LCG Persistency Framework (CORAL, COOL, POOL): Status and Outlook in 2012

R. Trentadue[1], M. Clemencic[2,a], D. Dykstra[3,b], M. Frank[2,a], D. Front[4,c], A. Kalkhof[l] , A. Loth[1], M. Nowak[5,c], A. Salnikov[6,c], A. Valassi[1], M. Wache[7,c]

[1] IT Department, CERN, CH-1211 Geneva 23, Switzerland
[2] PH Department, CERN, CH-1211 Geneva 23, Switzerland
[3] Fermi National Accelerator Laboratory, Batavia, IL 60510, USA
[4] Weizmann Institute of Science, Rehovot 76100, Israel
[5] Brookhaven National Laboratory, Upton, NY 11973, USA
[6] SLAC National Accelerator Laboratory, Menlo Park, CA 94025, USA
[7] Institut für Physik, Universität Mainz, D-55099 Mainz, Germany
[a] The author is a member of the LHCb Collaboration.
[b] The author is a member of the CMS Collaboration.
[c] The author is a member of the ATLAS Collaboration.

E-mail: andrea.valassi@cern.ch

Abstract. The LCG Persistency Framework consists of three software packages (CORAL, COOL and POOL) that address the data access requirements of the LHC experiments in several different areas. The project is the result of the collaboration between the CERN IT Department and the three experiments (ATLAS, CMS and LHCb) that are using some or all of the Persistency Framework components to access their data. POOL is a hybrid technology store for C++ objects, using a mixture of streaming and relational technologies to implement both object persistency and object metadata catalogs and collections. CORAL is an abstraction layer with an SQL-free API for accessing data stored using relational database technologies. COOL provides specific software components and tools for the handling of the time variation and versioning of the experiment conditions data. This presentation reports on the status and outlook in each of the three sub-projects at the time of the CHEP2012 conference, reviewing the usage of each package in the three LHC experiments.

1. Overview

The Large Hadron Collider (LHC), the world's largest and highest-energy particle accelerator, started its operations in September 2008 at CERN. Huge amounts of data are generated by the four experiments installed at different collision points along the LHC ring. The largest data volumes, coming from the "event data" that record the signals left in the detectors by the particles generated in the LHC beam collisions, are generally stored on files. Relational database systems are commonly used instead to store several other types of data, such as the "conditions data" that record the experimental conditions at the time the event data were collected, as well as geometry data and detector configuration data. In three of the experiments, ATLAS, CMS and LHCb, some or all of these types of data are stored and accessed using the software developed

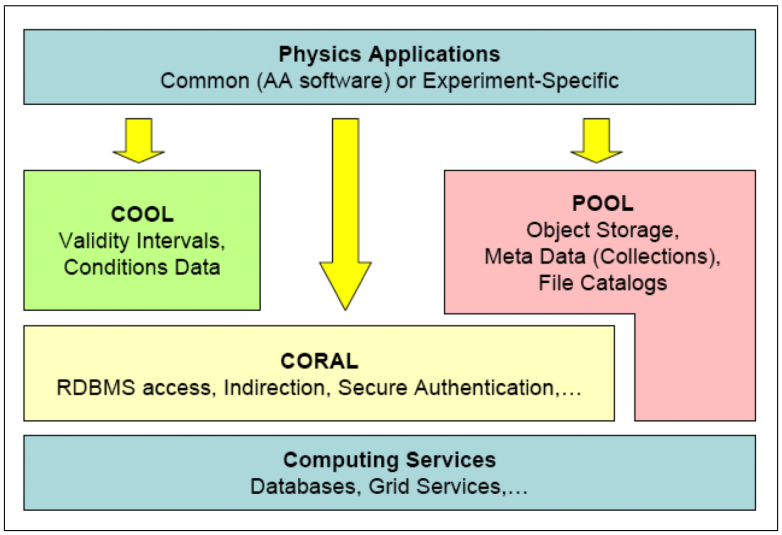


Figure 1. CORAL, COOL and POOL are used by physics applications to implement data persistency using lower level computing services.

by the Persistency Framework (PF), one of the projects set up within the LHC Computing Grid (LCG) to provide common software solutions for the LHC experiments.

The Persistency Framework consists of three packages (CORAL, COOL and POOL) that address the data access requirements of the LHC experiments in different areas. POOL is a generic hybrid store for C++objects, metadata catalogs and collections, using streaming and relational technologies. CORAL is a generic abstraction layer with an SQL-free API for accessing relational databases. COOL provides specific software to handle the time variation and versioning of conditions data. The software has been developed over several years through the well established collaboration of developers from the LHC experiments with a team in the CERN IT department, which has also ensured the overall project coordination. All packages are written in C++, but Python bindings are also provided for CORAL and COOL. As shown in figure 1, all three packages are used directly by physics applications, but CORAL is also used internally by COOL and POOL to access relational databases.

