Report on the ATLAS TDAQ Large Scale Tests 2006

27 April 2007

Contributors:
Doris Burckhart, Sarah Demers, Haleh Hadavand, Sergei Kolos, Joerg Stelzer, Serge Sushkov, Lourenco Vaz, Hans Von Der Schmitt, Haimo Zobernig, scholtes.
TWiki to PDF Converter TWiki2pdf Version 2.3.1 by Steve Lloyd.

Contents

Chapter 1

Abstract

This document provides a detailed report on the DAQ/HLT integrated Large Scale and Performance Tests which were performed in November/December 2006 on the CERN IT LXBATCH cluster. DAQ/HLT System Software tests as well as component tests were run on a PC farm size of up to 950 nodes. The integrated system under test included the Infrastructure Software for Controls and Communication, Level 2 trigger software and Event Filter software including algorithms, with emphasis on HLT and Online Databases. Results are presented, compared to previous tests where applicable and suggestions for the further development and future tests are discussed.

References

Ref1	Report of LST 2005 - ATL-D-TR-0003
Ref2	...

Chapter 2

The Large Scale Tests 2006

2.1 Scope, Aims and Testbed description of LST 2006

The ATLAS Data Acquisition (DAQ) and High-Level Triggers (HLT) system will be comprised initially of ∼2000 or more PC nodes which take part in the control, event readout, second level trigger and event filter operations. A first series of tests of the integrated DAQ/HLT system was performed successfully in summer 2005 at CERN on a farm size of up to 700 PC dual processor nodes over a period of 5 weeks. The need for further large-scale tests was identified and another series of tests is planned. The Large Scale Tests (LST) 2006 took place in the 4th quarter of 2006, starting at the end of October. A period of four weeks with a cluster increasing in steps from 400 to 800 and finally 1200 nodes has been made available.

The tests were done at CERN at the LXBATCH facility. The CERN IT support team delivered to us the operating system management and support of the farm.

The Atlas DAQ/HLT infrastructure software, the Level2 and EF algorithms and the Online Databases will be under test. Sub-systems will run tests exercising their system in order to verify its functionality and to optimise parameters and the layout. The integrated software system will be run as complete as it will be possible at the time of the tests.

The complete Trigger and Data Flow system in ATLAS is foreseen to involve about 2000 PCs. The planned availability of about 1200 PCs for LST 2006 allow constructing partitions with systems up to about 60% of the full ATLAS scale. The tests include the online database access by the HLT nodes for configuration of the Trigger from the Trigger configuration database, the geometry of the detector, and the read access for the configuration of the RODs.

2.1.1 Software releases

Offline-12.0.3-LST-1, initially TDAQ-01-06-01 and later TDAQ-01-06-02 , and the HLT release HLT2.0.3-LST-1 were used for the Large Scale Tests. The offline release was derived from 12.0.3 which became available in October. Several patches were applied during the LST and were entered into the subsequent offline releases.

2.1.2 Aim of the DAQ infrastructure tests

Infrastructure tests were performed with the aim of verifying the principle functionaliy of the infrastructure which is needed to allow the other sub-systems and the integrated system to run on the large scale. Given the overall TDAQ installation and commissioning schedule, DAQ specific tests were not planned for the LST but not excluded.

2.1.3 Aim of the Event Filter infrastructure tests

The general goal was to verify the functionality of the EventFilter infrastructure in TDAQ release 01-06, i.e. Event Filter without algorithms. This includes verification of EF specific changes (e.g. changes in EFIO protocol) and general changes (e.g. templated applications). The subsequent goal was to measure the response time of the EF subsystem during the state transitions Configure, PrepareForRun / Start and Stop of the TDAQ Run Control, in order to have a baseline measurement for the subsequent tests with algoritms. The transition times can be compared with those seen during the LST 2005. Note that there is considerable overlap with LVL2 and HLT algorithm tests.

Event building infrastructure tests have already been performed successfully at LST05 up to the required scale needed for the running of the Atlas experiment. Eventbuilding performance tests can only be exercised on a system providing adequate network topology and are being followed up at the installation in Point1. Therefore, no special EB tests are foreseen for LST06.

2.1.4 Aim of the HLT tests

The general goal is to measure the response time of the Level2 subsystem during the state transitions mentioned above. This should be done with as many trigger Algorithms as possible, and with a realistic way of accessing the involved data bases. The access to databases and associated files is, besides the interaction with Run Control, the one major aspect of running the HLT algorithms in Level2 and Event Filter that represents possible contention for resources across the HLT farms.

Note that to a large extent the aims and programs are the same for both L2 and EF, and thus considered together as HLT-common. However, there are some specific issues:

- For L2: additional performance studies on Data Collection and event building aspects
- For EF: possible additional test of calibration algorithms, and EFIO tuning.

2.1.5 Database test aims

Purpose of database usage in the LST

Test the performance of the various data paths involved between database servers, file servers, and the clients. Concentrate on the accesses which are likely the most demanding ones:

- Configuration of the HLT, both LVL2 and EF, including reading the job configuration from the trigger configuration DB, and reading the large amount of extra configuration data (patly from DB, partly from POOL files). Each of the L2 nodes and each of the EF nodes gets the same data, resp. Major system components used here are data replication, caching, or sharing mechanisms for DB contents and POOL files. This requires a number of nodes comparable to the number to be used at Point1 in 2007 datataking.
- For all detectors, emulation of DB read access with realistic estimates for data loads. This is in addition to the specific configuration of the trigger and of the calorimetry.
- Configuration of the calorimetry RODs, mainly reading pedestals and and optimal filtering coefficients. Each ROD gets different data so caching is not useful. An Athena-based server mechanism is mediating between the DB and the clients. This does not require a large number of nodes, but all the infrastructure to host the LAr RODs.

Access to the DCS database, though required during configuration, is not within the scope of LST 2006.

Context of the LST: related tests targeted at databases

The database infrastructure is used in both offline and online contexts. Apart from the LST, several other tests involving databases are in progress or planned:

- Database replication using the Oracle proprietary "Oracle Streams" mechanism is being excercised between the online (ATONR) and the offline (ATLR) Oracle servers, between Tier0 and the Tier1s, as well as from the MDT calibration centres to Tier0. The contents replicated is conditions data and TAG data. Databse replication is always accompanied by replication of files using the distributed data management implementation, DQ2.
- The TAG database provides metadata for selection of event samples for analyses. The large number of rows (10**9 events/year, corresponding to 200 Hz insertion rate into the database) is demanding. Some 200 metadata columns need to be indexed. The TAG mechanism is being exercised at present.
- The Data Streaming tests are part of the CSC (computing system commissioning) tests. Data streaming and luminosity blocks for RAW, ESD, AOD, and TAG data will be exercised for different streaming scenarios in

order to assess the impact on analyses. The streaming tests involve the generation of bytestream data from simulated data, including trigger and luminosity block information in the event headers, and writing out such data from a SFO to Tier0.

- The detector commissioning runs and cosmics data taking makes use of the overall system from DAQ to analysis, including databases, with increasing complexity as more detectors join in. It is proposed to complement the testing of the overall system with a "Full Dress Rehearsal" which would be based on simulated data (like in the Data Streaming tests) fed into some stage of TDAQ. The system installed at Point1, together with the offline Tiers, will be used in both cases.

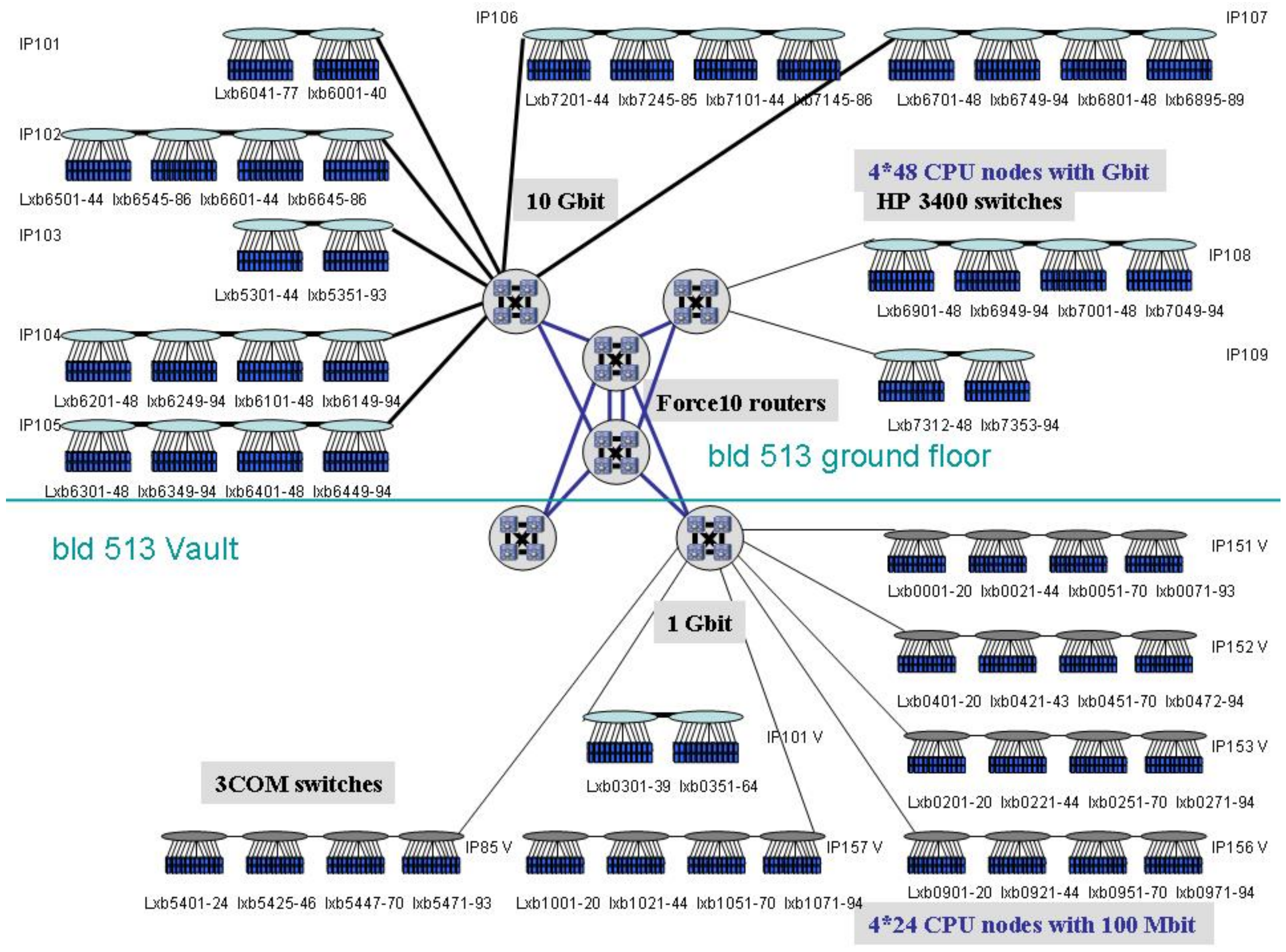
2.1.6 Testbed Description

The testbed was similar to the one used for the June/July 2005 tests. Up to 1000 PCs of the CERN IT LXBATCH testbed were used. All nodes were time synchronized via ntp . They were equipped with 2.4 GHz and 2.8 GHz Dual Pentium III processors, at least 1 GByte of memory running the Cern SLC4 operating system installation with selected system parameters adjusted to the needs of the tests. As there was no dedicated common file system such as NFS available for the nodes, the Software was replicated on the local disk of each PC. The TDAQ, HLT and Offline releases were packed together into a so-called image file and distributed with the farm tools. In addition, the release tdaq-01-06-02 including some patches was prepared as an RPM and distributed via the IT distribution mechanism. Afs was used for initial tests and to help installing the software. A few selected processes were run also from afs. The PCs were connected in groups of 22 via Fast Ethernet to local Fast Ethernet switches. Those were connected via Gigabit Ethernet uplinks to Gigabit Ethernet switchesthemselves being connected to the CERN network switching routers.The exact topology of the system is given in Figure 1. There was an installation phase on a set of 20 nodes before the tests started. The number of dual pc nodes available during the tests were increased throughout the testing period as listed below. The number of nodes which were operational was at most times significantly less then the number of nodes which were assigned to the tests. This was due to a number of technical problems on the side of the operating system installation.

- Phase 1: 8. November: 244 nodes available; 400 nodes assigned
- Phase 2: 14. November: 397 nodes available from ; 21st of Nov. 537 marked as up, 859 assigned
- Phase 3: 26. November to 10. December: 950 available, 1250 assigned

The Topology of the Test Bed is described in the followind diagram.

Figure 1 Topology of the LXBATCH testbed


IP101
Lxb6041-77 lxb6001-40
IP102
Lxb6501-44 lxb6545-86 lxb6601-44 lxb6645-86
IP103
Lxb5301-44 lxb5351-93
IP104
Lxb6201-48 lxb6249-94 lxb6101-48 lxb6149-94
IP105
Lxb6301-48 lxb6349-94 lxb6401-48 lxb6449-94
IP106
Lxb7201-44 lxb7245-85 lxb7101-44 lxb7145-86
IP107
Lxb6701-48 lxb6749-94 lxb6801-48 lxb6895-89
10 Gbit
4*48 CPU nodes with Gbit
HP 3400 switches
IP108
Lxb6901-48 lxb6949-94 lxb7001-48 lxb7049-94
IP109
Lxb7312-48 lxb7353-94
Force10 routers
bld 513 ground floor
bld 513 Vault
IP151 V
Lxb0001-20 lxb0021-44 lxb0051-70 lxb0071-93
1 Gbit
IP152 V
Lxb0401-20 lxb0421-43 lxb0451-70 lxb0472-94
IP101 V
IP153 V
Lxb0301-39 lxb0351-64
3COM switches
Lxb0201-20 lxb0221-44 lxb0251-70 lxb0271-94
IP85 V
IP157 V
IP156 V
Lxb5401-24 lxb5425-46 lxb5447-70 lxb5471-93
Lxb1001-20 lxb1021-44 lxb1051-70 lxb1071-94
Lxb0901-20 lxb0921-44 lxb0951-70 lxb0971-94
4*24 CPU nodes with 100 Mbit

2.1.7 Test Organisation and Approach

A team of HLT/DAQ testers and developers established the planning for the tests which was presented and discussed in the HLT/DAQ community. A working web page was established containing organisational information, details on testing tools and parameter settings, a description of the testbed and its layout, testbed testing time scheduling, meeting information, and the entry to the online test logbook. During the tests, regular short meetings were held to inform the test participants on the latest status and to discuss problems where necessary. For system integration, debugging and system tuning the test cluster shared among the testers. Timing measurements were run in exclusive mode, then the use of the PC farm was dedicated to a particular test only.

Complete: 🟥⬜
Responsible: Doris Burckhart
Author: Doris Burckhart 18 Dec 2006
Contributors: Doris Burckhart, Hans Von Der Schmitt
Last significantly modified by: Doris Burckhart 23 Jan 2007
Not yet reviewed

2.2 Operational Sequence for the State Transition Timing Measurements

The information on all the processes controlled by the Online System including their relationships, the Run Control hierarchy in the online system as well as start-up and shutdown dependencies is defined in the configuration database data file. A database data file which represents the partition was used to drive the tests involving all the components. Starting with booted but idle machines, the tests simulate the start of data taking activities by creating all the necessary processes in the defined order and then cycling the system through the states prescribed by the Run Control to simulate a data taking activity. At the end of these cycles, the system is shut-down in an orderly manner and all the DAQ related processes are destroyed. Timing values were written to a file and subsequently analysed. The configuration database schema allows to define segments to establish a Run Control hierarchy consisting of a number of detectors. Timing measurements were performed for the operator transitions and the internal breakdown on the synchronisation transitions. The operator transitions are defined as follows:

- Setup: start online system infrastructure including the root controller

- Close: remove online system infrastructure.

- Boot: start all supervised processes.

- Shutdown: stop all supervised processes.

- Configure: all progesses connect and are configured with their paramaters which mostly are read from databases

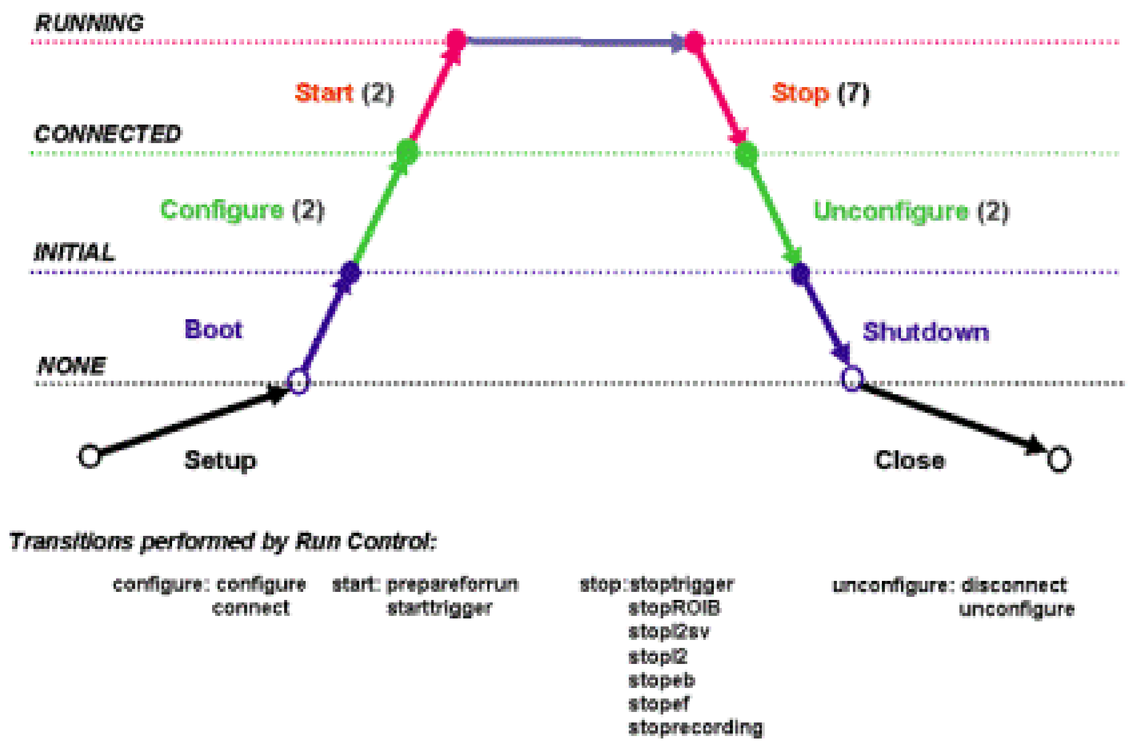
- Unconfigure: The processes unconfigure and disconnect

- Start: once all processes are started and configured the controllers and controlled processes go to the Running state.

- Stop: The controllers and processes go to the Configure state; the event flow is stopped in an orderly manner

The analysis of the setup, boot, shutdown and close times allows to test the process management and the initialization of Online Software System components (configuration). The state transitions give information on the communication overhead in the system.measurements were also recorded for the

The following graph illustrates the operator transitions. The individual transitions for each step are listed below.


RUNNING
Start (2)
Stop (7)
CONNECTED
Configure (2)
Unconfigure (2)
INITIAL
Boot
Shutdown
NONE
Setup
Close
Transitions performed by Run Control:
configure: configure connect
start: prepareforrun starttrigger
stop: stoptrigger stopROIB stopl2sv stopl2 stopeb stopef stoprecording
unconfigure: disconnect unconfigure

Complete:
Responsible:
Author: Doris Burckhart 29 Jan 2007
Contributors: Doris Burckhart
Last significantly modified by: Doris Burckhart 29 Jan 2007
Not yet reviewed

2.3 Tools for Test Execution and Analysis

2.3.1 Introduction

2.3.2 Tools and Utilities

Many individual tools and utilities were used for running the tests. There were automatic execution scripts, analysis tools, farm management tools, command line execution on multiple hosts, and checks on the correct functioning of the running DAQ/HLT system.

PartitionMaker was mostly used to generate the partition. Some partitions however were prepared manually and the older tool DBGeneration was upgraded for tdaq-01-06-02 by a tester for the use limited to DBStressor.was used for preparing and executing the HLT test runs. The hardware description file for the farm was prepared with the farm tools and the IT tool wassh helped for farm checking and the distribution of files. opmon recorded the state transition times and allowed for their analysis and comparison.

