL=ALL;

-- NICOS jobs within 10 days time range : 20th - 31st May
SELECT
jb.arch||'-'||jb.os||'-'||jb.comp||'-'||jb.opt AS "ARCH", j.code , j.nameln
FROM
ATLAS_NICOS.COMPresults j,
ATLAS_NICOS.jobs jb
WHERE jb.jid between 201305200053001 and 201305310053001
AND j.jid= jb.jid AND j.nid = jb.nid AND j.relid=jb.relid ;

...

699034 rows selected.

SELECT * FROM table(DBMS_XPLAN.DISPLAY_CURSOR(null, null, 'allstats last')) /* to get the collected stats*/ ;
-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation               | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  | OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |               |      1 |        |   699K|00:00:08.17 |    129K|  38055 |      |       |          |
|   1 |  PARTITION RANGE SINGLE|               |      1 |   223K|   699K|00:00:08.17 |    129K|  38055 |      |       |          |
|*  2 |   HASH JOIN             |               |      1 |   223K|   699K|00:00:07.95 |    129K|  38055 |  861K|   861K| 1261K (0)|
|*  3 |    TABLE ACCESS FULL    | JOBS          |      1 |    713 |   773  |00:00:00.01 |    119  |      0 |      |       |          |
|*  4 |    TABLE ACCESS FULL    | COMPRESULTS   |      1 |   702K|   699K|00:00:07.18 |    129K|  38055 |      |       |          |
-------------------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("J"."JID"="JB"."JID" AND "J"."NID"="JB"."NID" AND "J"."RELID"="JB"."RELID")
   3 - filter(("JB"."JID">=201305200053001 AND "JB"."JID"<=201305310053001))
   4 - filter(("J"."JID">=201305200053001 AND "J"."JID"<=201305310053001))
```



Case study 4

Rucio system



New development: Rucio system

Rucio is the new ATLAS file management system - successor of DQ2

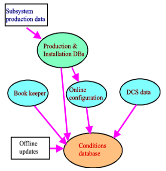
Challenges:

=> It is a bookkeeping system which represents the momentum state and placement of production and user data files over the ATLAS grid sites.

=> Because of the dynamic placement of files and datasets, large fraction of the the data is transient and has to be handled properly.

=> Fast and free pattern searches based on file and dataset names have to be supported.

=> Each Rucio action on the grid has to be recorded for traceability and accounting purposes



Ongoing studies: Rucio system

Ongoing studies on different partitioning techniques

Because of the application logic the most natural choice is **List partitioning** (each partition is bound to an explicit value or list of values)

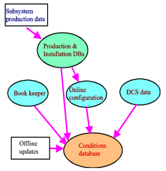
Currently under evaluation is :

=> **List** partitioning plus **list** sub-partitioning

=> **List** partitioning based on a virtual column (concatenation on values of two columns)

As usual a challenge comes with the decision of the right index strategy – global, local (partitioned as the table segments), compression, maintenance, access patterns … etc

LIST partition: an operational challenge



List partitioning seems appropriate, but it is rather static as partitions must be manually pre-created for each new partition key value

This is an operational burden as noone can predict the needed values for the lifetime of the Rucio system.

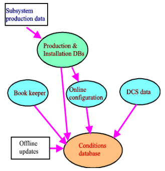
<u>Home made solution:</u>

Automatic partition creation in all relevant tables whenever a new partition key value is inserted into a dimensional table.

“After insert” row level trigger gets fired and executes a PLSQL procedure responsible for ALTER TABLE … ADD PARTITION …

Logic for handling ORA-00054 “resource_busy” error is in place

Each partition creation action is logged into a dedicated logging table

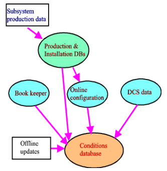


Handy DB monitoring metrics

- In case you have access to the Oracle performance views , e.g. GV$SYSMETRIC you can define your own alert system for the DB instance where your application runs.
- Defining thresholds for sending an alert e-mail is up to you knowing the specifics of your applicaiton

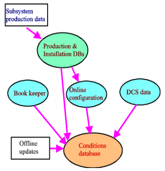
Metric_id = 2003, User Transaction Per Sec
Metric_id = 2030, Logical Reads Per Sec
Metric_id = 2106, SQL Service Response Time (in centiseconds)
Metric_id = 2058, Network Traffic Volume Per Sec (Bytes Per Second)
Metric_id = 2018, Logons Per Sec
Metric_id = 2147, Average Active Sessions
Metric_id = 2118, Process Limit %
Metric_id = 2119, Session Limit %
Metric_id = 2143, Session Count

- Those proved to be very useful for reacting on time on database or application misbehaviour.



My key messages

- Think of your application with long term vision and design accordingly
- Keep it simple (I know it is easier said rather than being done ☺)
- Consult with the DBAs from early stage of development throughout deployment and operation. And not only consult, but also put in practice his/her guidelines
- When tuning SQL statements consider the number of block reads per execution as measurement for what you have improved
- Keep the ‘operational live’ data and ‘archived’ in separate data segments
- Instrument your code for better traceability



Thank you!

(the content of this presentation was a small portion of what I have came across in my daily work in ATLAS)

Special thanks to the PhyDB DBAs for the excellent database support and pleasant collaborative work!