1.1. Summary of progress since CHEP2010

A detailed description of the PF package functionalities, software process and collaborations with other LCG projects can be found in the proceedings of the previous CHEP2010 conference [1] and in the original references cited therein. The goal of this paper is to focus on the evolution and progress in the project as a whole and in each sub-project since that time and to provide an updated summary of its usage in the LHC experiments and of the outlook for the future.

Software development of all three packages continues to follow a well established release process [1], which has been further streamlined and improved in the last two years. Regular production releases are prepared whenever one of the experiments demands it, leading to one release per month on average. As in the past, software libraries and binaries for ATLAS and LHCb are built and installed on shared disks by the SPI team from the CERN PH Department, but two improvements are worth pointing out: first, the full validation of the CORAL and COOL releases has been fully documented [2] and is now also under the responsibility of the SPI team, offloading the workload of the PF core support team in IT that was previously in charge of this task; second, the releases are now installed not only on AFS but also on the CVMFS file system [3], providing easier software distribution to remote sites [4]. The PF software is

supported on many production platforms, including several flavours of Linux (SLC5, SLC6) and MacOSX (10.6 Snow Leopard), using one or more compilers on each O/S (e.g. gcc4.3, gcc4.6, icc11 on SLC5). To improve software quality and speed up the early adoption of new external software versions, automatic builds and tests [5] of CORAL, COOL and POOL are performed every night on all production platforms, as well as on a few test platforms using new compilers and build options (such as gcc4.6 with c++0x, gcc4.7 and clang30 on Linux). Since 2011, the MySQL server and the CORAL server used for the nightly tests have been running on a dedicated cluster maintained by the PF team in the CERN Computer Centre; as in the past, an Oracle server maintained by the Physics Database Team in CERN IT is also used.

The latest software version recently released is the LCG63 configuration, based on ROOT 5.34 and including CORAL 2.3.23 and COOL 2.8.14. It must be noted that POOL is missing in this release series, as well as in the previous one based on ROOT 5.32 (the latest such configuration being LCG62b including CORAL 2.3.22 and COOL 2.8.13). POOL is only included in the still older release series based on ROOT 5.30 (the latest such configuration being LCG61, including CORAL 2.3.20c, COOL 2.8.12c and POOL 2.9.20a), which is still used only by ATLAS as their production software version for the 2012 LHC data taking.

One of the main news for the project as a whole, in fact, has been the decision by LHCb to drop the use of POOL [6] as the primary persistency mechanism for their event data. At the end of 2011, LHCb put in production a new persistency service that uses directly ROOT in a more efficient way than what was done in POOL and is also able to read old POOL files, making the migration simple and smooth. As a consequence, support for POOL is no longer required by LHCb, leaving ATLAS as the only user of this PF component. ATLAS will continue to need support for POOL from the core PF team in IT, including any relevant patches and releases, only for as long as the 2012 production version of the ATLAS software (based on the LCG61 series) is actively used. For subsequent production releases (e.g. those based on LCG63 and ROOT 5.34), an ATLAS-supported custom package derived from POOL will be used and already exists within the ATLAS repository; this will also allow ATLAS to better integrate into their persistency strategy several recent improvements [7] in their use of ROOT I/O. It is therefore expected that, as of 2013, POOL will no longer be maintained and supported within the context of the PF common project, whereas CORAL and COOL will continue to need to be supported as PF common packages used by more than one experiment [8].

Another important change in the PF usage by the experiments during the last two years was the LHCb decision to drop the use of the CORAL LFCReplicaSvc component, which provides database lookup and authentication functionalities using the LFC servers deployed on the Grid. LHCb is in fact moving towards a distributed conditions database infrastructure more heavily based on SQLite files [9], while also investigating the possible use of Frontier: both these options do not involve direct Oracle access from the Grid and as a consequence do not require the functionalities of the CORAL LFCReplicaSvc component, which has thus been dropped as of the LCG63 release.

Another change since CHEP2010 concerns the use of Frontier in ATLAS, which has further expanded: this technology, which was previously used in this experiment only for data access in the Grid and only for conditions data, is now used in ATLAS also for Tier0 processing [10] and also for geometry and trigger data.

Table 1 summarizes the usage of CORAL, COOL and POOL in the LHC experiments; the changes described in this section are highlighted in green (when specific components have been adopted for new use cases) and red (when the usage of some components has been dropped).

Last but not least, progress has also been made in each of CORAL, COOL and POOL to fix specific issues and to provide new functionalities as requested by the experiments or by individual users. The main changes in POOL concern several fixes and enhancements in the collection packages, developed exclusively by ATLAS to be used in ATLAS, which will not

Persistency Framework in the LHC experiments	ATLAS	CMS	LHCb
CORAL (Oracle, SQLite, XML authentication and lookup)	Conditions data (COOL) Geometry data (detector descr.) Trigger configuration data Event collections/tags (POOL)	Conditions data Geometry data (detector descr.) Trigger configuration data	Conditions data (COOL)
CORAL + Frontier (Frontier/Squid)	Conditions, Geometry, Trigger *(R/O access in Grid, Tier0)*	Conditions, Geometry, Trigger *(R/O access in Grid, HLT, Tier0)*	— *(will be tested in 2012)*
CORAL Server (CoralServer/CoralServerProxy)	Conditions, Geometry, Trigger *(R/O access in HLT)*	—	—
CORAL + LFC (LFC authentication and lookup)	—	—	Conditions data *(authentication/lookup in Grid)* *(will be dropped in 2012)*
COOL	Conditions data	—	Conditions data
POOL (ROOT storage service)	Event data Event collections/tags Conditions data (payload)	—	Event data *(dropped in 2011)*
POOL (Collections – ROOT and Relational)	Event collections/tags	—	—