2.3.3 Test execution

setup_daq options:

- Measures timing of the state transitions using the options: -tt -ng
- Measures timing of the state transitions using the text_gui allows for more flexibility.
    - New option 'r' to start/wait/stop from connect state
    - Any allowed sequence of transitions can be concatenated
    - Can be used in automatic mode, for example:

```
echo "bfstudqMyTests" | setup_daq -p $TDAQ_PARTITION -ng -ut -nc $TEXT_GUI_PATH/
    text_gui
```

- 'time' option as was in use at LST05

Test execution based on a Phyton driver by Andre dos Anjos

This is described at PartitionRunner.

OPMON: Monitoring and Analysis for Run Control State Transition Timings

OPMON is a tool set to monitor, record, view and analyse run control state transitions. It is aimed to be used as a help for testing, verification and system tuning at small scale and large scale and in the installation at Point1. OPMON has three components:

- opmon_monitor monitors the timing of the state transitions, records the values in a root file and publishes them optionally in OH.
- opmon_merger allows to merge root files which result from different datataking sessions.
- opmon_analyser is the tool to be used once the datataking session is terminated. Based on one or more root files which are written by the opmon_monitor, the timings of the state transitions of all or a selection of controllers and controlled applications can be viewed in a variety of combinations in histograms or graphs. The results from several result files can be compared.

Detais are found in OPMON documentation.

Last year's Analysis tool is described at LST05Analysis.

Complete: ▭
Responsible: Doris Burckhart
Author: Doris Burckhart 29 Jan 2007
Contributors: Doris Burckhart
Last significantly modified by: Doris Burckhart 29 Jan 2007
Not yet reviewed

Chapter 3

Online Databases Used in the Tests

3.1 Database Support

The first paragraph gives the overview. More detail, including sheel commands used etc. is given in the following paragraphs which can easily be skipped for an overview. The last paragraphs lists problems met during the LST - either fixed already or remaining as action items.

3.1.1 Servers available

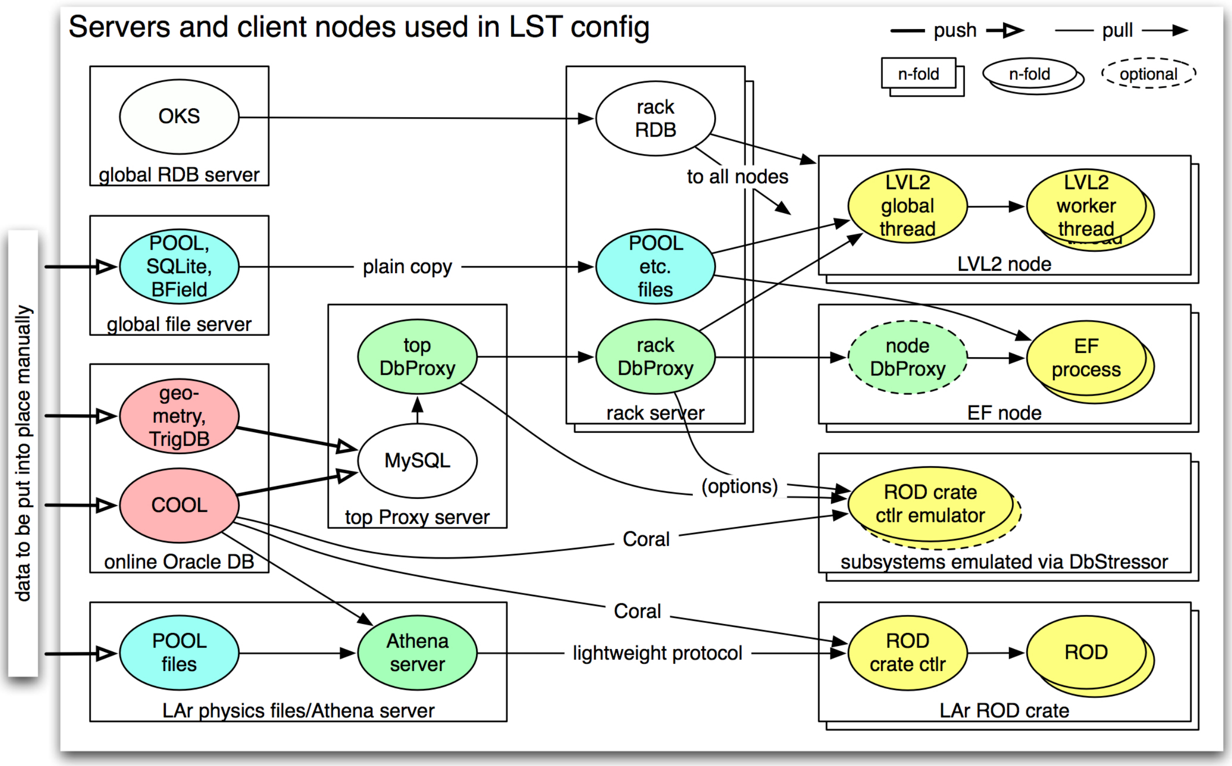
All data are made available on Oracle, MySQL, AtlasDBProxy based on MySQL, and SQLite (with the exception mentioned below).

- For Oracle, the LST uses the ATLAS Online RAC, which has server name ATONR. It is located in the CERN IT premises and is maintained by IT, as will be the case for production data taking. ATONR is at present connected to the CERN Public Network which will be changed after the LST to the ATLAS Technical Control Network ATCN. Oracle ATONR will be the primary source of all database contents needed online in production data taking. It has at present two server nodes which will be upgraded to six in early 2007. ATONR is the only database server where data are written to during data taking, e.g. DCS data.
- The MySQL server is running on one of the machines given to the LST by IT (lxmrra3801). Another server with the same data contents is running on lxmrra3804 and can be used in case the first one dies. MySQL server contents is read only. production running, the MySQL server will be replaced by SQL Relay which then connects to the Oracle server. While there are several options, this is the most likely solution and it has been already studied and verified.
- Each DBProxy behaves as a MySQL server but does caching of the data already requested. There will be many servers running - on the top level, at rack level, and for the EF also at node level. DBProxy is the main mechanism to serve data fast enough to the many HLT clients which all need the same data so caching can be fully exploited. There is a DBProxy server reserved for testing purposes at the LST (lxb5438). DBProxy is based on the open MySQL protocols as Oracle protocols are not made public. CORAL is used by the clients to connect to any MySQL, including the DBProxy. The top DBProxy can connect to Oracle via SQL Relay which can speak MySQL on the client side, and Oracle on the server side.
- All database contents needed is also available as (read only) SQLite files, with the exception mentioned of the Trigger Configuration DB which is not available on SQLite. For production data taking, it is planned access stable data (e.g. the geometry) from SQLite.

The figure below gives an overview of the database servers, file servers, and clients involved. The network structure available does not resemble exactly the structure present at Point1; "rack" in the LST means network switch level where a switch is responsible for about 60 nodes, while in production data taking, it means the physical rack housing about 30 nodes. "Rack" servers were made available for RDB and DBProxy. There was no equivalent of a distributed file system on rack level, just afs as global file system. Hence the files were placed on each node's local disk. Data contents was as follows:

- Data in databases are: COOL conditions data used for configuration, geometry, and trigger configuration
- Data in files are: POOL payloads for COOL, magentic field map
- As well as SQLite files for COOL and geometry data, serving as an alternative access to database data.

For the future SQL Relay solution, replace the MySQL server with SQL Relay, and the connections to Oracle by "pull" arrows.



3.1.2 Databases and connection strings

The database contents covers conditions data (COOL), geometry data, and trigger configuration data.

- COOL data
  These are the bulk of all database data which moreover change with relatively high rate, at least on a daily basis for the purposes of HLT running. Thus they must be distributed from a life database, and via a set of caches for efficiency.
  - Logical database names are by convention **COOL_xxx** where xxx is a detector name. Canonical values for xxx are **INDET, PIXEL, SCT, TRT, LAR, TILE, MUON, MUONALIGN, MDT** as well as **TDAQ, DCS, GLOBAL**. The COOL "database" (i.e. top folder) name for these is OFLP130. In addition we have **XPTWK130** and **LAROTEST** - the former for Athena-server based configuration of LAr RODs, the latter for LAr ROD and trigger configuration directly from the database.

- For LST, all logical databases are contained in one schema per technology. The COOL connection strings to OFLP130 for Oracle, MySQL, DBProxy, SQLite, and logical connections are, resp.

```
oracle://ATONR;schema=ATLAS_COOL_LST;user=ATLAS_COOL_LST_R;dbname=OFLP130;
    password=******
mysql://lxmrra3801;schema=ATLAS_COOL_LST;user=lst_user;dbname=OFLP130;password
    =******
mysql://lxb5438;schema=ATLAS_COOL_LST;user=lst_user;dbname=OFLP130;password
    =******
sqlite://X;schema=/pool/users/atlas/sqlite130/ALLP130.db;dbname=OFLP130
COOL_INDET/OFLP130
```

- Geometry data
  Geometry database contents is quite stable - it may change typically once per year. Deviations from geometry are recorded in COOL as conditions data. Thus geometry data could safely be distributed via !SQLite files.
  - The logical database name is **ATLASDD** for all technologies.
  - Connection information is

```
oracle://ATLONR/atlasdd user=atlasdd_reader password=******
mysql://lxmrra3801/ATLASDD user=atlasdd_reader password=******
mysql://lxb5435/ATLASDD user=atlasdd_reader password=******
sqlite_file:/pool/users/atlas/geomDB/geomDB_sqlite
sqlite_file:/pool/sw/lst06/DBRelease/2.8/geomDB/geomDB_sqlite
```

- Trigger configuration data (other than data for the algorithms)
  Trigger configuration data can change run by run, although the data volume is small. They must be distributed from a life database, via a set of caches for efficiency.
  - The logical database name is **TRIGCONF** for all technologies.
  - Connection information is (there is no SQLite representation)

```
oracle://ATONR/ATLAS_TRIGCONF user=atlas_trigconf_r password=******
mysql://lxmrra3801/TrigConf_lst user=lst_user password=******
mysql://lxb5435/TrigConf_lst user=lst_user password=******
```

3.1.3 Location of files on /afs and local /pool

The database content is complemented by POOL files referenced from COOL, by the magnetic field map file, and by a few setup files. Also, the SQLite databases reside in files.

- In general, the latest database release (3.0 at present) is found under

```
/afs/cern.ch/atlas/project/database/DBREL/packaging/DBRelease/3.0-nightly
```

- The directory structure of POOL files is similar on /afs and on the local file system under /pool which exists per node. Data have been replicated to all nodes used in LST. The mapping from the globally unique file identifiers (GUID) used in COOL to physical files if given in the POOL file catalogue (see further below). The paths to POOL files referenced by OFLP130 and XPTWK130, resp., start with

```
/afs/cern.ch/atlas/project/database/DBREL/packaging/DBRelease/3.0-nightly/
    poolcond
/pool/users/atlas/conditions/poolcond/vol0/
/pool/users/atlas/maxidisk/d29/LArCalorimeter/data/intr130/
```

- The magnetic field map is found at (DATAPATH to be pre-pended with this path so that the file is retrieved from the local file system)

```
/pool/sw/lst06/AtlasConditions/2.0.3-LST-1/InstallArea/share/bmagatlas*.data
```

- The POOL file catalogues are found at (DATAPATH to be pre-pended with this path so that the files are retrieved from the local file system)

```
/afs/cern.ch/atlas/project/database/DBREL/packaging/DBRelease/3.0-nightly/
    poolcond
/pool/users/atlas/conditions/poolcond/catalogue
```

- The SQLite files used are located at the following places, which are explicitely referenced from dblookup.xml

```
/afs/cern.ch/atlas/project/database/DBREL/packaging/DBRelease/3.0-nightly/geomDB
    /geomDB_sqlite
/afs/cern.ch/atlas/project/database/DBREL/packaging/DBRelease/3.0-nightly/
    sqlite130/ALLP130.db
          or /afs/cern.ch/user/a/atlcond/coolrep/sqlite130/ALLP130.db
/pool/users/atlas/geomDB/geomDB_sqlite
/pool/users/atlas/sqlite130/ALLP130.db
/pool/sw/lst06/DBRelease/2.8/geomDB/geomDB_sqlite
```

3.1.4 Data volumes

Data volumes used for HLT configuration were as follows:

- COOL database 38 MB
- Geometry database 12 MB (MySQL) or 23 MB (SQLite file) depending on DB technology
- !Trigger configuration database 1 MB
- POOL files 31 MB including the catalogues
- Magnetic field file 5 MB (packed).

Thus some 90 MB were used by each HLT node during configuration.

A wide range of data loads in COOL tables was generated and used in the tests with DBStressor, both on Oracle and on MySQL.

In addition to that, 1.7 GB of mostly POOL data for configuring the LAr RODs via the Athena server was made available but was not used in the tests.

A large volume (10 GB) of event data files was available locally to each ROS during running.

3.1.5 dblookup and authentication files

The dblookup.xml and authentication.xml are used by CORAL to translate logical connections to physical database server/schema, and to retrieve user/password per physical database server/schema. The files can be edited in place on the /afs location given. Pre-fabricated settings can be used from /pool as indicated below. CORAL_DBLOOKUP_PATH and CORAL_AUTH_PATH to be pre-pended with the appropriate path so that the files are retrieved as desired.
For running with a hierarchy of multiple DBProxy servers, individual dblookup.xml files are generated and used per node.

```
/afs/cern.ch/user/h/hans/public
/pool/users/atlas/lookup4dbproxy   (geometry from sqlite, COOL and trigconf from
    dbproxy)
/pool/users/atlas/lookup4sqlite    (geometry and COOL from sqlite, trigconf from
    dbproxy)
/pool/sw/lst06/DBRelease/2.8/XMLConfig/   (for the standard set delivered with
   the release)
```

3.1.6 Tools used to populate the databases

- Tools for COOL databases
  There is a variety of tools available from the DB project. The tools allow copying of COOL databases between the same or different database technologies (Oracle, MySQL, !SQLite). Also the conversion of older to new COOL data formats is supported. Other tools allow browsing of COOL databases.
  - Setting up for usage of the tools

```
source ~/cmthome/setup.sh -tag=AtlasOffline,12.0.3,gcc323
source /afs/cern.ch/atlas/software/builds/AtlasConditions/2.0.3/Database/
    CoolConvUtilities/cmt/setup.sh
```

-
  - The following commands have been used to get COOL data for the LST

```
AtlCoolCopy.exe "sqlite://X;schema=/afs/cern.ch/user/a/atlcond/coolrep/sqlite130
    /OFLP130.db;dbname=OFLP130" "oracle://ATONR;schema=ATLAS_COOL_LST;user=
    ATLAS_COOL_LST;dbname=OFLP130;password=******" -create -forcerecreate
AtlCoolCopy.exe "sqlite://X;schema=/afs/cern.ch/user/a/atlcond/coolrep/sqlite130
    /OFLP130.db;dbname=OFLP130" "mysql://lxmrra3801;schema=ATLAS_COOL_LST;user=
    lst_user;dbname=OFLP130;password=******" -create -forcerecreate
cp /afs/cern.ch/user/a/atlcond/coolrep/sqlite130/ALLP130.db /pool/users/atlas/
    sqlite130/ALLP130.db

AtlCoolCopy.exe "oracle://INTR;schema=ATLAS_COOL_LAR;dbname=XPTWK130" "mysql://
    lxmrra3801;schema=ATLAS_COOL_LST;dbname=XPTWK130;user=lst_user;password
    =******" -create -forcerecreate
AtlCoolCopy.exe "oracle://INTR;schema=ATLAS_COOL_LAR;dbname=XPTWK130" "oracle://
    ATONR;schema=ATLAS_COOL_LST;dbname=XPTWK130;user=ATLAS_COOL_LST;password
    =******" -create -forcerecreate

AtlCoolCopy.exe "oracle://ATONR;schema=ATLAS_COOL_LST;dbname=LAROTEST" "mysql://
    lxmrra3801;schema=ATLAS_COOL_LST;dbname=LAROTEST;user=lst_user;password
    =******" -create -forcerecreate
(LAROTEST needed conversion from COOL 1.2 to COOL 1.3 format before, which was
    done on ATONR)
```

- Tools for copying between MySQL databases
  The mysqldump utility can be used to generate a dump of one or more databases in form of an SQL script. This can then be uploaded to a MySQL server using the mysql source command. For the !SQLite geometry, used

```
cp /afs/cern.ch/atlas/project/database/DBREL/packaging/DBRelease/3.0-nightly/
    geomDB/geomDB_sqlite /pool/users/atlas/geomDB/geomDB_sqlite
```

- Other mechanisms used

- ◦ The Trigger Configuration group has a combination of shell and sql scripts to populate Oracle or MySQL databases with the jobOptions and TriggerMenu data needed for LVL1 and HLT configuration. These scripts have been used to populate ATONR and lxmrra3801.
- ◦ The DB Release makes available COOL and geometry databases for Oracle, MySQL, and !SQLite in form of files. These can be uploaded to the database servers, using Oracle imp and MySQL source commands, resp.

3.1.7 Things to remember...

- The DBProxy servers should be flushed by run control every time before HLT needs to reconfigure, e.g. at each stop-run.
  - ◦ Reason: IoV based COOL data will often have an IoV of "this-run-number (r) to infinity". The IoV is likely to be closed by making it "r to r". Then for new run r+1, the cached IoV "r to infinity" is still valid for run r+1, while it isn't in fact according to the DB.
  - ◦ This flushing does not introduce a performace penalty. DbProxy is mainly meant as a fanout mechanism to the many HLT nodes and not as a longer-term memory.
- Assuming that HLT algorithms won't perform a complete re-configuration at each start-of-run, they should be told about individual parameter changes in order to economise on startup times. Startup times have been observed to be O(min) when algorithms are started from scratch which is sizeable if a stop-start transition happens during beams time.

Problems solved and unsolved

- (SOLVED during LST) The CORAL release used gets confused with two connections which both go to MySQL and/or DBProxy, e.g. for geometry and COOL. All other combinations of database technology work. This happened for pure offline running, e.g. with InDetRecExample, and in LVL2 running as well.
  - ◦ Reason/solution: The problem comes when an Athena job has multiple CORAL clients, each loading the CORAL libraries into a context using the SEAL plugin manager. If they are loaded into different contexts, and two clients both use MySQL, the second one does not initialise properly. Modifications have been made to the connection code for RDB Access and Trigger Configuration in order to use existing context provided by POOL (packages RDBAccessSvc and TrigConfStorage).
  - ◦ Note that changes were also necessary in package TrigConfigSvc. These need to be made in sync with the changes to TrigConfStorage.
  - ◦ The error message was:
    RDBAccessSvc::connect() ERROR Exception caught: Type converter not loaded (CORAL : "ITypeConverter::MySQLAccess::SessionProperties::typeConverter" from "MySQLAccess")
  - ◦ As the geometry is very stable we intend to read it from !SQLite also for production datataking. Locking problems with SQLite files accessed via nfs may prevent this, in which case DBProxy would provide the service.
- (SOLVED during LST) CORAL cannot read the geometry from MySQL server lxmrra3801 while this works with atlmysql03. Happened offline and also in LVL2/EF tests.
  - ◦ Reason/solution: some primary keys had MySQL data type decimal(22,0), other fields had float(11,10). CORAL did not accept this in conjunction with MySQL 5. Schema on lxmrra3801/04 changed to int(22) and float, resp.
  - ◦ The error message was:
    DBRecordset::getData ERROR Schema Exception : Could not identify output type for "ELEMENTS_DATA.ELEMENTS_DATA_ID" (CORAL : "IQuery::execute" from "CORAL/RelationalPlugins/mysql")
- (SOLVED in Jan 2007) Geometry and COOL cannot both be read via DBProxy.

  - Reason/solution: MySQL accesses were not completely "atomic" in the CORAL version used, i.e., the database name needs always to be included within the queries.
  - IT has already provided a patched CORAL MySQL shared library which can be used now. Will be in release CORAL 1.7.
- athenaMT gets confused with DBProxy data after a "show tables from ATLAS_COOL_LST" has been issued from the client "mysql –user=lst_user –host= ATLAS_COOL_LST": it cannot connect any more. Does not happen offline in InDetRecExample.

Complete: ▨
Responsible: Hans Von Der Schmitt
Author: Hans Von Der Schmitt 16 Dec 2006
Contributors: Hans Von Der Schmitt
Last significantly modified by: Hans Von Der Schmitt 21 Dec 2006
Not yet reviewed

3.2 ATLAS DBProxy

3.2.1 Introduction

The DaqDbProxy is a mechanism through which the Atlas HLT can access configuration information from the database. Each proxy is able to cache information from the database in order to minimize the number of times a connection to the database is required.