Table 1. Summary of CORAL, COOL and POOL usage in ATLAS, CMS and LHCb.

described in this paper. Within CORAL and COOL, service and operation support continues to be the critical area where most of the work was carried out and most of the progress was obtained, especially for any issues involving the optimization of connectivity and data access performance for Oracle database services. In particular, the two main such achievements, about which more details will be given in sections 2 and 3, are the improvement of CORAL handling of network and database instabilities (also covered in much greater detail by another presentation at this conference [11]) and the validation of COOL query performance on Oracle 11g servers. Another enhancement for CORAL in this area, which however will not be described in detail in this paper, was the reduction of the number of Oracle data dictionary queries in CORAL, resulting in faster data retrieval from the database. Finally, some R&D on the monitoring of CORAL server and proxy components was also carried out [12], although this has not yet resulted in the development of production-quality components.

2. Improvements in CORAL handling of network instabilities

As described in section 1, the CORAL software is effectively the entry point for the LHC experiments to access from their C++ applications the data that they store in Oracle databases. This makes CORAL the ideal place in the software chain to implement in a common way several features optimizing the data access patterns of the LHC experiments to Oracle. In particular, one extremely desirable feature for CORAL would be a functionality enabling client applications to safely handle database and network instabilities, transparently reconnecting when possible and appropriate, and immediately throwing an exception in all other cases.

One implementation of this functionality was already present in CORAL since the very earliest versions of the software in 2006 [13] and had never been reported to cause any issue. As the load on the Oracle database servers and the usage of the CORAL software increased with the gradual ramp-up of LHC operations since the end of 2009, however, several application hangs, crashes and other issues have been reported by all three experiments using CORAL. While the exact details of these incident reports differ from case to case, it was immediately obvious that all these issues had ultimately been triggered by a network or database glitch,

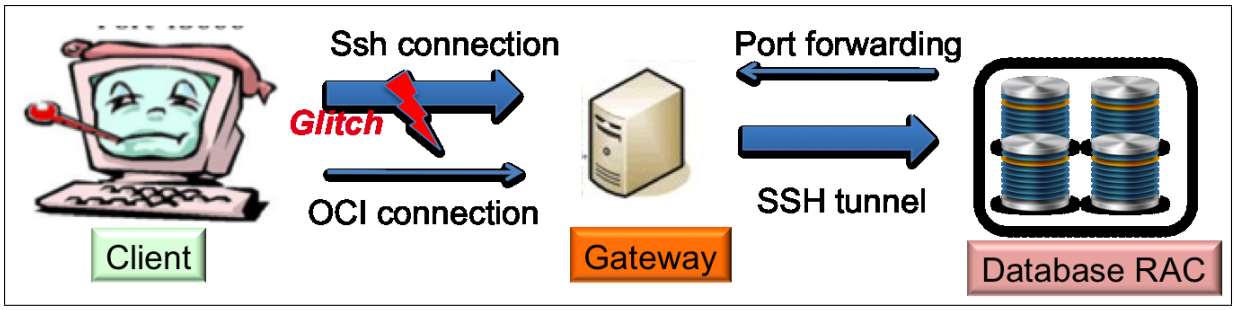


Figure 2. An ssh tunnel is used to simulate a network glitch in the connection to the database.

reported by the Oracle client to CORAL using a well-known error code (most often, ORA-03113). More importantly, as the analysis of the problem progressed, it soon became apparent that in many such situations the damage was caused not only by the network glitch itself, but also by bugs, and other intrinsic limitations of the handling of this external problem, within the old implementation of the 'reconnection' functionality in CORAL.

As a consequence, the improvement and eventual reimplementation of the handling of network and database instabilities has become one of the main areas of work in CORAL since CHEP2010. This is a large task which has been achieved in several phases over time and is now essentially complete. Only a brief summary of the most relevant issues and achievements will be presented in this paper, as all this work is described in far greater details by another presentation at this conference [11]. The following is an overview of the various tasks achieved in this area in different phases, in chronological order.