3.2.2 Proxy Trees

The proxy acts as a client on the database side and as a server on the HLT side:

```
DATABASE
                     /   |   \
                    /    |    \
                   /     |     \
          dbproxy   dbproxy   dbproxy   ...
            /|\       /|\       /|\
           / | \     / | \     / | \
              HLT     PROCESSES
```

It is possible to connect a proxy to another proxy:

```
DATABASE
                         |
                         |
                      dbproxy
                    /    |    \
                   /     |     \
                  /      |      \
          dbproxy   dbproxy   dbproxy   ...
            /|\       /|\       /|\
           / | \     / | \     / | \
               HLT     PROCESSES
```

The hierarchical proxy structure limits the number of HLT processes with which each proxy must communicate. Right now the proxy can communicate using the MySQL protocol only.

3.2.3 Usage

The usage of the dbproxy application is:

```
daqdbproxy −h <dbserver_address>
options:   −x <dbproxy_port>       [default: 3306]
           −c <dbproxy_ctrl_port> [default: 3307]
           −d <dbserver_port>     [default: 3306]
           −z <poolsize_bytes>    [default: 128MB]
           −r <partition>         [default: none]
```

For example:

daqdbproxy -h atlmysql03.cern.ch -r testPartition

The required argument is on which the proxy listens in order to run multiple proxies on the same machine. The options may be used to specify a different port for the database when the database is not listening on the standard MySQL port.

The option allows to increase the total size of memory, in bytes, used by the cache. If the proxy needs more memory at some point during its lifetime than what is specified by this option, it will raise an exception.

The option specifies the port number which can be used to send messages to the proxy through the daqdbproxycontrol application. This mechanism is not used for normal operations since Run Control manages the proxy via IPC using the partition name specified by . The IPC application name is 'daqdbproxy'.

3.2.4 Recognized Commands

The following commands are recognized:

```
flush()                flushes the cache
    dump()             dumps the cache
    connect()          connects the proxy to the database
    disconnect()       disconnects the proxy from the database
```

3.2.5 Example Implementation in a Partition

Below is an example of the piece of xml that needs to be in the partition file for DbProxy:

```
<obj class="Application" id="DaqDbProxy−top">
 <attr name="Name" type="string">"DaqDbProxy−top"</attr>
 <attr name="Parameters" type="string">" −h pc−preseries−efp−31.cern.ch "</attr>
 <attr name="RestartParameters" type="string">""</attr>
 <attr name="ControlledByOnline" type="bool">1</attr>
 <attr name="IfDies" type="enum">"Restart"</attr>
 <attr name="IfFailed" type="enum">"Restart"</attr>
 <attr name="StartAt" type="enum">"Boot"</attr>
 <attr name="StopAt" type="enum">"Shutdown"</attr>
 <attr name="InitTimeout" type="u32">10</attr>
 <attr name="StartIn" type="string">""</attr>
 <attr name="InputDevice" type="string">""</attr>
 <attr name="Logging" type="bool">1</attr>
 <rel name="InitializationDependsFrom" num="0"></rel>
 <rel name="ShutdownDependsFrom" num="0"></rel>
 <rel name="Program">"Binary" "daqdbproxy_wrapper"</rel>
 <rel name="ExplicitTag">"" ""</rel>
 <rel name="Uses" num="0"></rel>
 <rel name="RunsOn">"Computer" "pcuwtr6.cern.ch"</rel>
</obj>
```

The L2 processes and EFs need to have dependencies set (InitializationDependsFrom) so that they are started after the proxies to which they connect.

The connection information currently is stored in two files, authorization.xml and dblookup.xml. The authorization files stores username and password info for each of the databases that will be accessed, and dblookup allows the user to select the database techonology to use (Mysql, Oracle, sqlite files, DbProxy). The location of these files is included in the partition as environmental variables.

3.2.6 Timing Studies

The DbProxy log file currently records information about each communication from-to clients and the database. The files can be read to learn about configuration times, bytes delivered by the proxy, and whether or not queries were answered by the DbProxy cache. Below is an example of analysis on a proxy log file taken during the March 07 Technical Run in which the Combined Trigger Slice (including electrons, photons, muons, jets, and taus) was configured:

```
 Total number of nodes: 42

Summary of Commands:
COM_INIT_DB    1142
COM_QUERY    9218
COM_FIELD_LIST    895
COM_PING    43782

Host:          First Query (s)      Last Query (s)        Delta      Bytes
    Time (ns)      # of Queries

0xa9523f7:  1174600091.4033  1174600157.2743  65.8710000515  344468
    18446744069639521320  214
0xa9523fb:  1174600091.4443  1174600157.2744  65.8301000595  345918  199791000
    216
0xa9523f6:  1174600091.4828  1174600157.2732  65.7904000282  344468  198904000
    214
0xa9523fa:  1174600091.5273  1174600157.2738  65.7465000153  344446  199409000
    216
0xa9523f9:  1174600091.5277  1174600157.2739  65.7462000847  344446  200720000
    216
0xa9523f8:  1174600091.5428  1174600157.2742  65.731400013  344468  197487000
    214
0xa9523f4:  1174600091.5583  1174600157.2737  65.7153999805  344468  197160000
    214
0xa9523fd:  1174600091.5656  1174600157.2742  65.7086000443  345193  225936000
    215
0xa9523fc:  1174600091.6060  1174600157.2734  65.6674001217  345918  198911000
    216
0xa9523f5:  1174600091.6612  1174600157.2741  65.6129000187  344468  196957000
    214
```

```
0xa9523f3:  1174600091.8707  1174600157.2745  65.4038000107  172234  102142000
    107
0xa9523fe:  1174600091.8896  1174600157.2747  65.3850998878  345193  200627000
    215
0xa9521ee:  1174600092.2429  1174600157.2640  65.0211000443  370528  241957000
    231
0xa9521f1:  1174600092.2996  1174600157.2715  64.9719002247  358561  211276000
    227
0xa9521ec:  1174600092.3041  1174600157.2677  64.9635999203  360066  210782000
    228
0xa9521e0:  1174600092.3135  1174600157.2694  64.9558999538  376536  219881000
    232
0xa9521da:  1174600092.3328  1174600157.2687  64.9358999729  360130  209446000
    229
0xa9521d9:  1174600092.3376  1174600157.2688  64.9312000275  360130  210754000
    229
0xa9521d7:  1174600092.3424  1174600157.2666  64.9241998196  361602  210294000
    229
0xa9521e1:  1174600092.3480  1174600157.2692  64.921200037  360260  209043000
    230
0xa9521e7:  1174600092.3533  1174600157.2748  64.9214999676  359947  207352000
    226
0xa9521ef:  1174600092.3583  1174600157.2700  64.9117000103  368409  211742000
    231
0xa9521e3:  1174600092.3635  1174600157.2750  64.9114999771  360442  209389000
    227
0xa9521e6:  1174600092.3804  1174600157.2671  64.8867001534  359876  208440000
    225
0xa9521df:  1174600092.3908  1174600157.2703  64.8794999123  368398  225606000
    231
0xa9521e8:  1174600092.3959  1174600157.2674  64.8715000153  360600  208805000
    226
0xa9521e9:  1174600092.4069  1174600157.2750  64.8681001663  363074  208956000
    230
0xa9521ed:  1174600092.4183  1174600157.2676  64.8493001461  360066  208708000
    228
0xa9521eb:  1174600092.4545  1174600157.2707  64.8162000179  360130
    18446744069622298320  229
0xa9521ea:  1174600092.4604  1174600157.2711  64.8106999397  358517  206903000
    227
0xa9521de:  1174600092.4663  1174600157.2681  64.8018000126  359151  207946000
    224
0xa9521e4:  1174600092.4719  1174600157.2713  64.7994000912  359892  206902000
    226
0xa9521db:  1174600092.4775  1174600157.2701  64.7926001549  360344  210854000
    226
0xa9521d8:  1174600092.4893  1174600157.2708  64.7815001011  360066  206523000
    228
0xa9521e5:  1174600092.5011  1174600157.2686  64.7674999237  360066  209549000
    228
0xa9521d6:  1174600092.5614  1174600157.2697  64.7083001137  361133  207758000
    228
0xa9521f0:  1174600092.5736  1174600157.2633  64.6896998882  363204  209407000
    230
0xa9521e2:  1174600092.5996  1174600157.2696  64.6699998379  360260  208917000
```

```
    230
0xa9521dd:  1174600092.6063  1174600157.2635  64.6571998596  360622  208064000
    224
0xa9521f2:  1174600092.9706  1174600157.2719  64.3013000488  356807  205669000
    224
0xa9521d5:  1174600093.4804  1174600157.2721  63.7916998863  221803  146883000
    138
0xa9521dc:  1174600093.5185  1174600157.2683  63.7497999668  362072  211851000
    226
```

The two unusually long configure times are the result of an overflow to be fixed in the printing of the log file. The python script that creates the table above, along with other information, by reading the DbProxy output file is attached to this page.

3.2.7 Comments

The proxy is started through the process manager mechanism by Run Control at boot time and it has the same lifetime as the partition.

This package uses ERS for reporting issues and messages. The proxy is currently in the online CVS Repository in the area DAQ/online/DaqDbProxy, but the location may change in the near future.

---

* read_proxyout.py.txt: python script to parse DbProxy output file

Complete:
Responsible:
Author: Sarah Demers 05 Sep 2006
Contributors: Sarah Demers
Last significantly modified by: Sarah Demers 05 Sep 2006
Not yet reviewed

Chapter 4

Configuration of the High Level Trigger through the TriggerDB

4.1 Configuration of the High Level Trigger through the Trigger Configuration Database (the TriggerDB)

4.2 Introduction

The configuration of the ATLAS High Level Trigger (HLT) is complex. The large and diverse ATLAS physics program requires a flexible and scalable approach to define and maintain a large variety of trigger signatures in a straight forward and robust manner. Bookkeeping should be simplified and aided, while at the same time the history of the trigger during the lifetime of ATLAS must be recorded. For these reasons a relational database was designed that can store all trigger configurations. A mechanism was put in place that can retrieve a configuration from the database and configure the Level 1 trigger hardware and HLT software for data taking. This database (the TriggerDB) and the software to configure the trigger are briefly described in this chapter, links for more detailed information are given.

The second part of this chapter describes the objectives and the results of the Large Scale Test 2006 with respect to the trigger configuration.

4.3 Configuration Technology

4.3.1 The Trigger Configuration Database

The HLT part of the trigger configuration database is designed to reflect the HLT concept of trigger chains (describing a complete physics trigger) which in turn consist of trigger elements (which are the logical equivalents of the reconstruction and selction algorithms). This information is used by the HLT steering. A separate set of tables holds the option values of all algorithms, services, and tools that make up an HLT job. For more information one can find a complete description of the database layout (including the Level 1 part). A graphic of the TriggerDB schema is also available as a pdf file. HLT configurations are extracted from HLT offline jobs that are prepared and tested for the purpose of online running; the information is then loaded into the TriggerDB. This process is supervized by the Online Integration Group.

4.3.2 Configuring the Level 2 and EventFilterTrigger Applications

Two applications are dedicated to run the Level 2 and the High Level trigger software, L2PSC and . Both use internally the GaudiSvc/ApplicationMgr class, which in ATLAS handles the initialization and configuration of all needed trigger algorithms and services in the proper order. For this purpose the ApplicationMgr holds ordered lists of all required dll's, all top-algorithms, and all services. These lists are stored in the TriggerDB as properties of the ApplicationMgr, after their retrieval from the TriggerDB the ApplicationMgr is able to initialize and configure all algorithms and services. The initialization and configuration of the top-algorithms and services means that they in turn retrieve their properties from the database and subsequently perform the necessary actions.

Special among the services and top-algorithms are the HLTMenuService and the HLTSteering algorithm. During initialization the HLTMenuService loads the trigger menu information (trigger chains and elements and their binding to the algorithms) from the TriggerDB. The HTLSteering algorithm, which controls the execution of the trigger algorithms, uses this information during the configuration phase to initialize and configure the trigger algorithms.

4.4 Large Scale Test Objectives and Results

4.4.1 Goals of the trigger configuration group during the Large Scale Tests 2006

The test program consisted of the following steps. The TriggerDB will be used to configure and run

1 a single Level 2 trigger node with direct access to the TriggerDB,
2 a Level 2 and an EventFilter node together with direct access to the TriggerDB,
3 a number (∼10) of Level 2 and EventFilter nodes with direct access to the TriggerDB,
4 and a number (40) of Level 2 and EventFilter nodes with access to the TriggerDB via the DBProxy.

4.4.2 Preparation of the TriggerDB and OKS

For the Large Scale Tests 2006 a designated software release (12.0.3-LST-1) was used. In this release four trigger configurations were prepared and loaded into the database. These were a configuration of a single *egamma* slice, a single *muon* slice, a single *jet* slice, and a configuration that combined these three slices.

To upload a trigger setup into the database the following steps were done:

1 The setups were tested within the !athenaMT and the !athenaPT framework, and a log of the job options of all configured algorithms and services was produced (using the athena THistSvc)
2 This job option log and the HLT trigger menu (all in form of xml files) are loaded into a MySQL database on the MySQL server that was running on the lxfarm (lxb5438.cern.ch). The version of the MySQL server and client software must be at least 5.0, since the trigger configuration upload scripts use stored procedures. After the upload of the 4 trigger configuration, 4 configuration masterkeys were produced, which in conjunction with the MySQL host and account names are all that is needed to specify a complete trigger configuration.

OKS specification

The following modifications were necessary to inform the trigger applications to configure from the database and which configuration key to use. In the OKS obj `L2PUConfiguration` and `PTConfiguration` the attributes `physicsAnalysisLibrary` and `physicsAnalysisConfiguration` need to be changed. Here that is shown for the EF nodes configuration

```
<obj class="PTConfiguration" id="PTConfig">
  <attr name="physicsAnalysisLibrary" type="string" num="4">"TrigEFPSC" "
      TrigEFServices" "TrigServices" "TrigConfigSvc" </attr>
  <attr name="physicsAnalysisConfiguration" type="string">
```

```
"POSTCOMMAND=&apos;&apos;;JOBOPTIONSTYPE=&apos;\&apos;DB\&apos;&apos;;
    WORKERTHREADS=&apos;0&apos;;L
OGLEVEL=&apos;ERROR&apos;;TRACEPATTERN=&apos;&apos;;PYTHONSETUPFILE=&apos;
    TrigEFPSC/TrigEFPSCPythonSetup.py&apos;;SHOWINCLUDE=&apos;0&apos;;
    JOBOPTIONSPATH=&a
pos;\&apos;DBType=MySQL:DBServer=lxb5438.cern.ch:DBUser=lst_user:DBPasswd=lst06:
    DBName=TrigConf_lst:Instance=EF:DBKey=1\&apos;&apos;;DLLNAME=&apos;[&apos;Tri
gEFPSC&apos;, &apos;TrigEFServices&apos;,
&apos;TrigServices&apos;]&apos;;PRECOMMAND=&apos;\&apos;doEFOnline=True\&apos;&
    apos;;EVENTSELECTOR=&apos;NONE&apos;
;FACTORYNAME=&apos;&apos;;JOBOPTIONSSVCTYPE=&apos;\&apos;TrigConf::
    HLTJobOptionsSvc\&apos;&apos;;MESSAGESVCTYPE=&apos;MessageSvc&apos;"
  </attr>
</obj>
```

Important are the additional library "TrigConfigSvc" that needs to be loaded, and the three modifications to the physicsAnalysisConfiguration string:

- JOBOPTIONSTYPE=apos;&apos;DB&apos;apos;
- JOBOPTIONSSVCTYPE=apos;&apos;TrigConf::HLTJobOptionsSvc&apos;apos;
- JOBOPTIONSPATH=apos;&apos;DBType=MySQL:DBServer=lxb5438.cern.ch:DBUser=lst_user:DBPasswd=lst06

4.4.3 Test Results

For the trigger configuration only functionality tests were performed, no timing measurements were executed. All four goals were achieved. The TriggerDB was successfully used to configure on a partition that consisted of 10 machines, each running 2 L2PUs and 18 machines, each running 2 PTs. The caching mechanism of the DBProxy was used to retrieve the configuration information.

4.4.4 Encountered Problems

During the tests two problems were encountered:

- MySQL Database multi-session access problem
- Job option configuration order defined by their order in the TriggerDB, but should be defined by their order in the C++ classes. This leads to problems when configuring the ApplicationMgr.

Complete:
Responsible:
Author: Joerg Stelzer 23 Jan 2007
Contributors: Joerg Stelzer
Last significantly modified by: Joerg Stelzer 23 Jan 2007
Not yet reviewed

Chapter 5

Tests with DBStressor

5.1 Tests of ROD configuration from Oracle

5.1.1 Introduction

5.1.2 Main heading

Complete: ☐
Responsible:
Author: Hans Von Der Schmitt 20 Dec 2006
Contributors: Hans Von Der Schmitt
Last significantly modified by: Hans Von Der Schmitt 20 Dec 2006
Not yet reviewed

5.2 DBProxy tests with DBStressor

5.2.1 Introduction

The DBStressor allows a user to stress the database by employing a large number of clients to simultaneously request information from the database. While the Stressor was designed in order to emulate the detectors accessing the database it can be used to test the performance of the DbProxy in under extreme conditions. Our hope was to test many different configurations of Stressors and Proxies, but the limited amount of time available for the LST, combined with the large amount of time it took (on the order of 20 - 40 minutes) to get a large ($\sim$ 500 node) partition to the point of the configure transition, did not allow for extensive testing. We present the results of our large partition test below.

5.2.2 Test Setup and Limitations

We tested a 600 node partition with a DBStressor Controller running on each node with the nodes distributed in segments of 25 nodes each. At the configure transition each Controller requested 10,000 lines from the Mysql Database LST06_100s_10K. Each line in the database consisted of 100 strings. The total size of the data requested per controller was therefore about 1.3 MBytes. The time required for the successful acquistion of the data was compared between the two following situations:

1) All Controllers requesting information directly from the Mysql server, lxmrra3801

2) All Controllers requesting information via one DbProxy, with the DbProxy connected to the Mysql server lxmrra3801

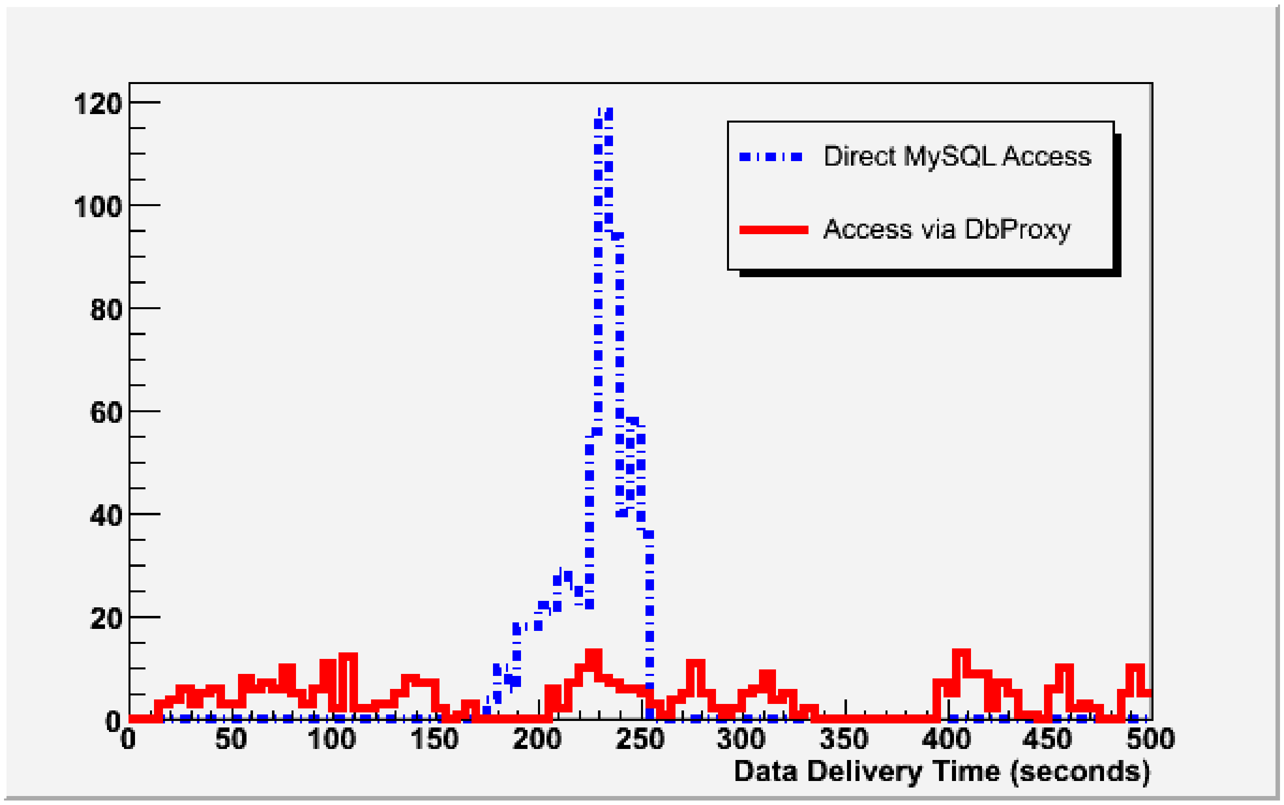
This test setup is unrealistic in that the DbProxy is intended to handle a much smaller number of clients in order to take the best advantage of its database caching. It is estimated that the final running conditions will require hundreds of proxies, with each proxy handling on the order of 10 to 30 clients. There was not sufficient time to allow for a test with a large number of proxies, which would have been a more relevant configuration.

5.2.3 Results and Conclusions

In both cases there was on the order of 1-2 nodes per segment that did not reach the configure transition and so were not included in the measurement. In case 1, going directly to Mysql, the fastest node configured in 172 seconds and the slowest node configured in 251 seconds. In case 2, going via the DbProxy, the fastest node configured in 15 seconds and the slowest node configured in excess of 500 seconds. The spread of configuration times is shown below.

---

* dbstressortest.gif:



Complete:
Responsible:
Author: Doris Burckhart 18 Dec 2006
Contributors: Doris Burckhart, Sarah Demers
Last significantly modified by: Sarah Demers 11 Jan 2007
Not yet reviewed

Chapter 6

Level 2 Tests

6.1 Trigger Level 2 tests

6.1.1 Introduction

6.1.2 Basic "stop watch" measurements

For the Configure transaction, a few time intervals were measured with a stop watch with a precision of ∼sec. This was done during the measurements taken with more detail using OPMON. The database connections used were:

- Geometry always from SQLite
- POOL and B-field files always local to each node
- Trigger configuration (1) from job options, (2) from the TriggerDB in MySQL
- COOL data (1) via DBProxy, (2) via DBProxy pre-loaded with all necessary data, (3) from the MySQL server, (4) from SQLite files local to each node, (5) from the Oracle server.

The time interval measured between pushing the Configure button and the fastest L2PU to be ready (column **first** in the table below), and between pushing the button and the majority to be ready (column **bulk**) was taken. In addition is shown the time for L2PU number 1 in subfarm 1, named L2PU-1, as recorded in the log file (column **1-1**). The time for the last L2PU was not measured by stopwatch although it determines the overall configuration time. This is shown in the OPMON distributions further below.

All times are in seconds. "nm" denotes that the value was not measured.

COOL data	**first**	**bulk**	**1-1**
Trigger configuration from job options			
DBProxy	72	90	62
DBProxy preloaded	71	85	61
MySQL	70	95	63
SQLite	71	85	60
Oracle	74	nm	65
Trigger configuration from the TriggerDB			
DBProxy preloaded	77	91	58

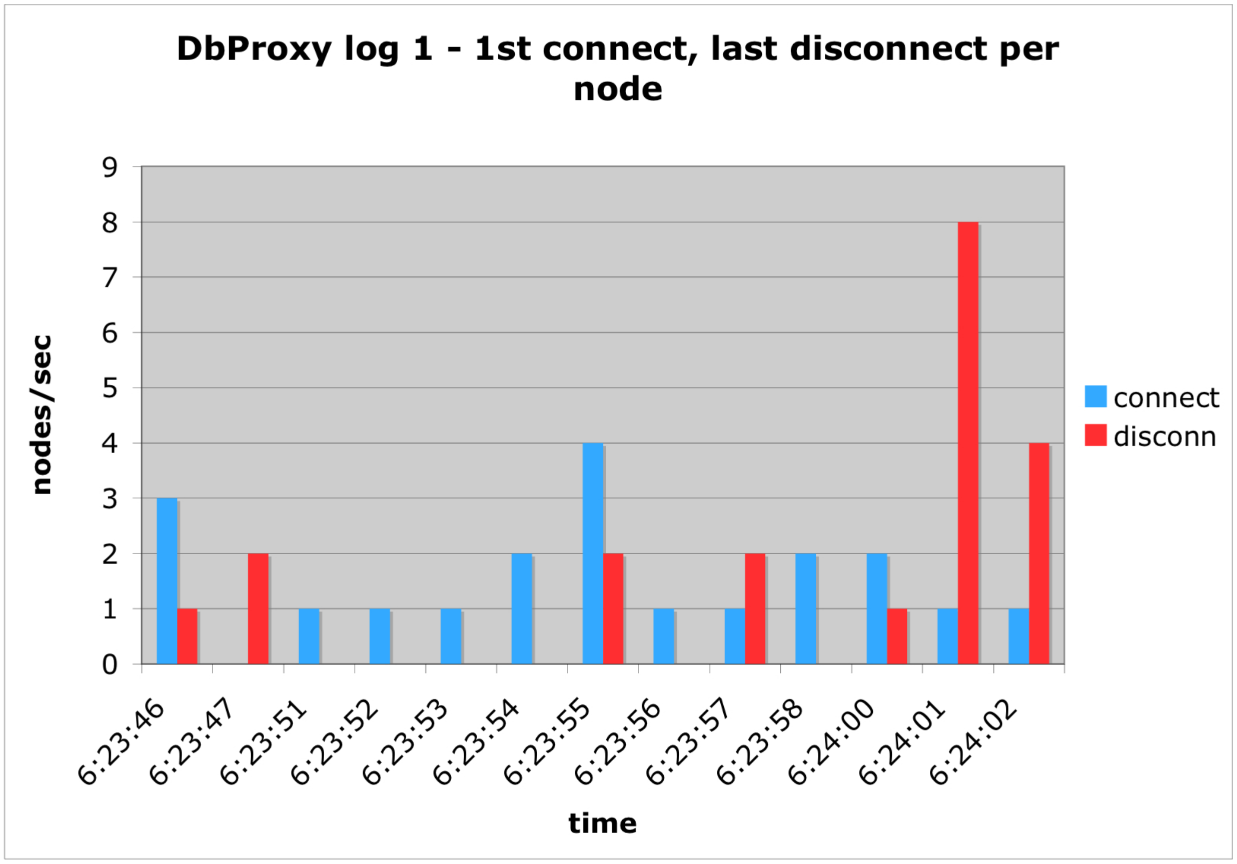
The fastest time was always observed for the same L2PU which may be due to its CPU speed as well as to the network structure. The bulk of the L2PUs all got ready within a few seconds from each other.

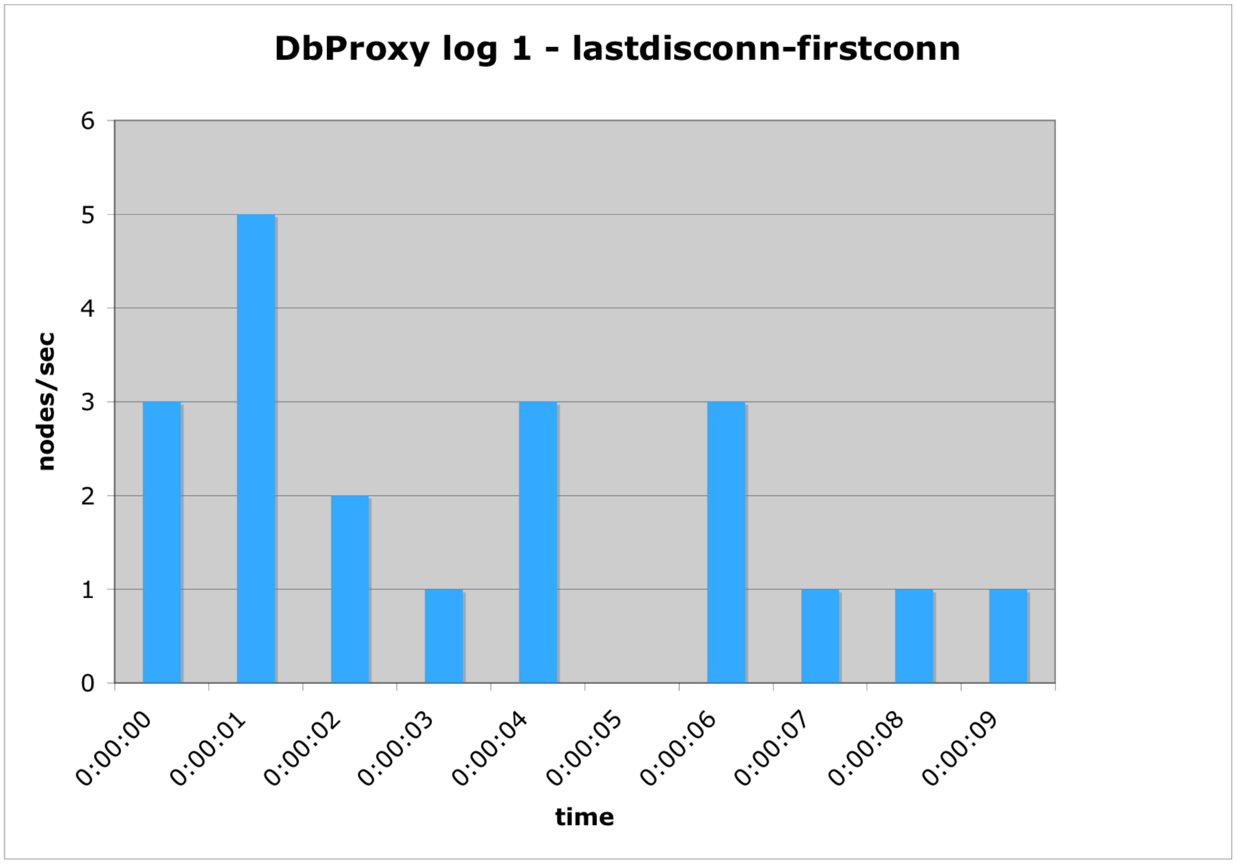
6.1.3 Measurements of DbProxy connect/disconnect

During the configuration exercises, the activity of the !DBProxy servers was written to log files. Access timings were then extracted from these files. The log files are taken from one of the DbProxy servers, serving 20 nodes, i.e. 40 L2PUs. During configure, each L2PU makes six DB connects and six DB disconnects, plus several queries inbetween. The DbProxy servers were set up to log the connects and disconnects, but not the queries (see figures below).

- Upper diagram: absolute time of the 1st connect and the last disconnect, one entry per node. All database accesses are contained in an interval of 17 sec. This is the interval from the first access to !DBProxy to the last one, for the set of all L2PUs. It is an upper bound of the time spent with DB accesses because the detailed activity of the DB server in terms of queries going on was not recorded. The server is likely to be idle for part of this time. The absolute time if DB connects (blue) and disconnects (red) is plotted.

- Lower diagram: time difference per node of last disconnect - 1st connect, one entry per node. This difference is 9 sec at maximum.





The data indicate that the majority of the total configuration time (75 to 90 sec) is spent in setup tasks other than DB accesses. Detailed studies of the remaining time (60 to 75 sec) spent outside of DB access (∼15 sec) are needed to significantly reduce the HLT configuration time.

Complete:
Responsible: Haimo Zobernig
Author: Doris Burckhart 18 Dec 2006
Contributors: Doris Burckhart, Haimo Zobernig
Last significantly modified by: Haimo Zobernig 19 Dec 2006
Not yet reviewed

Chapter 7

Event Filter Tests

7.1 Event Filter tests

(AtlasTDAQLargeScaleTests2006EventFilter

7.2 Introduction

The EventFilter system, as a part of the HighLevelTrigger and TDAQ total system, was tested to a certain extent in the LST 2005 in the WestGrid and the LST 2005 at CERN. The LSTs 2005 had very general goals and a general program. TDAQ and EventFilter *without* PESA algorithms showed good scalability up to order of $\sim$ 1000 hosts. The general/detailed TDAQ tests were therefore not among the goals of the LST 2006. The main interest was to validate new versions of TDAQ HLT software and to study the Database access.

7.3 Main goals for EventFilter@ LST06

Given the features and status of the EventFilter system and its functionality overlap with the Level-2 system it was considered optimal to avoid repetition of tests which could (and were decided to) be done with EventBuilding + Level-2. The focus of the EventFilter tests was therefore on Algorithms-to-CondDB performance with the DBProxy, as well as on general validation of the EventFilter with algorithms on the Large Scale.

The following goals were therefore chosen for the EventFilter @ LST06:

- run the largest possible EF partition and prove that it can work reasonably well;
- run the EF partition with PESA algorithms using the DBProxy to access the CondDB and verify operations of DBProxy as a part of aTDAQ+EF partition;
- study the performance of the (TDAQ+)EF+algorithms system in detail (config times)
    - at various scales (total size of TDAQ/EF partition);
    - with various SubFarm sizes and numbers of PTs / EF node;
    - comparing performance using CondDB with sqlite files, directly accessing the database, and using DBProxies.

In particular, to optimize working time, it was decided to first focus on EF-standalone partitions (without EventBuilding and Level-2), preloading event data in SFIemulators. The combination with Level-2 was also desirable, but there was not enough time (manpower) to achieve this.

Also, the tests were all based on only the EGamma PESA algorithm, which was fully integrated and prepared for the LST 2006. It was an attractive idea to include other algorithms *together* with EGamma, in order to study *addition* effect against signle algorithm (how load of each PT influences on DB and scalability), but it was also not completed due to lack of time (manpower), very unfortunatelly.

7.4 Program of tests for EventFilter@ LST06

Based on the goals of EF@LST06 (above) and on the general test plan for LST 2006, the tests of the EventFilter were subdivided and performed in 3 groups:

- a study/comparison of reading COOL DB data from sqlite/directMYSQL/rackDBproxy with SubFarms of 30 nodes, 2 PTs/node, 1-3-8-15 Subfarms
- a study of the DBproxy caching, comparing the times of the 1st TDAQ configurations (when the DBproxy forwards requests to the MYSQL server) to the 2nd config (when data is in the DBproxy cache from the 1st config); 3-8-15 SubFarms of 30 nodes, exercise repeated for 2 and 4 PTs/node
- studying effect of SubFarm size and number of PTs per EF node on DBproxy performance (in comparison of 1st and 2nd configure)
  - 3 SubFarms of 30 nodes, 1-2-4 PTs/node
  - 8 SubFarms of 5 nodes, 2 PTs/node
- run validate TDAQ+EF+algo+DB partition on the highest available/runnable scale (focusing on scalability issues, i.e. without detailed *functionality* validation which is not related to the system scale)

7.5 Particular Tests Results

In the tests studying the performance of algorithm-to-CondDB connections, the configuration times of each particular PT were recorded, thus, providing N values for each single run of TDAQ+EF+algo partition, where N is the total number of Processing Tasks (PTs) used. The whole config transition for one particular PT consists of several steps, which can be roughly divided into two major parts: first, the Athena jobOptions and libraries are read out from disk/network and loaded into memory, and second, the "DB access" part which contains connection to CondDB server (or emulated by file for sqlite or DBProxy), reading data from CondDB and building corresponding objects in program memory. Thus, the total config time is equal to "JO/Lib read" (JO = PESA Algorithm JobOptions) part plus "DB access" time. For each PT, the time of DB access was recorded as well as the whole time for PT configuration.

The DB data read during the config transition (within DB access time) can be in turn subdivided into geometry data (it was always read from sqlite files in these tests), magnetic field and some calibration data (POOL files on local disk were always used), and finally, the conditions data read from the Cond.DB servers. The performance of PT/Algorithms reading the conditions data in the later case, from Cond.DB servers was studies mainly in these tests, alternating the single Cond.DB server with DBProxy caching programs on each EF SubFarm, and with sqlite files read out from local disks. In this case, the particular references to Cond.DB source were switched in "dblookup.xml" and "authentication.xml" files, and special tools were developed in BASH scripts to map particular EF node to address of DBProxy host of given SubFarm "server". (MySQL DB was always used as Cond.DB).

The parameters of PESA Trigger Algorithm itself were read out using the JobOptions files from local disk. We did not studied the comparison of the system with JobOptions with respect to the scenario with reading Algorithm parameters from corresponding TrigConfig. DB, because the amount of this parameters data is negligible w.r.t. to the Cond.DB data, and such studies were considered by other, DB-study group.

There are two ways to interprete these numbers:

- since the whole TDAQ finishes a transition (here the "config" one) when the *last* component finishes it (in our case, the PT), then one resulting value is the maximal timing of config among PTs (i.e. the max in distribution);

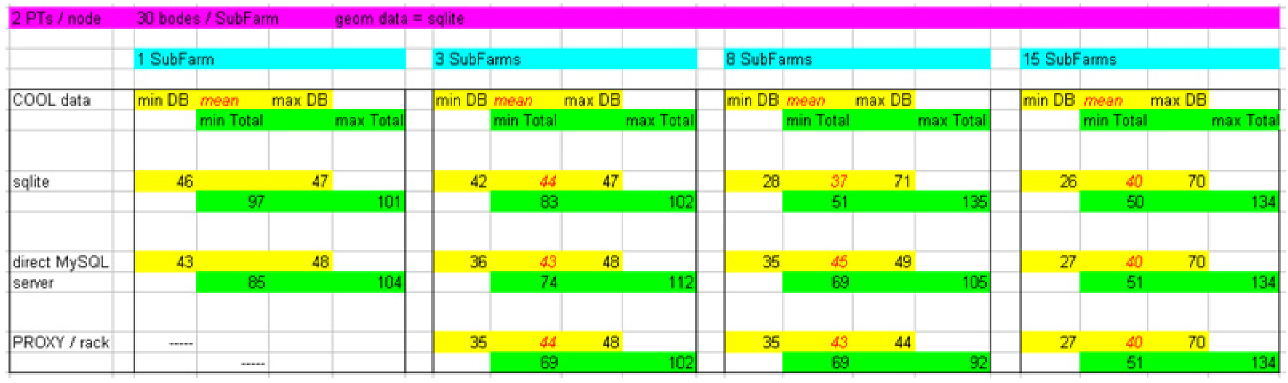
- since we are dealing with large number of PTs, order of 100s, then the distributions themseves and their shapes are also important, because in particular, two distributions may have very different shapes and mean values, while very close maximums (e.g. due to fluctuations of just a few PTs).

So, the results are presented in two forms: distribution of config. times of all PTs and min/max of these times.

Note that only one run of TDAQ system was performed/logged for each study case due to the fact that each particular run needs extra efforts for preparations, checks and start-up itself, and then extra actions after the run: collecting logs with the measured times, cleaning logfiles from disk and check/cleaning processes from previous run before new one. Thus, no any significant statistics available within each particular study case for factors such as external network/CPU loads, network caching, etc. On the other side, since for each study case the number of EF nodes and PTs used is >> 1, then we can assume good statistics when considering factors related to functionality of PT/Algorithm/DB components as SW programs.