- The issue most frequently reported by all three experiments, an application hang caused by an infinite loop when CORAL retries over and over to begin a user session over a physical connection that was broken by a network glitch (causing a flood of ORA-24327 error messages), was successfully analyzed, reproduced and fixed by a patch released in CORAL 2.3.13 in December 2010.
- To reproduce this issue, an ad-hoc test suite, based on CORAL python bindings (PyCoral) and using an ssh tunnel to simulate a network glitch, as shown schematically in figure 2, had to be developed: this test, first introduced in CORAL 2.3.13 in December 2010, has then been significantly extended over time to cover the many other situations covered by the new CORAL reconnection mechanism and is now still routinely executed within the CORAL nightly test suite.
- The next major milestone was the analysis and fix of the CORAL crashes reported by some users in situations involving network glitches. It was soon understood that this was due to a set of bugs that affect not only the old CORAL handling of network glitches, but more generally the CORAL management of closed sessions, even when a session is closed as the result of an explicit user request to disconnect. To solve this large family of issues, a major internal reimplementation of all CORAL plugins has been planned and largely implemented, involving for instance the replacement of bare pointers by shared pointers in most C++ classes of the CORAL plugins. Out of the many patches required, those necessary to address these bugs for Oracle (in both single-threaded and multi-threaded use cases) and SQLite (in single-threaded use cases only) have been successfully completed in the CORAL 2.3.16 release in June 2011, while those for the other plugins (Frontier, MySQL and CoralServer) are being added over time and are in some cases still pending.
- Another set of crashes reported by some users during the cleanup phase of their application (e.g. when closing a statement associated to an already deleted session), not necessarily after a network glitch, was eventually understood to be specific to the OracleAccess plugin and

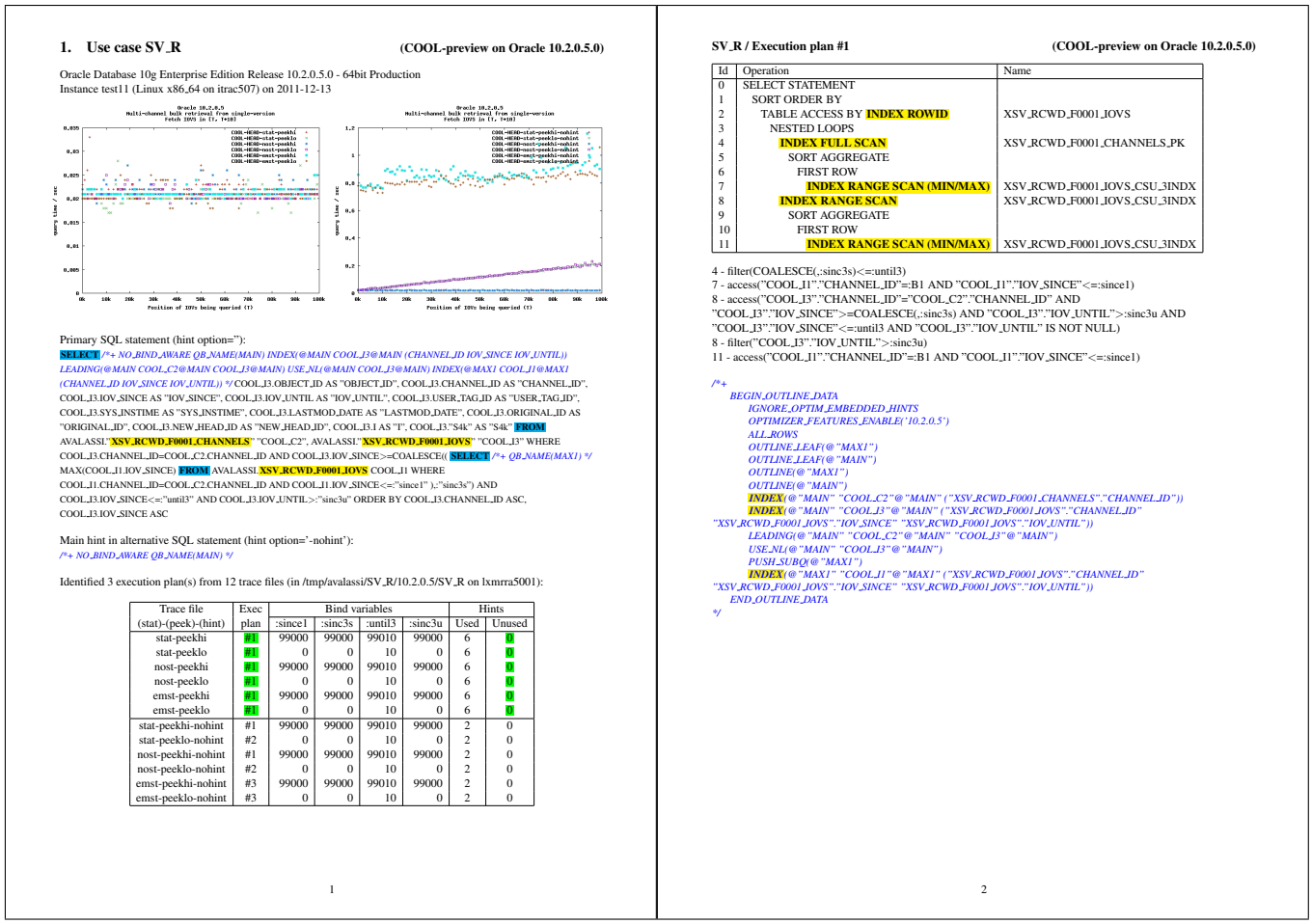
to be caused by the way this uses the Oracle Call Interface (OCI) [14] client structures. The issue is quite similar to the other set of crashes described above, but involves segmentation faults deep inside the Oracle client library, rather than in the CORAL code itself. The relevant fixes, which essentially consist in delaying the release of the OCI structures until they are no longer used by any other OCI structure in CORAL, have been completed in the CORAL 2.3.23 release in June 2012,

- Building on the expertise and tools developed to address all of these preliminary issues, a new strategy for the reconnection functionality in CORAL was designed and implemented. The previous version of this functionality was implemented in a common CORAL component used by all back-ends (Oracle, SQLite, MySQL...) and implied the destruction and re-creation of an instance of the relevant CORAL C++ class describing database sessions. The new strategy is implemented only for the Oracle back-end within the OracleAccess plugin and essentially consists in "refreshing" the existing CORAL session instance with the new OCI data structures obtained when reconnecting to Oracle. The CORAL client actively probes the connection to the server using a relatively inexpensive OCI call to check if it has been lost, every single time that an SQL statement or another command is about to be sent to the server for processing. If the connection has been lost, a reconnection is attempted if possible and appropriate (i.e. if there is no risk of data inconsistency), otherwise an exception is thrown (e.g. if data has been lost because an update had been executed but not yet committed). The reimplementation of the CORAL handling of network glitches has now been completed and was finally released with CORAL 2.3.23 in June 2012.
- Finally, a few issues reported by CORAL users and involving the Oracle Transparent Application Failover (TAF) [14] mechanism have been analysed. In particular, some tests have been performed to understand if TAF could be a useful complementary mechanism to transparently handle network glitches, or conversely if it could be a source of interference with their handling in CORAL. The results of this analysis are however still preliminary and more work will be done in this area to get a better understanding of these issues.

3. COOL query performance validation on Oracle 11g servers

Out of the many backends supported by COOL via CORAL, Oracle is the one that has been used for many years by both ATLAS [15] and LHCb [9] to store the master copies of their conditions databases. In particular, these data are stored in Oracle Real Application Clusters (RAC) at the CERN Tier0 site, which have been running until 2011 version 10g (10.2.0.5) of the Oracle database server software. The plan for an upgrade of these and all other LHC physics databases to the next major Oracle server version 11g was announced in 2011 by the Physics Database Team in CERN IT, who are responsible for the operation of these installations. Following a well established change management policy, all of the applications accessing these databases, including COOL, underwent an extensive period of performance validation using dedicated "integration" services [8] running Oracle 11g, to ensure the absence of any major showstoppers before the migration could be eventually performed at the end of 2011.

After a large optimization effort in 2008 [16], COOL query performance is by now considered to be relatively under control. The two most important issues which in the past were found to lead to poor performance due to execution plan instability, i.e. unreliable statistics and bind variable peeking, are taken care of using SQL hints in the most important COOL queries. The test strategy identified in 2008 continues to be considered useful and sufficient to identify any problems with query performance and scalability: this consists in creating relatively small COOL test tables containing a few thousand "intervals-of-validity" (IOVs) scattered across a range of validity times T, and in measuring the query response time to retrieve COOL IOVs from that table as a function of the validity time T used to look them up. Query performance



1. Use case SV_R **(COOL-preview on Oracle 10.2.0.5.0)**

Oracle Database 10g Enterprise Edition Release 10.2.0.5.0 - 64bit Production
Instance test11 (Linux x86_64 on itrac507) on 2011-12-13

Primary SQL statement (hint option=''):

```
SELECT /*+ NO_BIND_AWARE QB_NAME(MAIN) INDEX(@MAIN COOL_I3@MAIN (CHANNEL_ID IOV_SINCE IOV_UNTIL))
LEADING(@MAIN COOL_C2@MAIN COOL_I3@MAIN) USE_NL(@MAIN COOL_I3@MAIN) INDEX(@MAX1 COOL_I1@MAX1
(CHANNEL_ID IOV_SINCE IOV_UNTIL)) */ COOL_I3.OBJECT_ID AS "OBJECT_ID", COOL_I3.CHANNEL_ID AS "CHANNEL_ID",
COOL_I3.IOV_SINCE AS "IOV_SINCE", COOL_I3.IOV_UNTIL AS "IOV_UNTIL", COOL_I3.USER_TAG_ID AS "USER_TAG_ID",
COOL_I3.SYS_INSTIME AS "SYS_INSTIME", COOL_I3.LASTMOD_DATE AS "LASTMOD_DATE", COOL_I3.ORIGINAL_ID AS
"ORIGINAL_ID", COOL_I3.NEW_HEAD_ID AS "NEW_HEAD_ID", COOL_I3.I AS "I", COOL_I3."S4k" AS "S4k" FROM
AVALASSI."XSV_RCWD_F0001_CHANNELS" "COOL_C2", AVALASSI."XSV_RCWD_F0001_IOVS" "COOL_I3" WHERE
COOL_I3.CHANNEL_ID=COOL_C2.CHANNEL_ID AND COOL_I3.IOV_SINCE>=COALESCE(( SELECT /*+ QB_NAME(MAX1) */
MAX(COOL_I1.IOV_SINCE) FROM AVALASSI.XSV_RCWD_F0001_IOVS COOL_I1 WHERE
COOL_I1.CHANNEL_ID=COOL_C2.CHANNEL_ID AND COOL_I1.IOV_SINCE<=:"since1" ),:"sinc3s") AND
COOL_I3.IOV_SINCE<=:"until3" AND COOL_I3.IOV_UNTIL>:"sinc3u" ORDER BY COOL_I3.CHANNEL_ID ASC,
COOL_I3.IOV_SINCE ASC
```