Test Group 1: compare sqlite/directMYSQL/DBProxy

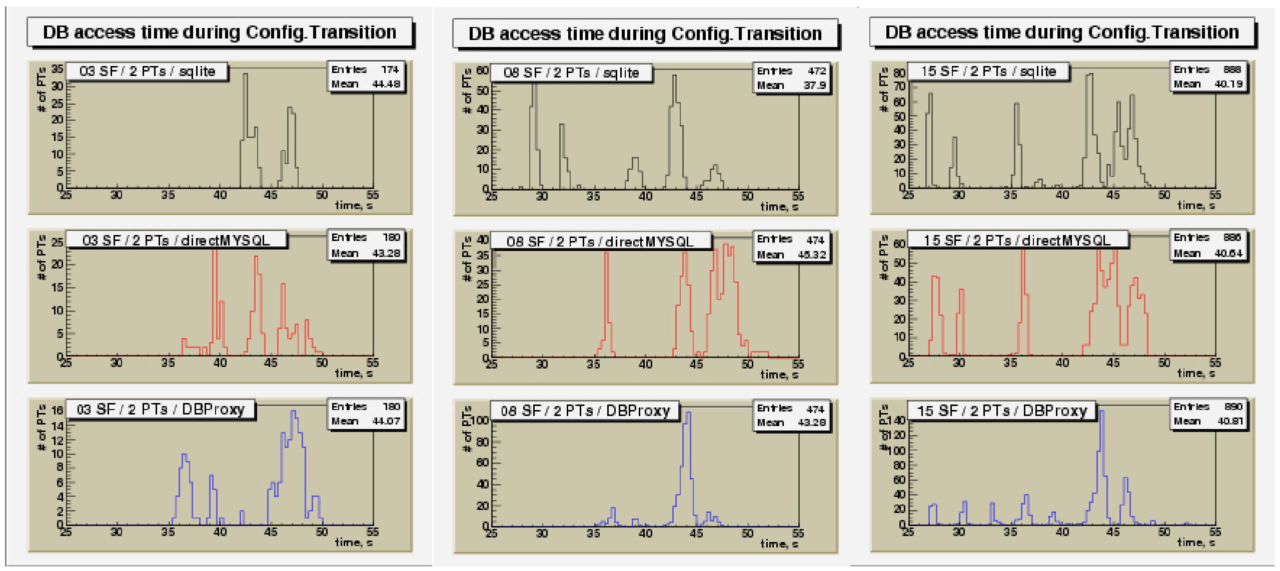
The table below summarizes the measurements in Test Group 1, presenting min and max values for each distribution. The green cells correspond to the total times of config transition for PTs, while the yellow cells correspond to the part of this time spent on DB connection/reading, i.e. without reading JobOptions and loading libraries.



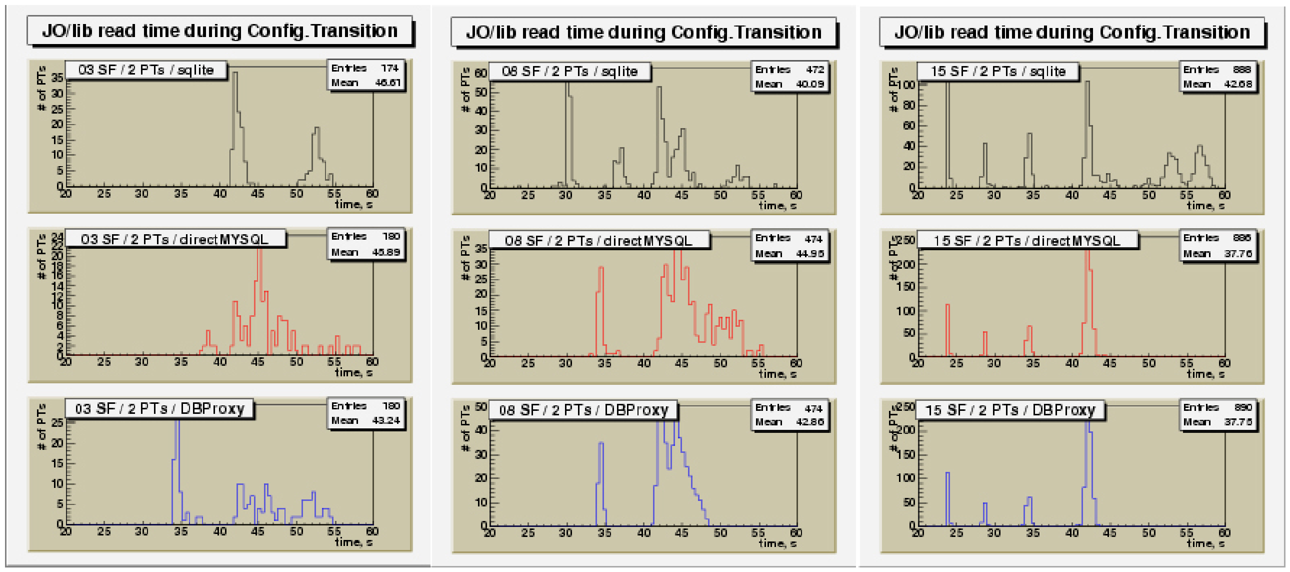
2 PTs / node 30 bodes / SubFarm geom data = sqlite

	1 SubFarm					3 SubFarms					8 SubFarms					15 SubFarms				
COOL data	min DB	*mean*	max DB	min Total	max Total	min DB	*mean*	max DB	min Total	max Total	min DB	*mean*	max DB	min Total	max Total	min DB	*mean*	max DB	min Total	max Total
sqlite	46		47	97	101	42	*44*	47	83	102	28	*37*	71	51	135	26	*40*	70	50	134
direct MySQL server	43		48	85	104	36	*43*	48	74	112	35	*45*	49	69	105	27	*40*	70	51	134
PROXY / rack	-----			-----		35	*44*	48	69	102	35	*43*	44	69	92	27	*40*	70	51	134

The min/max values in the table above should be complemented by corresponding distributions of PT config times shown below. Black curves correspond to the reading Cond. data via sqlite files, the red curves correspond to direct connections from each PT to single MySQL DB server, and the blue curves represent measurements with DBProxy on each EF SubFarm. The left column is test with 3 EF SubFarms, the middle corresponds to 8 SubFarms, and the right one is 15 SubFarms.

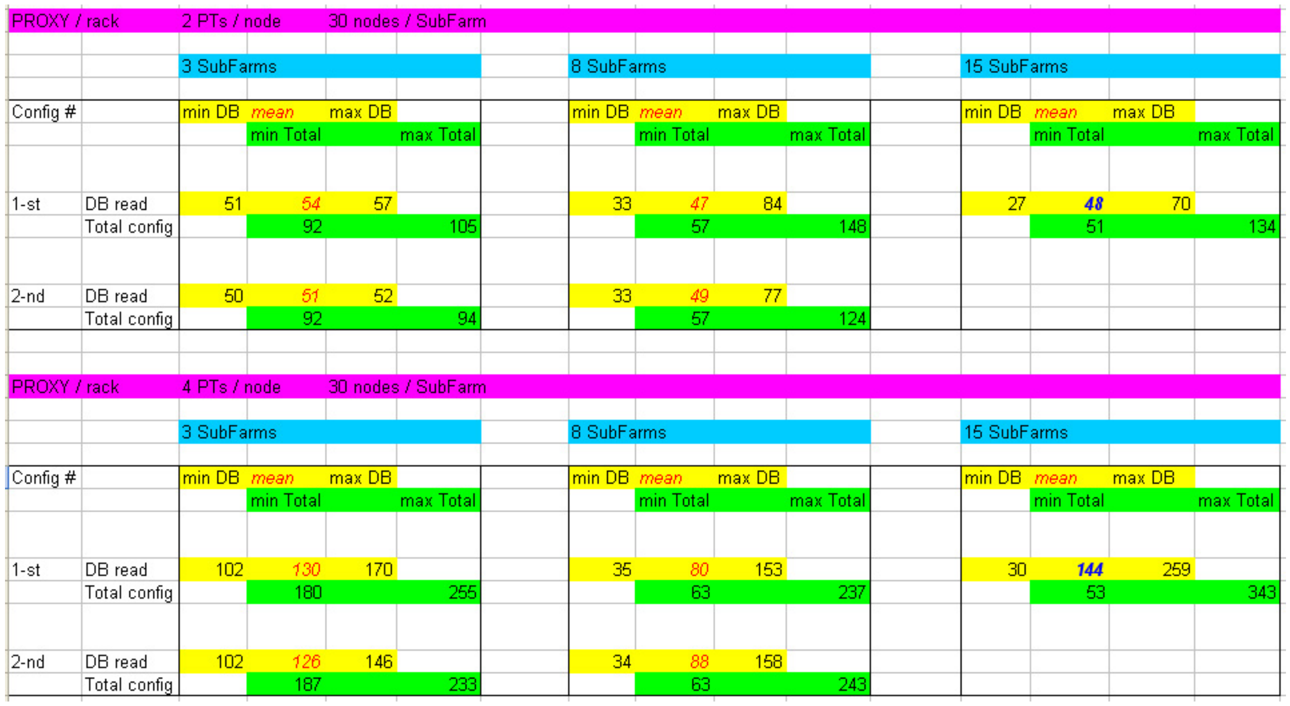


//– Fig. 1a. Distributions of JO/Lib read at config. times for PT/Algorithm using Cond. data from sqlite/directMySQL/DBProxy for 3,8,15 SubFarms –//



Test Group 2: compare 1-st and 2-nd configure with DBProxy

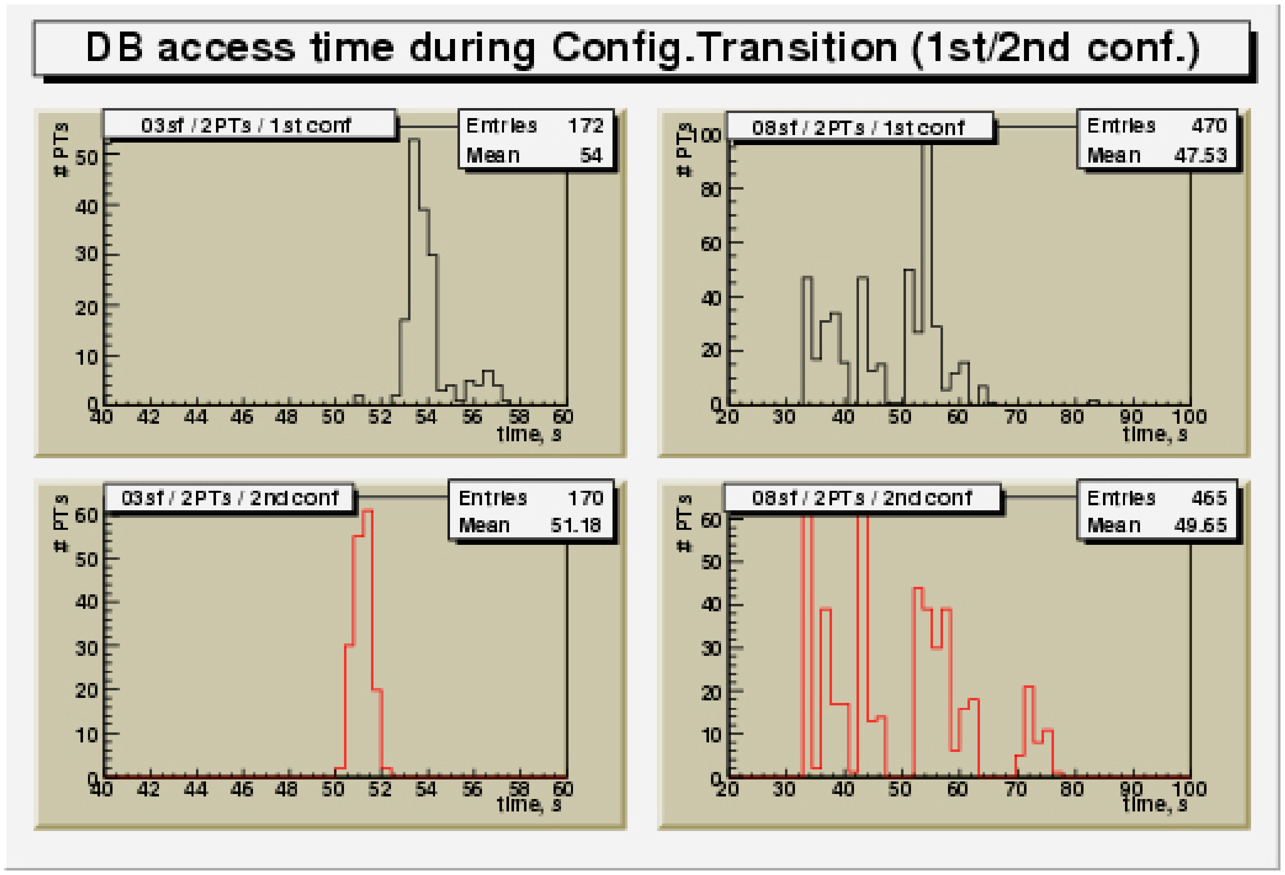
Caching effect in DBProxy was studied by comparing timings of 1-st and 2-nd config of EF partitions with DBProxy applications, for EF partitions of 3, 8 and 15 SubFarms. Tests were repeated for 2 PTs /node and for 4 PTs /node. Below is the summary table with min/max timing values.



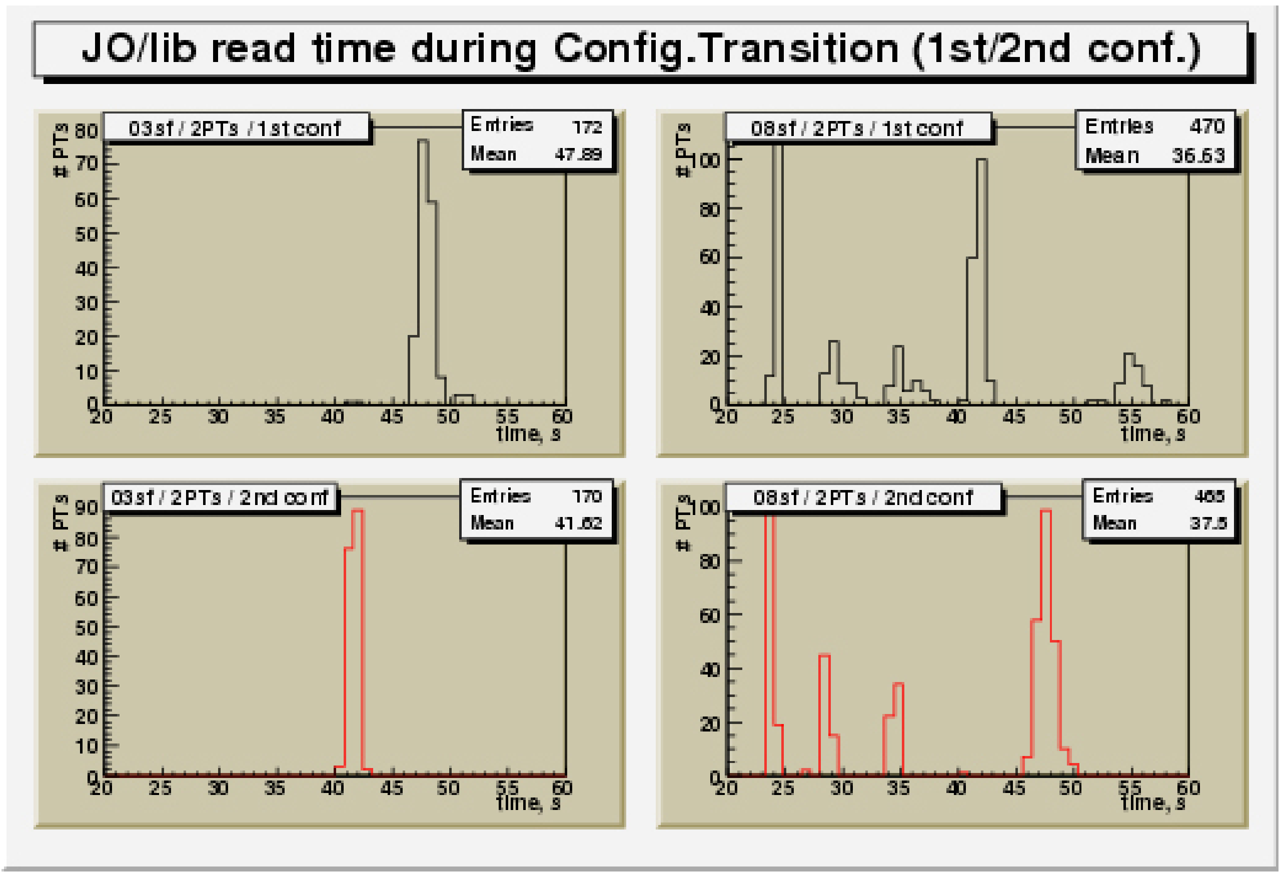
PROXY / rack	2 PTs / node	30 nodes / SubFarm													
		3 SubFarms					8 SubFarms					15 SubFarms			
Config #		min DB	mean	max DB	min Total	max Total	min DB	mean	max DB	min Total	max Total	min DB	mean	max DB	min Total	max Total
1-st	DB read	51	54	57			33	47	84			27	48	70		
	Total config				92	105				57	148				51	134
2-nd	DB read	50	51	52			33	49	77							
	Total config				92	94				57	124					

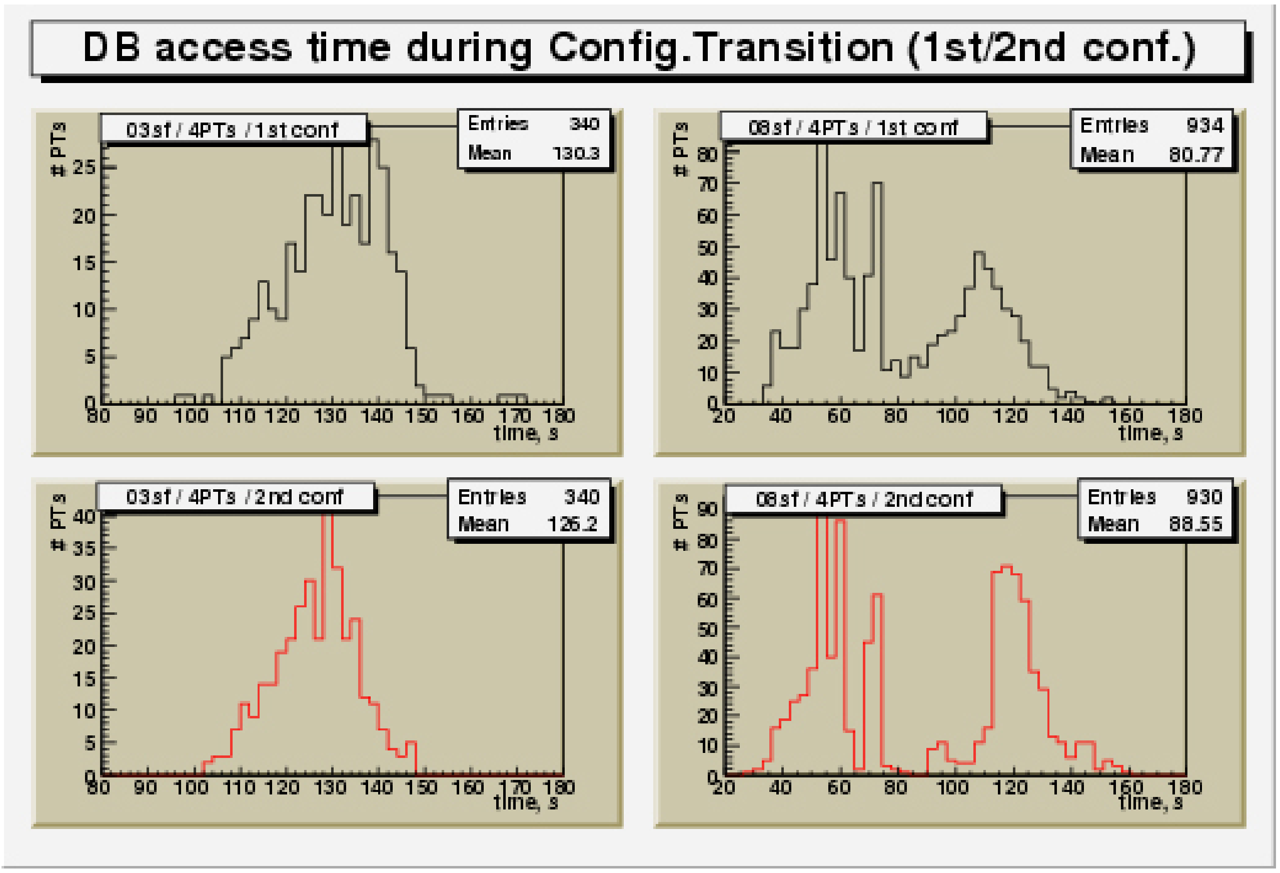
PROXY / rack	4 PTs / node	30 nodes / SubFarm														
		3 SubFarms					8 SubFarms					15 SubFarms				
Config #		min DB	mean	max DB	min Total	max Total	min DB	mean	max DB	min Total	max Total	min DB	mean	max DB	min Total	max Total
1-st	DB read	102	130	170			35	80	153			30	144	259		
	Total config				180	255				63	237				53	343
2-nd	DB read	102	126	146			34	88	158							
	Total config				187	233				63	243					

And below there are distributions of PT config timings compared for 3 and 8 SubFarms, first for 2 PTs/node, and later for 4 PTs/node.

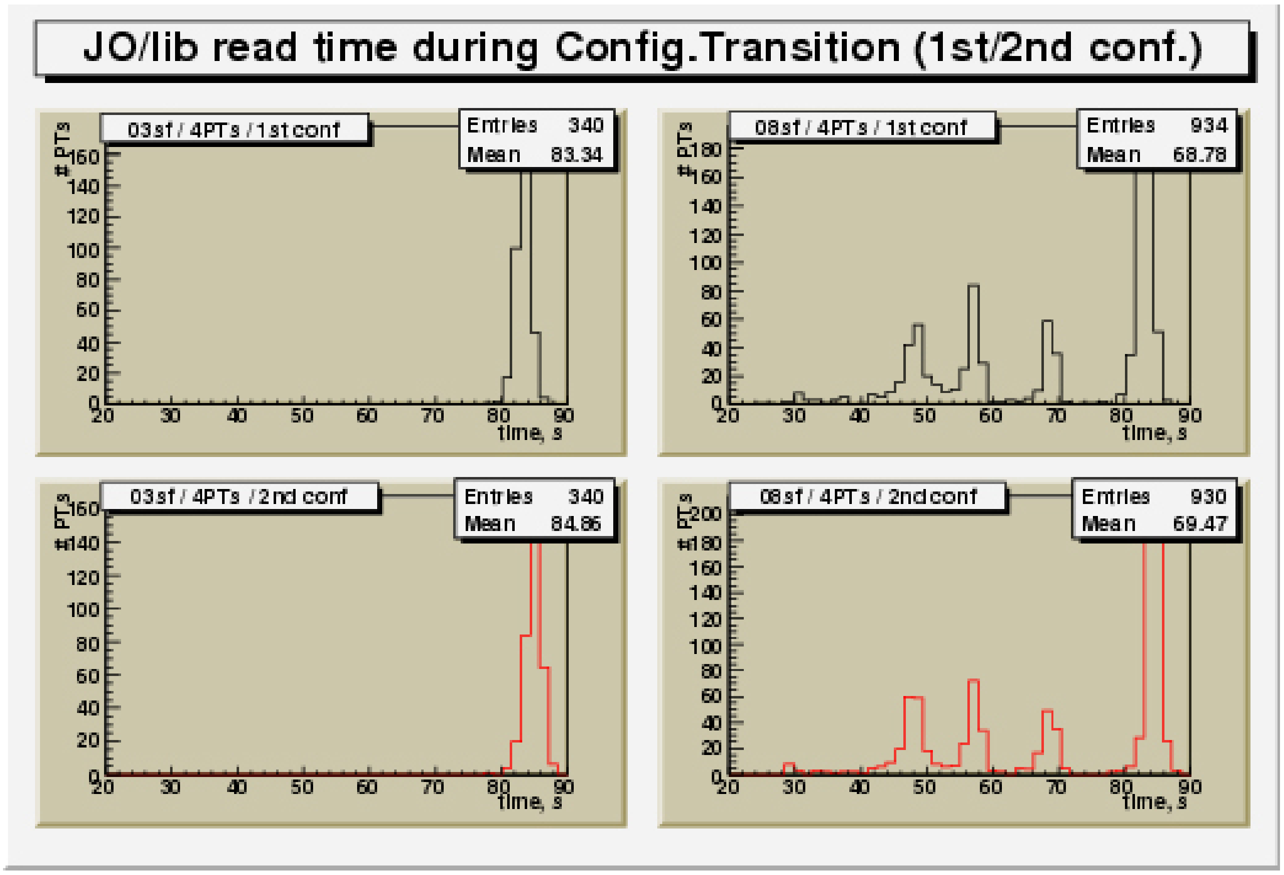


//– Fig. 2a. Distributions of JO/Lib read times for PT/Algorithm configured from DBProxy, comparison of 1-st (top) and 2-nd cached (bottom) configurations, 2 PTs / node –//





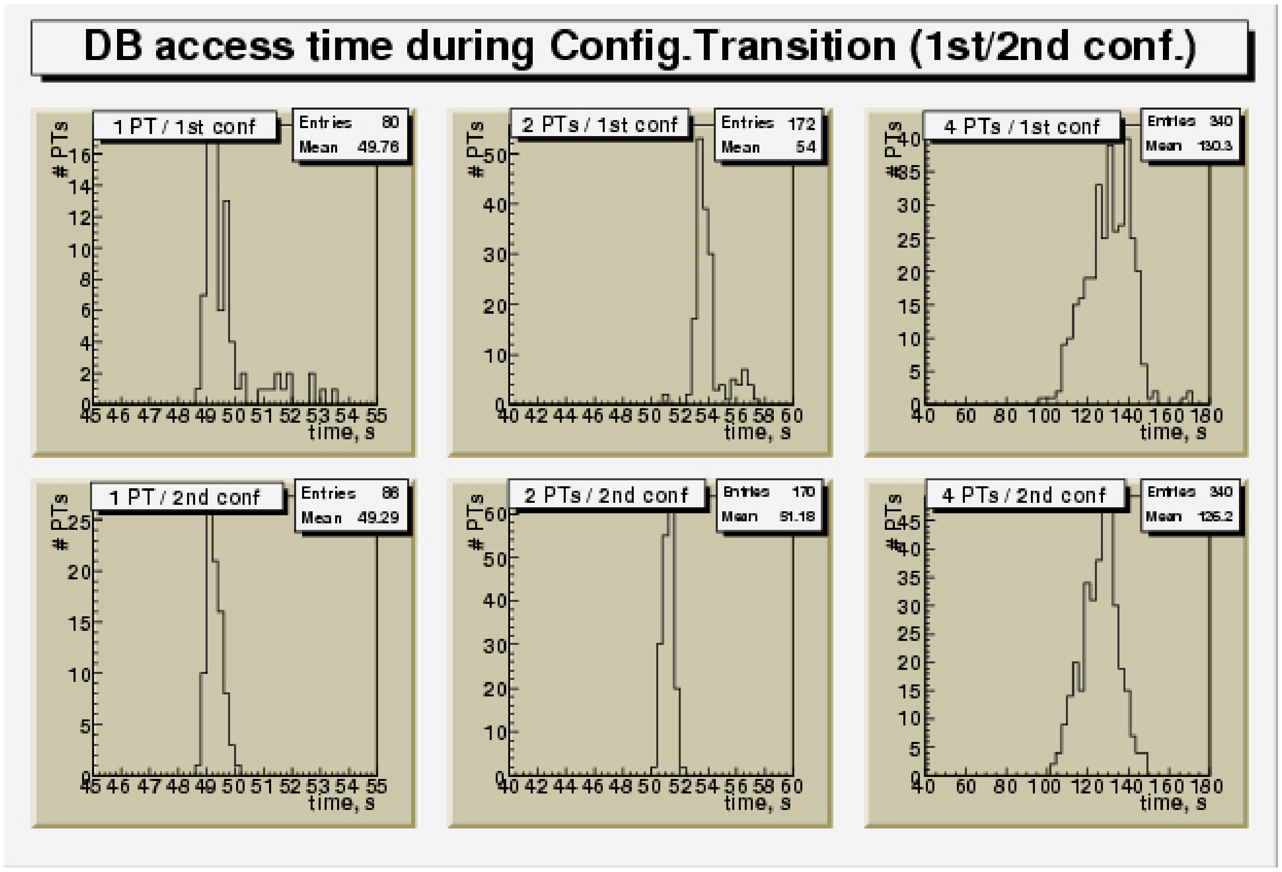
//– Fig. 3a. Distributions of JO/Lib read times for PT/Algorithm configured from DBProxy, comparison of 1-st (top) and 2-nd cached (bottom) configurations, 4 PTs / node –//



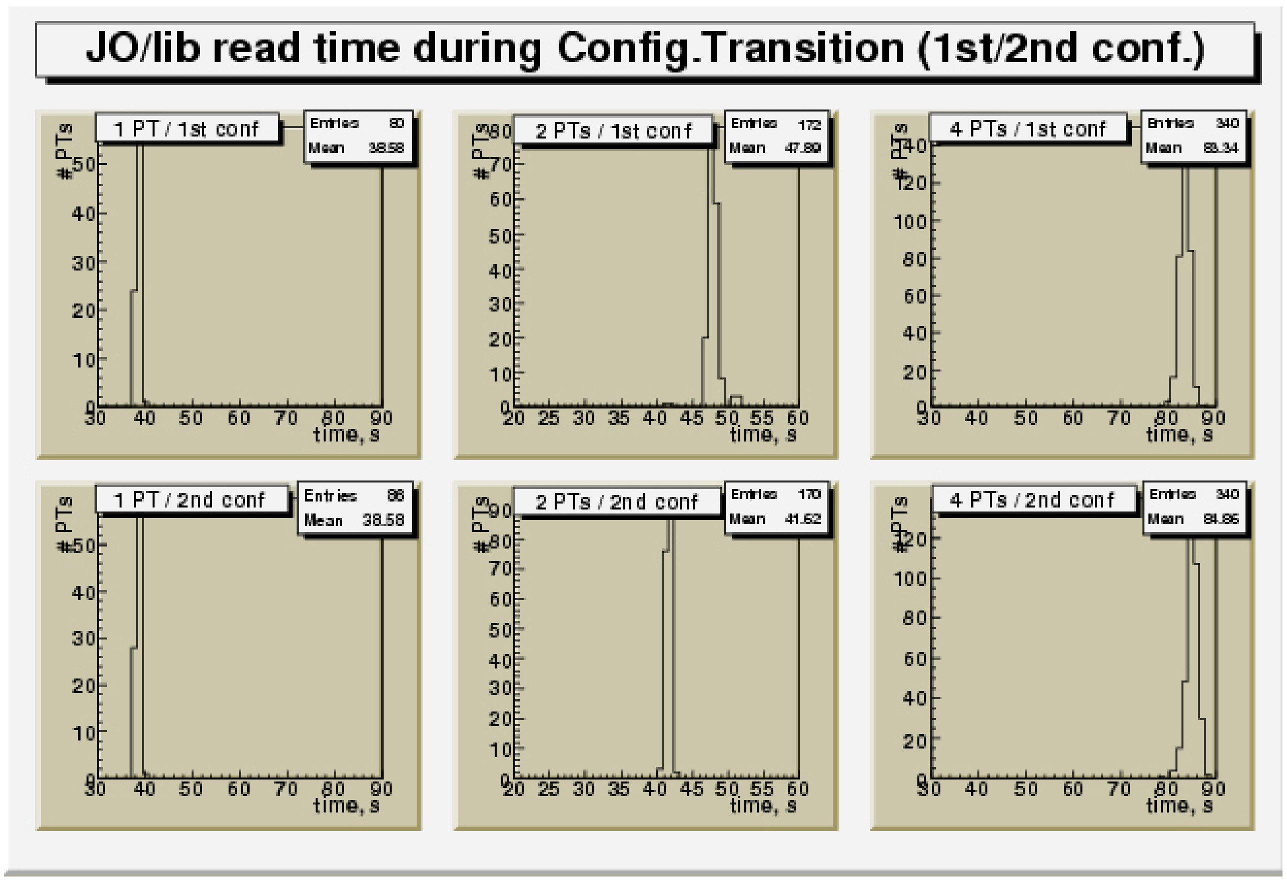
Also, 1-st to 2-nd configurations can be compared for EF partitions of the same sizes (3 SubFarms of 30 nodes) with various numbers of PTs per EF node: 1, 2 and 4. This is shown in the next topic (just below).

Test Group 3: SubFarmsizes and number of PTs / EF node

Another tests were done for fixed size of SubFarms with variation (comparison) of number of PTs per EF node: 1, 2 and 4 PTs.


DB access time during Config.Transition (1st/2nd conf.)
1 PT / 1st conf
Entries 80
Mean 49.76
2 PTs / 1st conf
Entries 172
Mean 54
4 PTs / 1st conf
Entries 340
Mean 130.3
1 PT / 2nd conf
Entries 86
Mean 49.29
2 PTs / 2nd conf
Entries 170
Mean 51.18
4 PTs / 2nd conf
Entries 340
Mean 126.2
# PTs
time, s

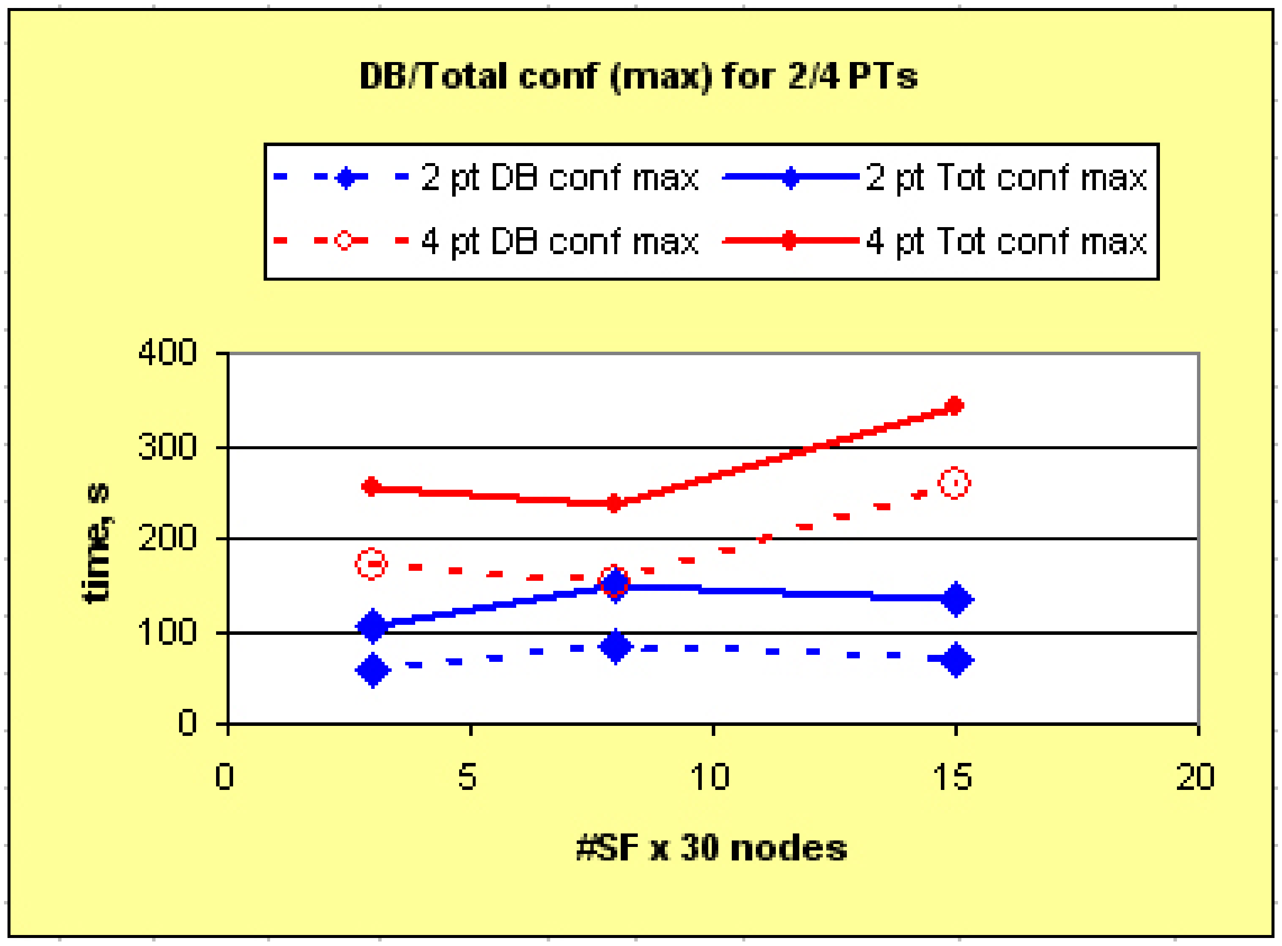
//– Fig. 4a. –//



Test Group 4: run validate at highest scale

The largest EF+algo+DBProxy partition succeeded to run was 15 SubFarms of 30 nodes with 4 PTs per each EF node, which comprises of 15x30=450 EF nodes, and 450x4=1800 PTs in total.

From Tests of Groups 1-3 we can draw the scaling curves of configuration times against partition size (# of SubFarms which is proportional to total # of nodes and # of PTs). Note that in the cases of 4 PTs/node there are two factors which may contribute in bigger config. values: bigger total number of PT as clients of DBProxy, and on the other side, CPU resource concurence, when each of 2 CPUs runs 2 PTs each instead of just 1.



General Results

- able to run EF-only partition with egamma algo (top events data preloaded @ SFIemu, no L2) up to 15 SubFarms x 30 nodes x 4 PT/node
- performance (config times) of EF with DBproxy studied @ various modes/sizes
  - in some cases there is clean improvement (with/without DBproxy) seen
  - in some the same performance observed (no improvement seen).

Several factors can contribute here explain:

-
  - Client-type and server-type functionalities/design of DBproxy should understood more exactly with specific of fast/simultaneous requests from many PT clients;
  - Overall topology of partition (number/sizes of SubFarms) may influence;
  - The network HW was not homogenous (connection speeds, number of net switches between SubFarms) and extra activity/load could influence.

7.6 Lessons and Follow-Up

Problems Observed

- few cases EFD died silently;
- few cases DBconnection initialisation failed / PT crashed;
- DAQ GUI hanged up while DAQ was running for very large partition (15 SF x 30 nodes x 4 PTs, start of ISL caused this)
- lost processes nodes status – FarmTools are needed/important !

The log files in all major cases are saved, the cases are to be investigated

Follow-Up Actions

- more detailed studies of DBproxy on SLAC/P1 clusters
- ideas for possible DBproxy use:
  - run permanently on each HLT node (as pmg_agents) ?
  - templated config object for dbproxy aplication
  - optimal allocation (1 or 2 or 3 per SubFarm) ?
  - more parametrization of OKS objects improving PMaker

7.7 Overall Conclusions

- Much better w.r.t. LST05 !! (more reliable, able to run directly up to 15 Sf x 30nodes x 4 PTs with just very few PTs crashed ignored, but not bottleneck for whole TDAQ). Not inacceptable long conf time ($\sim$ 5 min)
- EF/algo/DBproxy functionality in place and much better under control
- Several/many interesting detailed performance studies done, to be analyzed more
- Still not perfect system and many thing to study in more details improve

---

Complete: ▨
Responsible: Serge Sushkov
Author: Doris Burckhart 18 Dec 2006
Contributors: Doris Burckhart, Sarah Demers, Serge Sushkov
Last significantly modified by: Serge Sushkov 26 Jan 2007
Not yet reviewed

Chapter 8

DAQ and HLT Infrastructure Startup and Fault Tolerance Notes

8.1 DAQ, HLT and EB Infrastructure

8.1.1 Introduction

Infrastructure tests including the online infrastructure, ROS, Level1, EB, and EF were performed with tdaq-01-06-02 with the aim of verifying its principle functionality on a large scale before other sub-systems and the integrated system with trigger algorithms were run. Given the overall TDAQ installation and commissioning schedule, DAQ specific tests were not planned for the LST but not excluded. Basic state transition timing measurements were done. During the process of getting the TDAQ software system up and running on a large scale, many problem situations occurred. Studies of these situations are reported in the chapter 'Problems found and remarks to Fault Tolerance'.

Initially, tests were performed on an installation of SLC4 on 4 nodes where the Linux system was installed as 32 bit architecture with the native SLC4 release of tdaq-01-06-02. When entering running state, the SFis systematically crashed and the issue was reported. No further tests were performed on this platform and the SCL3 emulation mode on SCL4 nodes was used from then on for all the tests.

A LST06 specific setup segment was used, where the infrastructure processes are distributed over a set of 10 nodes. Distributed rdb_servers were residing on nodes where only one controller was running (this is the way PartitionMaker distributes them).

From 100 up to 600 nodes were used for the infrastructure tests without algorithms, using only tdaq-01-06-02. A partition type was prepared which included ROS, LVL2, EB, EF segments. It was possible to bring the partition up to running state and keep it running for some time. Problems were in many parts of the system. Most prominent were the problems with the setup component and the problems with the stop and the unconfigure transition, which concerned the LVL2, EF and EB processes. While - with a number of retries and a great deal of patience - it was possible to setup, configure and start the partition, the stop transition normally failed. Usually DFM, SFIs or SFOs died. The situation could not be recovered (numerous tries) and an orderly shutdown was not possible. The infrastructure needed to be removed with ipc_rm .