Main hint in alternative SQL statement (hint option='-nohint'):

```
/*+ NO_BIND_AWARE QB_NAME(MAIN) */
```

Identified 3 execution plan(s) from 12 trace files (in /tmp/avalassi/SV_R/10.2.0.5/SV_R on lxmrra5001):

Trace file (stat)-(peek)-(hint)	Exec plan	Bind variables :since1	:sinc3s	:until3	:sinc3u	Hints Used	Unused
stat-peekhi	#1	99000	99000	99010	99000	6	0
stat-peeklo	#1	0	0	10	0	6	0
nost-peekhi	#1	99000	99000	99010	99000	6	0
nost-peeklo	#1	0	0	10	0	6	0
emst-peekhi	#1	99000	99000	99010	99000	6	0
emst-peeklo	#1	0	0	10	0	6	0
stat-peekhi-nohint	#1	99000	99000	99010	99000	2	0
stat-peeklo-nohint	#2	0	0	10	0	2	0
nost-peekhi-nohint	#1	99000	99000	99010	99000	2	0
nost-peeklo-nohint	#2	0	0	10	0	2	0
emst-peekhi-nohint	#3	99000	99000	99010	99000	2	0
emst-peeklo-nohint	#3	0	0	10	0	2	0

1

SV_R / Execution plan #1 **(COOL-preview on Oracle 10.2.0.5.0)**

Id	Operation	Name
0	SELECT STATEMENT	
1	SORT ORDER BY	
2	TABLE ACCESS BY INDEX ROWID	XSV_RCWD_F0001_IOVS
3	NESTED LOOPS	
4	INDEX FULL SCAN	XSV_RCWD_F0001_CHANNELS_PK
5	SORT AGGREGATE	
6	FIRST ROW	
7	INDEX RANGE SCAN (MIN/MAX)	XSV_RCWD_F0001_IOVS_CSU_3INDX
8	INDEX RANGE SCAN	XSV_RCWD_F0001_IOVS_CSU_3INDX
9	SORT AGGREGATE	
10	FIRST ROW	
11	INDEX RANGE SCAN (MIN/MAX)	XSV_RCWD_F0001_IOVS_CSU_3INDX

```
4 - filter(COALESCE(,:sinc3s)<=:until3)
7 - access("COOL_I1"."CHANNEL_ID"=:B1 AND "COOL_I1"."IOV_SINCE"<=:since1)
8 - access("COOL_I3"."CHANNEL_ID"="COOL_C2"."CHANNEL_ID" AND
"COOL_I3"."IOV_SINCE">=COALESCE(,:sinc3s) AND "COOL_I3"."IOV_UNTIL">:sinc3u AND
"COOL_I3"."IOV_SINCE"<=:until3 AND "COOL_I3"."IOV_UNTIL" IS NOT NULL)
8 - filter("COOL_I3"."IOV_UNTIL">:sinc3u)
11 - access("COOL_I1"."CHANNEL_ID"=:B1 AND "COOL_I1"."IOV_SINCE"<=:since1)

/*+
    BEGIN_OUTLINE_DATA
      IGNORE_OPTIM_EMBEDDED_HINTS
      OPTIMIZER_FEATURES_ENABLE('10.2.0.5')
      ALL_ROWS
      OUTLINE_LEAF(@"MAX1")
      OUTLINE_LEAF(@"MAIN")
      OUTLINE(@"MAX1")
      OUTLINE(@"MAIN")
      INDEX(@"MAIN" "COOL_C2"@"MAIN" ("XSV_RCWD_F0001_CHANNELS"."CHANNEL_ID"))
      INDEX(@"MAIN" "COOL_I3"@"MAIN" ("XSV_RCWD_F0001_IOVS"."CHANNEL_ID"
"XSV_RCWD_F0001_IOVS"."IOV_SINCE" "XSV_RCWD_F0001_IOVS"."IOV_UNTIL"))
      LEADING(@"MAIN" "COOL_C2"@"MAIN" "COOL_I3"@"MAIN")
      USE_NL(@"MAIN" "COOL_I3"@"MAIN")
      PUSH_SUBQ(@"MAX1")
      INDEX(@"MAX1" "COOL_I1"@"MAX1" ("XSV_RCWD_F0001_IOVS"."CHANNEL_ID"
"XSV_RCWD_F0001_IOVS"."IOV_SINCE" "XSV_RCWD_F0001_IOVS"."IOV_UNTIL"))
    END_OUTLINE_DATA
*/
```

2

Figure 3. COOL performance report for SV queries against an Oracle 10.2.0.5 server.

and scalability are good if the resulting plots show that query response time is flat and does not increase as a function of the validity time T. As explained in Ref. [16], two sets of curves are generally considered, with and without SQL hints: when using hints, it is expected that all curves should be flat (indicating that hints stabilize the Oracle execution plan and the indexes in the schema are optimally used), while some of the curves without hints may exhibit an increase in the query response time (indicating that an alternative sub-optimal Oracle execution plan is used, for instance one involving a full index scan or a full table scan). In particular, each set contains six curves, representing all combinations with low and high values of peeked bind variables, and with good, bad or missing statistics.

One major result achieved during the validation of COOL query performance in 2011 is that this procedure has now been fully automated. With a few clicks, it is now possible to produce an extremely detailed report of COOL query performance for any COOL software version and against any chosen Oracle database, covering nine of the most important data retrieval use cases in COOL, e.g. both "single-version" (SV) and "multi-version" (MV) data retrieval for several data types and versioning/tagging strategies. The first two pages of the performance report for a typical SV use case against an Oracle 10.2.0.5 server are shown in figure 3: as described above, the two plots on the first page show that performance for this version of the Oracle server software was adequate with hints (left); the SQL query executed and its execution plans for each of the twelve curves (as obtained from server-side trace files) are also displayed and summarized, confirming at a much more detailed level that strictly the same execution plan is used for all of

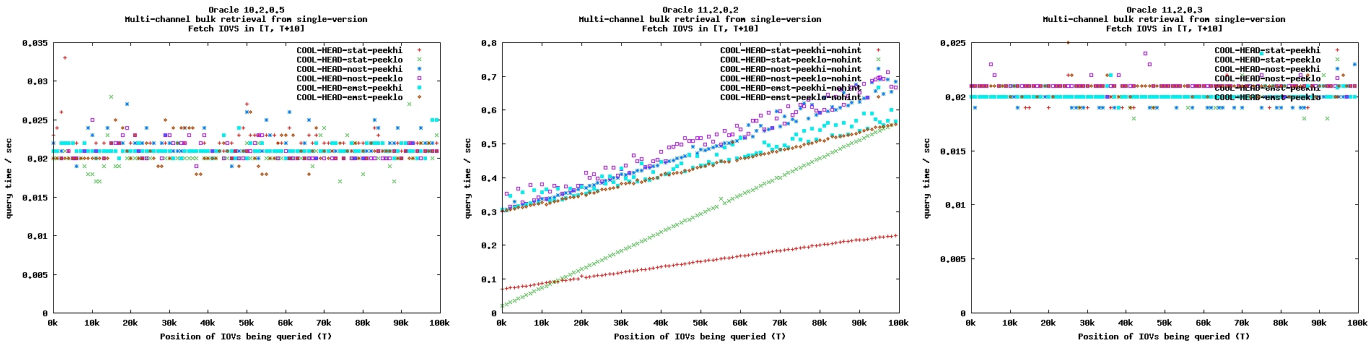


Figure 4. Performance plots for COOL SV queries with hints, for 10.2.0.5 (left), 11.2.0.2 (centre) and 11.2.0.3 (right) Oracle server versions.

the six cases when hints are present.

The main reason why a fully automated tool to produce a COOL performance report was prepared in 2011 is that the initial results of COOL query validation on Oracle 11g servers indicated a problem, so large that it could be a potential show-stopper for the migration. The preliminary results of this issue seemed to indicate that the Oracle 11g Cost-Based Optimizer (CBO) was leading to worse performance for COOL queries than the 10g CBO; at some point, when the problem had not yet been completely understood, the workaround of using a hint in COOL to force the use of the 10g CBO within the 11g server had even been considered and successfully tested. Eventually, the problem was understood as being due to a bug in the version 11.2.0.2 of the Oracle 11g software, which had been used for the initial tests, fixed in the next version 11.2.0.3 of the software. The effects of this bug and of its fix can be seen in figure 4, where the performance plots for COOL SV queries with hints are shown for 10.2.0.5 (left), 11.2.0.2 (centre) and 11.2.0.3 (right) Oracle server versions: it is immediately obvious that queries are too slow and not scalable for Oracle 11.2.0.2 (query response times are high and increasing), even when hints are used. Luckily, figure 4 also shows that performance is again good and scalable against Oracle 11.2.0.3. To be 100% sure that the problems observed were due to a limitation of 11.2.0.2, this issue was actually analysed in much more detail, by installing a test 11.2.0.2 database and modifying it with a few software patches coming from the 11.2.0.3 patch set, until one particular documented bug (absent in 11.2.0.1, introduced in 11.2.0.2 and fixed in 11.2.0.3) was identified with certainty as being the cause of the observed performance degradation.