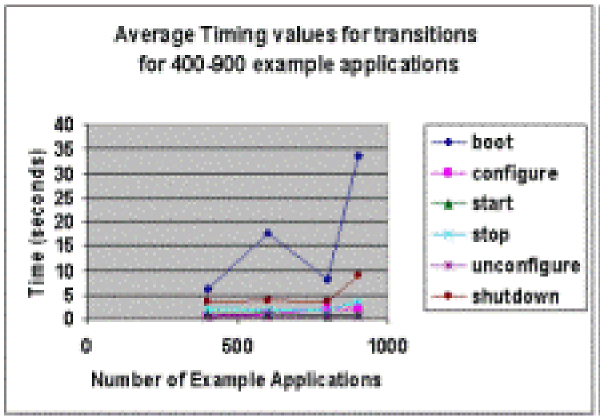
As for the running phase, many variants to the HLT and EB parameter settings were probed and finally a reasonably stable run was achieved. Dataflow timing and throughput measurements were not in the scope of the tests, given that there were no algorithms running and there was no separate control and dataflow network. These test allowed to adjust the system parameters and the time out parameters to the size of the system and the specifics of the farm.

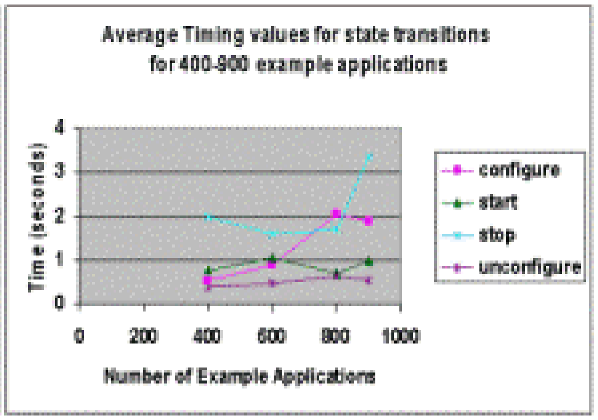
Being in the forefront before other tests were performed at the same scale, many problems concerning the infrastructure could be identified before more complex tests including trigger algorithms and database access would be performed. The problems were logged in the LST06 elog, reported to the developer and screen shots were taken. Some bugs were fixed quickly, and a patch was applied, a workaround was found for some others. The remaining and more complex points are discussed in the chapter 'Problems found and remarks to Fault Tolerance'.

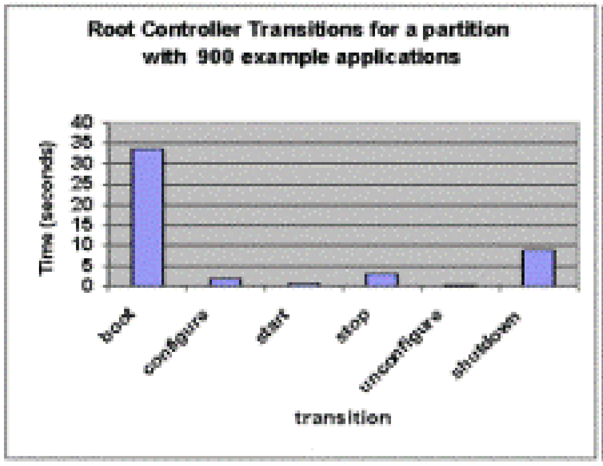
8.1.2 State Transition Timing Measurements of the Infrastructure

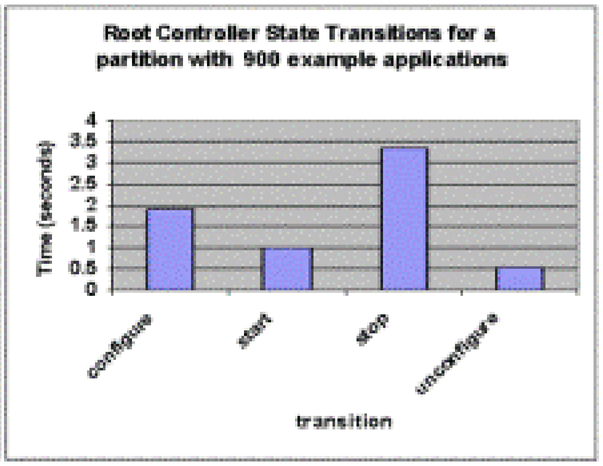
Run Control State Transitions

The partitions were produced with PartitionMaker, which consisted of a Segment with Controllers and example applications. The example applications are controlled applications and respond to the state transition request of the controller and publish their state but do not perform any other time consuming operation. The measurements show the overhead which is generated by the control infrastructure. Timing measurements have been recorded for a set of partitions including 400, 600, 800 and 900 controlled example applications. The generation of a such a partition, in which the ROS application is replaced by the example application for ∼400 nodes took ∼1/2 hour with PartitionMaker and even more time for the larger ones. Unfortunately, even having searched for and explored numerous 'tricks', it was not possible to generate clean partitions with PartitionMaker which contained more than 1 example application per node.









Graph 1 shows the timing values of the transitions for the 4 partitions, averaged over a couple of runs (2-6, depending on the partition). Graph 2 shows the same measurements, without the boot and shutdown transition for better visibility of the values. The boot transition, where all the participating processes are bootet, is network dependant and no stable values were measured. The trend of the transitions configure, start, stop and unconfigure is very flat, which corresponds to the measurements taken during the Large Scale Tests 2006 [Ref1]. The difference in values for the different transitions correspond to the number of internal transisions, as explained in Chapter 2, AtlasTDAQOperationalSequence. The number of internal transitions are for configure:2; start:2; stop:5; unconfigure:2. It must be noted that configuration database read and publishing operations occur during the configure step which results in a higher timing value compared to the other transitions. .average value for the partition controlling 900 example applications are shown in graph3 and the same in graph 4 without the boot and the shutdown transition. One can see that an internal state transition amounts to 0.5 seconds. This good value obtained during the 2005 tests, where 0,7 s was obtained for an internal transition of 1000 controlled example applications, has even slightly improved.

Setup timing:

During the 2005 Large Scale tests, the setup component was already reported to be problematic, with a value of 16 minutes to set up a partition with 2000 example applications on 700 nodes [Ref1].
This year, the time to set up a partition on 900 nodes was of the order of 20 minutes when having the pmg_agents already running. In addition, the setup component gave problems when running on a large scale and the details are reported in the following chapter.

8.1.3 Problems found and remarks to Fault Tolerance

During the process of DAQ/HLT system preparation (using tdaq-01-06-02) during the tests introduced at the beginning of this chapter, numerous issues came up and situations occurred in which observations about the fault tolerance behaviour of the system were made. Those are reported here.

Setup

The setup is a vital component and the first one when running the system. Therefore it is most prominent and possible problems affect everyone who wants to run the TDAQ system. The problems reported here do not matter really on small scale which explains why they had not been seen before. Even the system size which is used in point 1 currently is 'small' compared to the system used here and the timing does not increase linearly with scale but much steeper. </o:p>

The time to start the pmg_agents was clearly very long, the setup starting with pmg_agents running on a system with 900 nodes took about 20 minutes. As the agents will be started by the system in the future, primarily the setup time starting with running pmg_agents should be investigated and improved further. The time to subsequently execute the test on the pmg_agents, an action which is performed by setup each time the partition is launched, was also far too long. Following dedicated tests by the developer, the setup time could be reduced to a reasonable value towards the very end of the LST06 testing period by using a new knowledge base.

After this modification, the time merely for the hardware test was still 7 minutes for 900 agents. This HW tests was then normally skipped via an option in setup_daq. It should be noted that during hundreds of test runs on the large scale the hardware test never failed, once the pmg_agent test was passed successfully.

On a large system, it can happen that the time out is reached before the end of the tests, even when set to

a large value. The retry option currently initiates a complete retry, meaning it tests once more all agents which had already been tested, instead of continuing. However, an (unintended) feature made that, once the timeout was reached, the setup still accepted the outstanding test status information from the agents which had not been handled yet and were still in the queue because setup used 100% CPU. This was helpful and should be kept. It was possible to enable an individual agent, but the result was not propagated up the chain and therefore this feature is not of use. One had to initiate yet another test on the higher level. In this release version one had to test the higher and highest parent component, which in turn then tests all its children again, which once more costs time. In simple cases it is possible to re/setup one component and then 'ignore and continue' directly. But this options was only available once the time out on retry had expired.
A clear policy with a number of options to retry partially or globally, and all the options available at all phases of the setup is desirable. It would be useful to have a "remove all absent nodes" feature. Clicking through nodes individually becomes time consuming when the partitions get very large. A "retry all nodes not in initial" feature would be good.
If the time out is chosen to be larger than the estimated time to set the system up, and then for example a node is unavailable then one has to wait until the end of the time out. For a retry one would have to wait the full time out time (i.e. 20 minutes for 900 nodes) to be able to continue if one wants to 'ignore' the faulty node (in a case where it does not host a vital component but for example a PT or L2PU This is unfortunately the reason why the logger was often disabled. The setup test for the logger always gave an error even that it was started correctly. The MySQL server was running on a dedicated node, but the test was searching for a standard server node. The nodes assignment is apparently not easy to change in the system. This meant that even when the system status was otherwise correct, one would have to wait for the time-out to be expired. For all items controlled by setup, it would be good to be given the choice of a quick retry, acting only on those elements which had failed before, and with a timeout which can be different from the first one to avoid having once more to wait for many minutes.

When the infrastructure is started, the root controller was always red, also other components were sometimes red, and switched later to green. This is confusing. The display should not reflect the internal response of a test while the internal retries are still going on but rather display a global state reflecting that the test is on-going (i.e. blue), in a different colour like for the other tests. The font size should adjust to the number of nodes to be displayed. It can be much smaller, this could avoid a lot of scrolling. The information could also be presented in several rows.

A useful feature, provided as an option, would be the following: If a test on a node fails, then it should be checked if to this node only non-vital components (i.e. EFD, PT) are assigned. Up to a certain percentage, which should be a configurable parameter, setup should only give a warning message, remove or disable those processes from the partition and continue with 'success'.

In one configuration, without multiple rdb_servers, one of the pmg_agents was not started and not in the list of agents to be started. It was traced down to a problem in the setup test concerning template controllers in non-HLT segments. This problem occurred rarely and not normally when setting the multiple rdb servers in PartitionMaker because of the way the controllers are distributed (there the pmg_agent was anyway started for another process).
When using nodes which have not been used before then there is a pmg problem trying to create the /tmp/pmg file from 2 applications at the same time. The output file reports failed creating '/tmp/pmg directory' and the err file reports 'file not found'. The second time the partitions is started on the same set of hosts this error does not occur because /tmp/pmg exists.

Setup did not start pmg_agents of segments with template Applications other than HLT; As the controllers are also template this only sometimes worked by chance namely when another component was also assigned to that node.

Proposal

The setup is a vital component and the first one when running the system. Therefore it is most prominent and possible problems affect everyone. As feedback from the tests it is suggested to revisit the user interface functionality.
When running a big system, the following factors are important:
1) time it takes to start the system
2) options to continue in case of a time-out on the start of the pmg_agents, the HW checking or the start of the infrastructure; time outs optionally to be set different for different phases (1st try, 2nd try,..)
3) recovery, retry, and continue options during all data-taking phases, not only at the occurrence of the timeout, optionally automatic, configurable, and manually.
4) automatic propagation of a recovered component test status up the setup component hierarchy tree
5) clear presentation of the status of the involved components as a result of the test which setup performs

Applications crashing at Boot

During the boot transition on large scale, it happened rather frequently, that some processes didn't boot. This was either due to the occurrence of a time-out, to a failing node but also to a problem with the application.
In a number of cases, a single process would die (SIGV, as reported above). In the case that this is a non-vital process (a PT or L2PU), a retry is performed and the process starts correctly. Tests with an integrated partition on 500 nodes with tdaq-01-06-02, multiple rdb servers and dedicated setup segment, release taken from afs, were investigated. In this situation all the nodes involved were up and responding well. A segmentation fault on one PT (problem in glibc, segmentation fault in libc.co.6) out of hundreds in the partition (2 PTs per node) would occur. At the next boot the same application started without problems but the same type of error occurred in one L2PU out of hundred (2 L2PUs per node). More details are reported in the elog. This underlines the need and importance of correctly designed and implemented fault tolerance on a large system.
When running from afs, one of the PTs reported error on opening the shared library liboh.so. This is most likely due to the simultaneous access to the same file by a large number of PTs. Therefore the local installation was used later on for the test.

IGUI

When receiving a large amount of messages, the IGUI gave problems in the RC panel (separation between the 2 sides of the panel could not be re-established after having moved to single view) and in the detached MRS panel. The latter was blocked until long after the partition and the IGUI were terminated and finally he process had to be killed manually.

When stopping a large partition which is in fault state, the IGUI took 100% cpu when running on one of the lxb nodes. The same was true when running the IGUI locally on the (local and not high performant) desktop. When running the IGUI on an otherwise unused and very high performant node (backup node for the log service) then this run well.

Stop, Shutdown and Cleanup

The stop transition was critical and normally did not finish. As a consequence, the only possible command then was shutdown. This command had to be given many times, finally the whole infrastructure needed

to be removed. All this took considerable time. Often processes were still hanging.
An ipc_rm on the entire partition, and subsequent ipc_cln did not clean the field: when starting the same partition up again, PT error messages appeared after having started the infrastructure after the boot.
The PTs didn't seem to respond to the global ipc_rm command and re-connected. Only when removing the initial partition and with it the pmg_agents this effect disappeared.create a sharedHeap per partition: if EF processes don't stop gracefully, then this file it is left over on /tmp. As the name includes the partition name, there could be many of these large files left. A general cleanup should include removing this file.

SFIs died systematically after the stop/start sequence.

Run Control

For all items controlled by run control, it would be good to be given the choice of a quick retry, acting only on those elements which had failed before, and with a timeout which can be different from the first one to avoid having once more to wait for many minutes.
At boot, it would happen that one or a few processes did not boot in the timeouts time. It also happend that a process failed and could not be restarted, so needed to be removed. A selective retry was possible, but the change of state was not seen automatically by its controller, and one had to continue the retry up the chain by hand. An automatic way of having the change of state propagating up, or a retry top down would be helpful.

Like for setup, a useful feature, provided as an option, would be the following: If a test on a node fails, then it should be checked if to this node only non-vital components (i.e. EFD, PT) are assigned. Up to a certain percentage, which should be a configurable parameter, run control should only give a warning message, remove or disable those processes from the partition and continue with 'success'. In addition, dependancies should be taken into acccount. If one EFD fails and is taken out, the corresponding PTs should also be taken out automatically.

The number of retries for processes to be started after failure or when having died should be a parameter which one can set by the user.

DVS

The DVS crashed when more than 100 nodes were in use. The debug version did not have this problem and was used. When running from afs, cmtconfig had to be set to point to slc4.

Log service

The log service was available and installed into a dedicated node. It was apparently not possible in setup to make the test for the logger pointing to this node. Therefore, setup failed each time when the logger was included. As a consequence, one had to wait for the timeout and the appropriate options to continue to appear. To avoid this long waiting (up to 20 minutes, see description above about setup), the log service was unfortunately not used systematically.

Messages

Many applications send far too many messages. In a large system, these messages are multiplied by a factor 100 or 1000 thus flooding the system and making it very hard to 'see' the real problems. Only the really important messages should be send and they could be grouped by a centralized server.

Templated Applications

The introduction of templates for applications and for run control was very useful. Problems however are the very long name which has been chosen. Normally it does not fit into the window, or as a result, other information which follows in the same line is not visible. This is not only true for the IGUI but also in other GUIs and tables. On a large system, lots of scrolling right to left is involved in order to understand the full picture, which can be unnecessarily tiering and annoying. I suggest that this problem should be changed at the source level, changing the name itself. It could otherwise be solved at the GUI level with the disadvantage that each GUI would have to treat it separately, make a special case only for template applications, and have the displayed name differ from the original one (possibly displayed differently in different GUIs) and thus loose the authenticity.

It would be very helpful if this change could be done. For example, instead of adding 'TemplateApplication' to the name, it could be very much shortened, like for example 'TA'. Other solutions could be equally good.

Tools

The Farm tools PartitionMaker PartitionRunner were under development during the tests and not part of the release . Although partially dependant of each other, the versions did not work well together.

The farmtools did not work as expected. </p:colorscheme> </span> synchronization was done and claimed Ok but verification showed that this was incorrect. The Hardware file production up to O(100) nodes was working but failed when going larger. The underlying worm technology seemed to have hit serious problems. One of the consequences was that s ome farm tool processes were left hanging for days taking 100 % CPU time. This is hard to find on a cluster of 1000 nodes and when the tool to be used was the same that had originally caused the problem. Another effect was that the farmtools left large files on /tmp with names build of random numbers. Those are hard to check for and to remove in bulk operations. In addition, Cern IT reported attacks to the Cern security server nodes.

The * PartitionMaker* has the python pointer hardcoded. This gave a problem when having to point to a particular version of python. Many features which had to be or would have had to be used at LST06 were not available in the version of PartitioMaker at the time. For some of them, a workaround was available. Details will be supplied for the upcoming review of the PartitionMaker.

Opmon was used for the automatic recording of the state transitions. This worked well. It was experienced that it would be desirable to add a time stamp to each of the transition as an additional help to trace back a situation.

Major updates :
– Main.DorisBurckhart - 2 March 2007

Complete: ☐
Responsible:
Author:
Contributors:
Last significantly modified by:
Not yet reviewed

Chapter 9

Monitoring Tests

9.1 IS, EMON, and Gatherer tests

9.2 Introduction

9.3 Gatherer

The Gatherer application is designed to gatherer monitoring information such as histograms, scalars, and vectors from the Information Service, IS. In the final ATLAS running there will be many nodes processing the data. Each node can then produce monitoring information which is then published on the IS. The Gatherer will make a sum of this information and publish it back to IS. The published information can then be used by other monitoring applications which would need the full statistics, ie the Data Quality Monitoring Framework(DQMF) or the Monitoring Data Archiver (MDA). Only histogramming functionality was tested here (other functionality not available in this release).

9.3.1 Program of Tests for Gatherer

The original plan for the LST tests for the Gatherer was two parts:

1 Get some timing measurements while changing the parameters
   - the number of histogram per node, N
   - the number of bins in each histograms, B
   - the type of histograms, ie 1d, 2d, 1p, D
   - the type of bins, ie float, double, T

2 Use this timing information to determine an optimal configuration of Gatherers to be used to sum histograms produced by the L2PU and PT algorithms.

We only however managed to finish 1) since there was no oppertunity to include the Gatherer in the partitions produced for doing L2 and EF tests. This was mostly due to the timing of the tests.

9.3.2 Configuration of Tests

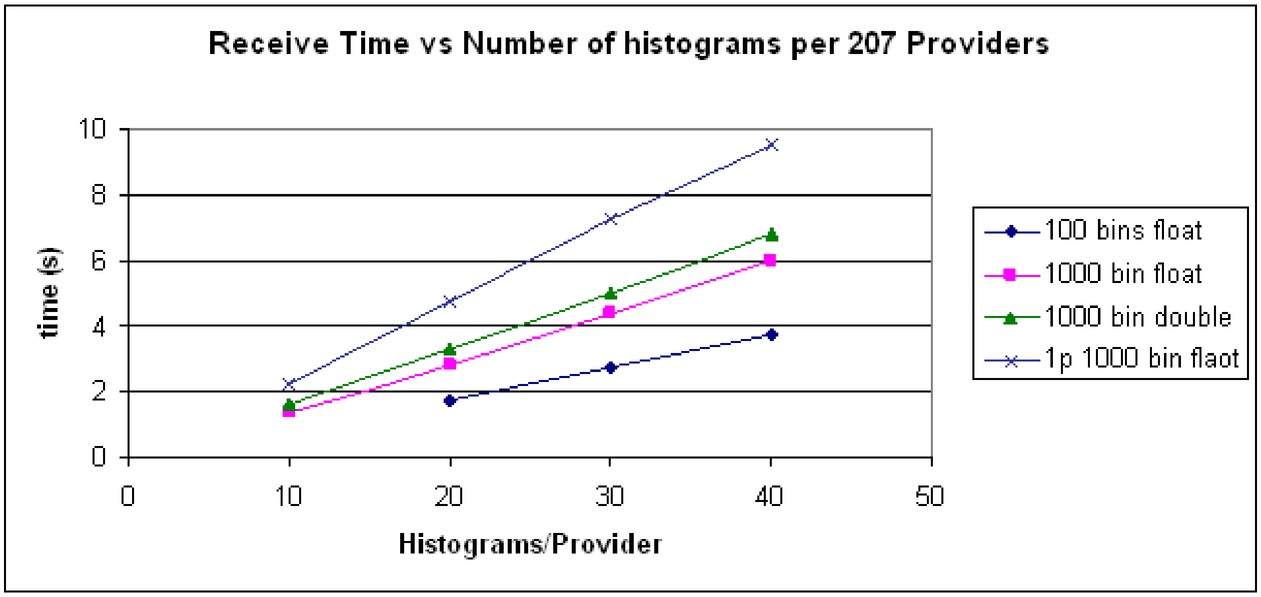
The Gatherer timing tests were done on 207 nodes while changing the parameters listed above under 1). The machine on which the Gatherer was running was a 2.8 GHz machine with 2.0 GB of memory. The network speed was 10 GB/s. The number and type of histograms indicated were produced from a given node and send to a server which was running on an independent machine. The Gatherer was then run on a seperate machine from the server. The Gatherer would then sum over all providers, ie nodes, and produce a summary for each histogram. ie for the 207 nodes we have 207xN histograms and end up with N summary histograms.