The major upgrade of the database server software from Oracle 10g (10.2.0.5) to Oracle 11g (11.2.0.3) was at last successfully implemented during the winter 2011-2012 shutdown [8]. The services are now running smoothly and many new useful functionalities that have become available thanks to the upgrade to Oracle 11g (e.g. in the areas of data replication, recovery, caching and partitioning) are also being evaluated [15]. The tool to produce COOL performance reports will certainly remain useful in the future to validate any new upgrades in the COOL or Oracle software versions. In particular, one unexpected lesson that has been learnt from this story is that serious performance regressions may be expected not only in major Oracle server upgrades (e.g. 10g to 11g), but also in minor patch set upgrades (e.g. 11.2.0.1 to 11.2.0.2), hance all such upgrades should also be carefully tested.

4. Conclusion

CORAL, COOL and POOL have been essentially ingredients in the data storage and access stack of the ATLAS, CMS and LHCb experiments at CERN for many years and have been used for

LHC data taking since 2008. POOL has recently been dropped by LHCb and the responsibility for its maintenance is being moved to ATLAS, which is now its only user. COOL and especially CORAL, conversely, remain very active projects which require a large development and support effort, mainly coming from the core Persistency Framework team in CERN IT. Most of the activities concern the application-level optimization of connectivity and query performance for the Oracle relational database technology. Two major achievements in this area, in particular, concern major improvements in the CORAL handling of network and database instabilities and the validation of COOL query performance against Oracle 11g servers.

References

[1] A. Valassi et al., *LCG Persistency Framework (CORAL, COOL, POOL): Status and Outlook*, CHEP 2010, Taipei, http://iopscience.iop.org/1742-6596/331/4/042043

[2] A. Valassi, *Persistency Framework Software Release Process*, PF twiki documentation, https://twiki.cern.ch/twiki/bin/view/Persistency/PersistencyReleaseProcess

[3] J. Blomer et al., *Status and Future Perspectives of CernVM-FS*, CHEP 2012, NY, http://indico.cern.ch/contributionDisplay.py?contribId=93&sessionId=6&confId=149557

[4] A. De Salvo et al., *Software installation and condition data distribution via CernVM FileSystem in ATLAS*, CHEP 2012, NY, http://indico.cern.ch/contributionDisplay.py?contribId=349&sessionId=8&confId=149557

[5] V. Diez Gonzalez et al., *The LCG/AA integration build system*, CHEP 2012, NY, http://indico.cern.ch/contributionDisplay.py?contribId=125&sessionId=8&confId=149557

[6] T. Bell et al., *Report of the WLCG Technology Evolution Groups in Data and Storage Management*, April 2012, https://twiki.cern.ch/twiki/bin/view/LCG/ReportDataStorageTEG

[7] W. Bhimji et al., *The ATLAS ROOT-based data formats: recent improvements and performance measurements*, CHEP2012, NY, http://indico.cern.ch/contributionDisplay.py?contribId=378&sessionId=3&confId=149557

[8] D. Barberis et al., *Report of the WLCG Database Technical Evolution Group*, March 2012, https://twiki.cern.ch/twiki/bin/view/LCG/WLCGTEGDatabase

[9] I. Shapoval et al., *LHCb Conditions Database Operation Assistance Systems*, CHEP2012, NY, http://indico.cern.ch/contributionDisplay.py?contribId=143&sessionId=8&confId=149557

[10] A. Dewhurst et al., *Evolution of grid-wide access to database resident information in ATLAS using Frontier*, CHEP2012, NY, https://indico.cern.ch/contributionDisplay.py?contribId=400&sessionId=6&confId=149557

[11] R. Trentadue et al., *Handling of network and database instabilities in CORAL*, CHEP2012, NY, https://indico.cern.ch/contributionDisplay.py?contribId=102&sessionId=8&confId=149557

[12] A. Loth, *CORAL monitoring*, EGI Community Forum 2012, Munich, https://indico.egi.eu/indico/contributionDisplay.py?contribId=16&confId=679

[13] I. Papadopoulos et al., *CORAL, a software system for vendor-neutral access to relational databases*, CHEP2006, Mumbai, http://indico.cern.ch/contributionDisplay.py?contribId=329&sessionId=4&confId=048

[14] Oracle Corporation, *Oracle® Call Interface Programmers' Guide, 11g Release 2 (11.2)*, http://docs.oracle.com/cd/E11882_01/appdev.112/e10646.pdf

[15] G. Dimitrov et al., *The ATLAS database application enhancements using Oracle 11g*, CHEP2012, NY, https://indico.cern.ch/contributionDisplay.py?contribId=567&sessionId=8&confId=149557

[16] A. Valassi et al., *COOL, LCG conditions database for the LHC experiments* NSS2008, Dresden, http://cdsweb.cern.ch/record/1142723

Acknowledgements

We are grateful to the users of the CORAL, COOL and POOL software in the LHC experiments for their continuous feedback and suggestions for its improvement. We thank the SPI team for maintaining the development infrastructure and external software dependencies for the Persistency Framework. We are also grateful to the ROOT team for their help and suggestions. Finally, we warmly thank our colleagues from the Physics Database Team in CERN IT, together with the DBAs in the LHC experiments, for assisting us in understanding the subtleties of the Oracle database servers they operate.