The histograms were published on the server only once and not updated. The Gatherer retrieves the histograms off of the server every 10 seconds and makes a new summary. The retrieval of the histograms from the server is done in pull mode.

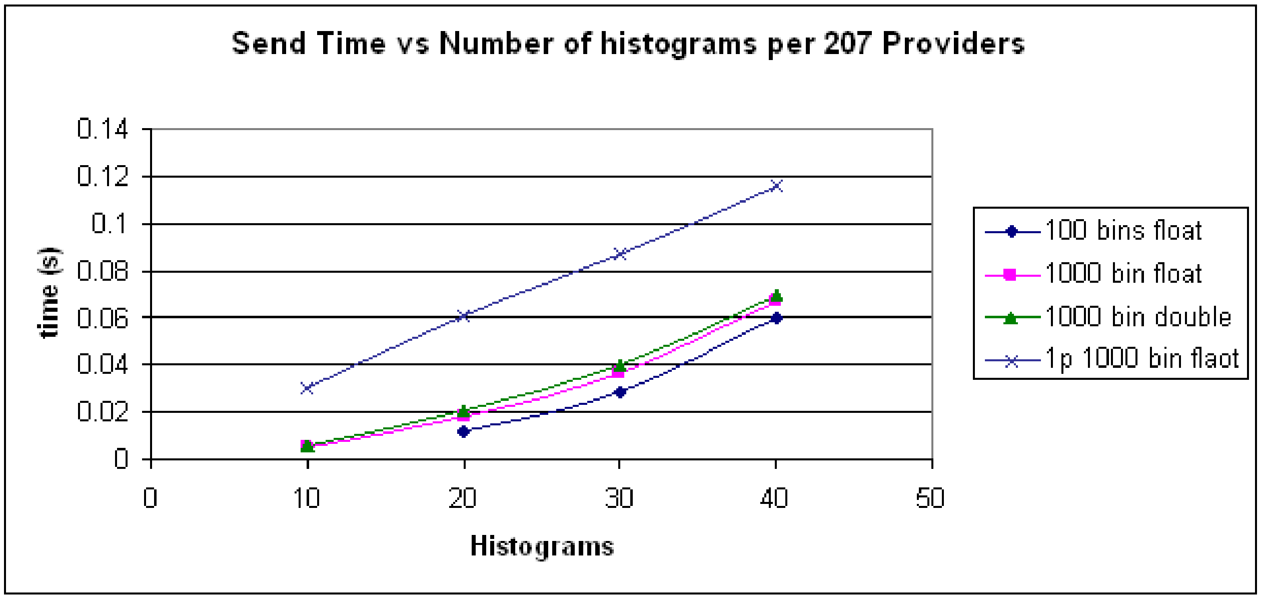
9.3.3 Timing Results

We see that the timing results looked reasonable and are mostly linear. The profile histogram timing was a bit higher than expected due to warning messages from ROOT which will be suppressed in future releases.

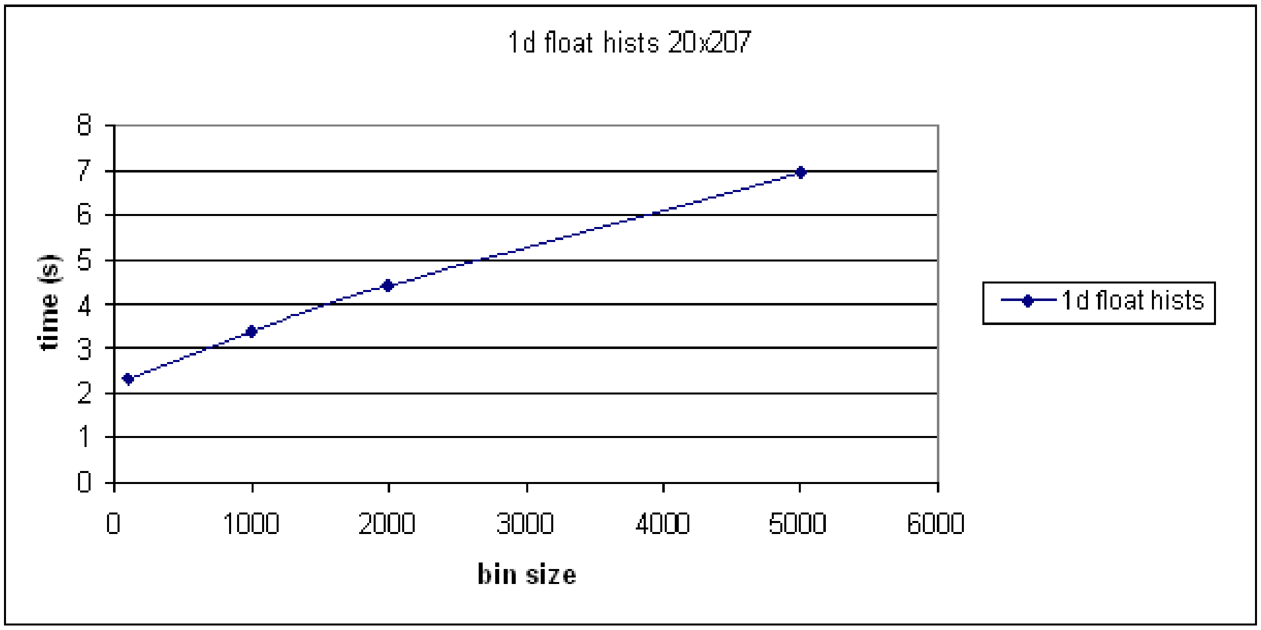
This plot shows the time to receive the histograms in the various configurations. We see a nice linear distribution and we also see the timing increase with histogram size as expected.



The following plot shows the time it took to send the summary histograms to the server. The distribution is slightly less linear than the receive time.



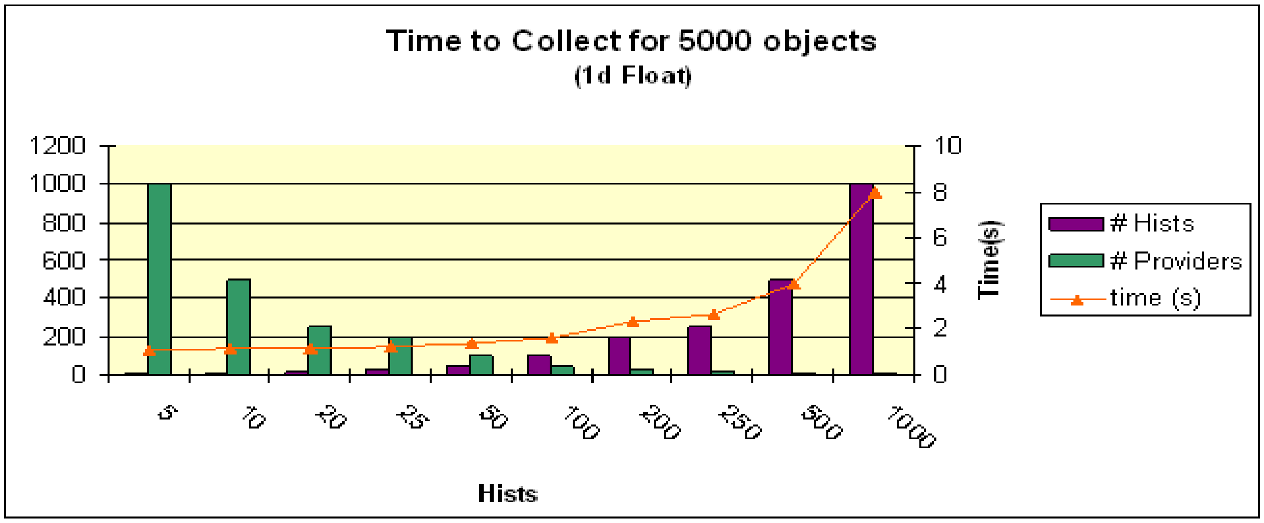
The following plot shows the time to Receive 20x207=4140 histograms 1d float histograms with various bin sizes. As expected we see a nice linear distribution.



Timing Results - Histograms vs Providers

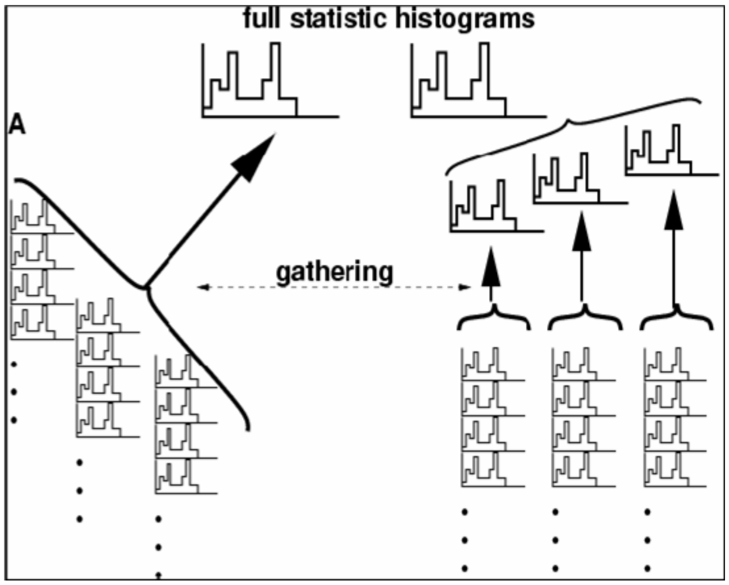
The Plot below was done during the Small Scale tests in preparation for the LST. This plot shows that for a total of 5000 histograms the Gatherer receiving time varies significantly as a function of number of histograms/provider; For the 5000 objects the timing varies between 1-8sec. We observe that the timing is much less if one loops over a small number of histograms but over many providers. This is because for a given histogram name the Gatherer produces a histogram iterator. The time for producing these iterators is significant with respect to receiving times. So the overhead for producing each iterator adds to the total receive time. That is why for the 1000 histogram 5 provider scenario we see such a large timing. In this case 1000 iterators are produced.

In a scenario where we have a large number of histograms, this tells us that it's probably best not to assign a Gatherer application to a given subFarm, where the number of nodes would be of order 10 and histogram number could be large. It's more efficient to collect some subset of histograms over all nodes, ie all LAr histograms from a given algorithm.



9.3.4 Summary and Conclusions

The Gatherer timing results show good linear performance as a function of bin size, histogram type, histogram bin type, and number of histograms. The result of tests done over varying histogram and provider numbers shows the best configuration for the Gatherer is to run over a large number of providers and a smaller number of histograms. One should be aware of the type of histograms being summed in order to keep the timing in an appropriate interval. When necessary more Gatherers can be added into a cascade of Gatherers to keep within the allotted time (See diagram below).



9.4 Information Service (IS) tests

The Information Service (IS) component is used to share user defined information between applications in the Trigger/DAQ system. It is responsible for a large number and potentially high rate of the monitoring data exchange. Last year a comprehensive set of tests have been performed for the IS and results of these tests are described in the test report document. Recently a number of new functionalities have been added to the IS. The main aim of the IS tests this year was to investigate a possible impact of these new features to the IS performance and scalability. In order to get this information some of the last year tests have been repeated with the new IS implementation.

9.4.1 Test description

Test applications

There are three applications which have been used for the IS tests:

- is_server - this application implements the IS repository functionality.
- is_lst_provider - this application is a configurable test for the IS information provider.
- is_test_receiver - this application implements a configurable test for the IS information receiver.
- is_cmd - this application can be used to send commands to the IS infomation providers

All of them are part of the DAQ/HLT-I release. One can run them with the #8216;-h#8217; flag to see their usage.

Hardware description

Slightly more then 500 computers have been used for the IS tests. One dedicated computer (dual PIV 2.8 GHz with 4 GB memory) was used to run a single IS server application. Fifteen machines (dual PIV 2.8 GHz with 2 GB memory) have been used to run 1, 3, 5, 10 and 15 IS information receivers with only one receiver per machine. Another 500 machines were used to run from 1 to 6 IS information providers on each of them simultaneously. The slowest section of the network connection between those computers was fast Ethernet.

Tests description

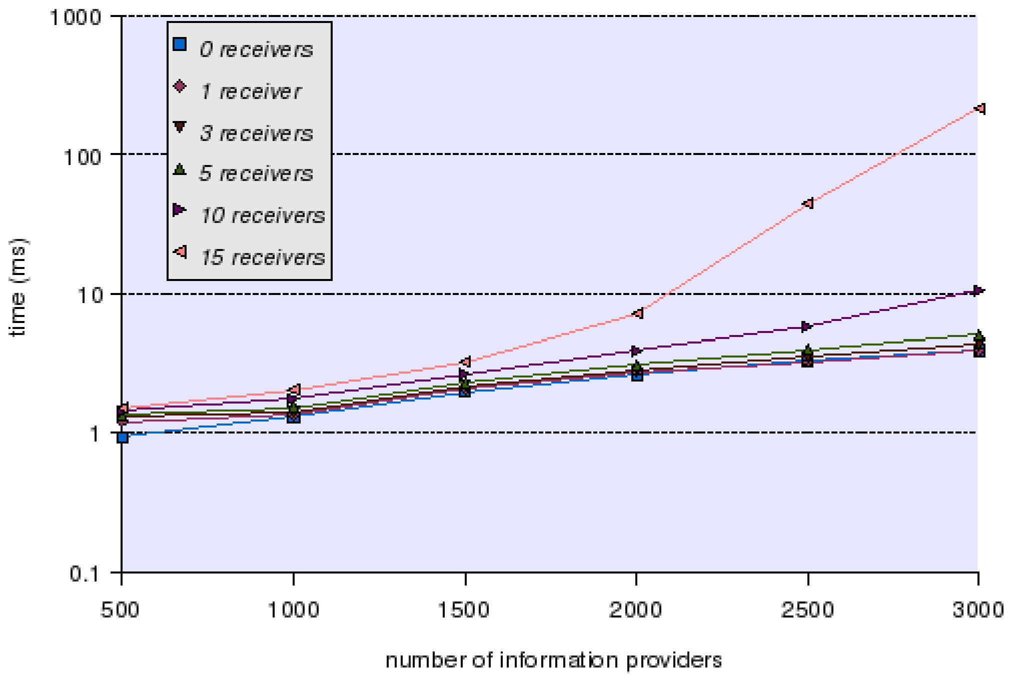
Initially 6 is_lst_provider applications have been started on 500 computers which results in 3000 infomation providers in total. Each of those applications created one information item in the given IS server. Then several test series with different number of information receivers have been performed. Here is a descriptions of the steps which have been done for an individual test serie:

1. N is_test_receiver applications have been started on the 15 dedicated machines, one per machine. Each of them subscribed for all information in the given IS server. Number N was set to 1, 3, 5, 10 and 15 for different test series.
2. is_cmd application was used to send the "start" command to M infomation providers. Each information provider started to update its own infomation object once per second as soon as it received this command. Every infomation provider has updated its infomation 100 times. Number M varies from 500 to 3000 with step 500.
3. All information receiver applications have been stopped.

9.4.2 Tests Results

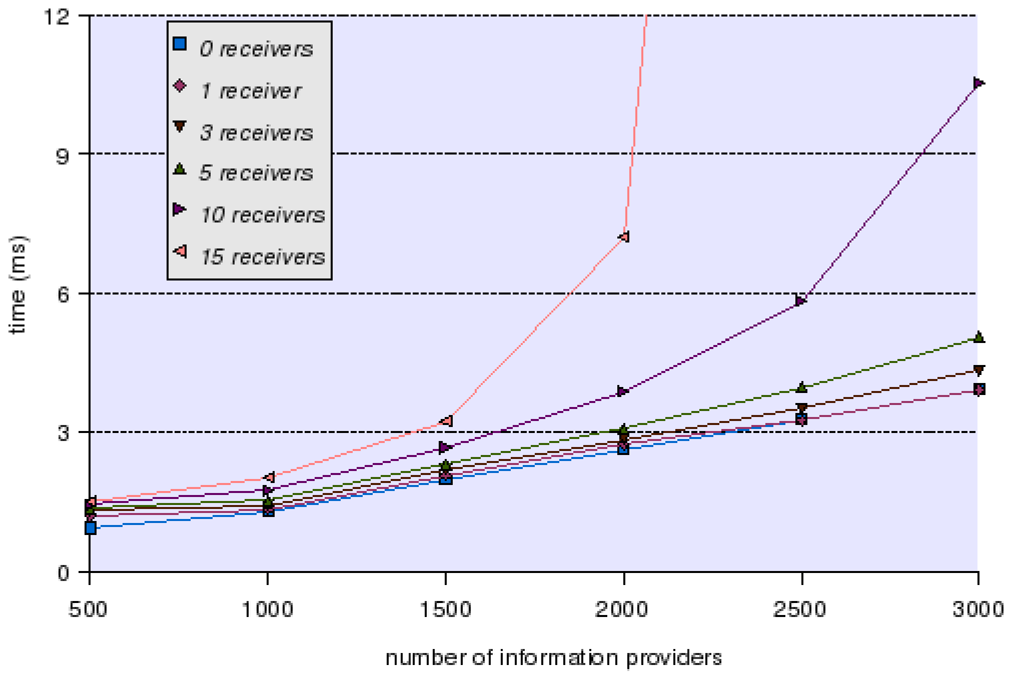
Two figures below show the mean time for one information update as function of a number of concurrent information providers and receivers. The first plot has a logarithmic scale in order to present the complete results.

Mean value for one information update (logarithmic scale)



The following figure shows the same plot as the previous one but with the normal scale along Y axis in order to present the behaviour of curves in more details.

Mean value for one information update



9.4.3 Summary and Conclusions

Presented results are very similar to the ones, which have been obtained in July 2005 (during the last LST) using very similar hardware configuration. This demonstrates that a number of new functionalities, which have been provided for the IS, do not made negative impact to the IS scalability and performance.

9.5 Event Monitoring (EMon) Tests

The Event Monitoring (EMon) component can be used to sample events from different points in the data flow chain and distribute them to monitoring applications written by users. Using a code framework provided by EMon, users can contribute their own applications in order to sample events from a certain sampling address, all sampled events matching certain selection criteria. Such a combination of event sampling code, sampling address and selection criteria is called an event channel.

In order to prevent a bottleneck in event channels when large numbers of monitoring tasks ask for events from the same channel, a tree-based approach finds application. Only a single root task is directly attached to a sampling channel, all the remaining task are attached to this root task in a tree-based manner. A set of tasks arranged in such a tree, requesting events from a certain channel is called a Monitoring tree.

A major goal of the 2006 tests is to prove the scalability of this approach. Having performed these tests already during the LST phase in 2005, a bug has been found which - in the version used during the 2005 tests - prevented actual data from being transferred from root monitoring tasks and their children. As this bug has been fixed, the main purpose of the 2006 tests was to repeat the measurements of the 2005 tests with a current version of EMon.

9.5.1 Test description

The following three applications have been used for the EMon tests:

- emon_conductor - a central application managing EMon task subscriptions and sampler registrations and controlling the tree topology of tasks
- emon_push_sampler - an application generating random events and pushing them to subscribers
- emon_task - an application requesting events from an event sampler, possibly forwarding them to any attached child tasks

About 50 computers have been used for the EMon tests. A single emon_conductor and a single emon_push_sampler where running on two dedicated machines. A variable number of up to 50 emon_task applications has been started on the remaining machines, all of them requesting events from the same event channel in the emon_push_sampler.

In order to measure the effect of different tree topologies, an additional parameter k was used to specify the tree type of the Monitoring tree. In case of k=1 , tasks have been arranged as a unary list, in case k=2 as a binary tree. In case of k=5 each task had up to 5 child monitoring task it needed to forward events to.

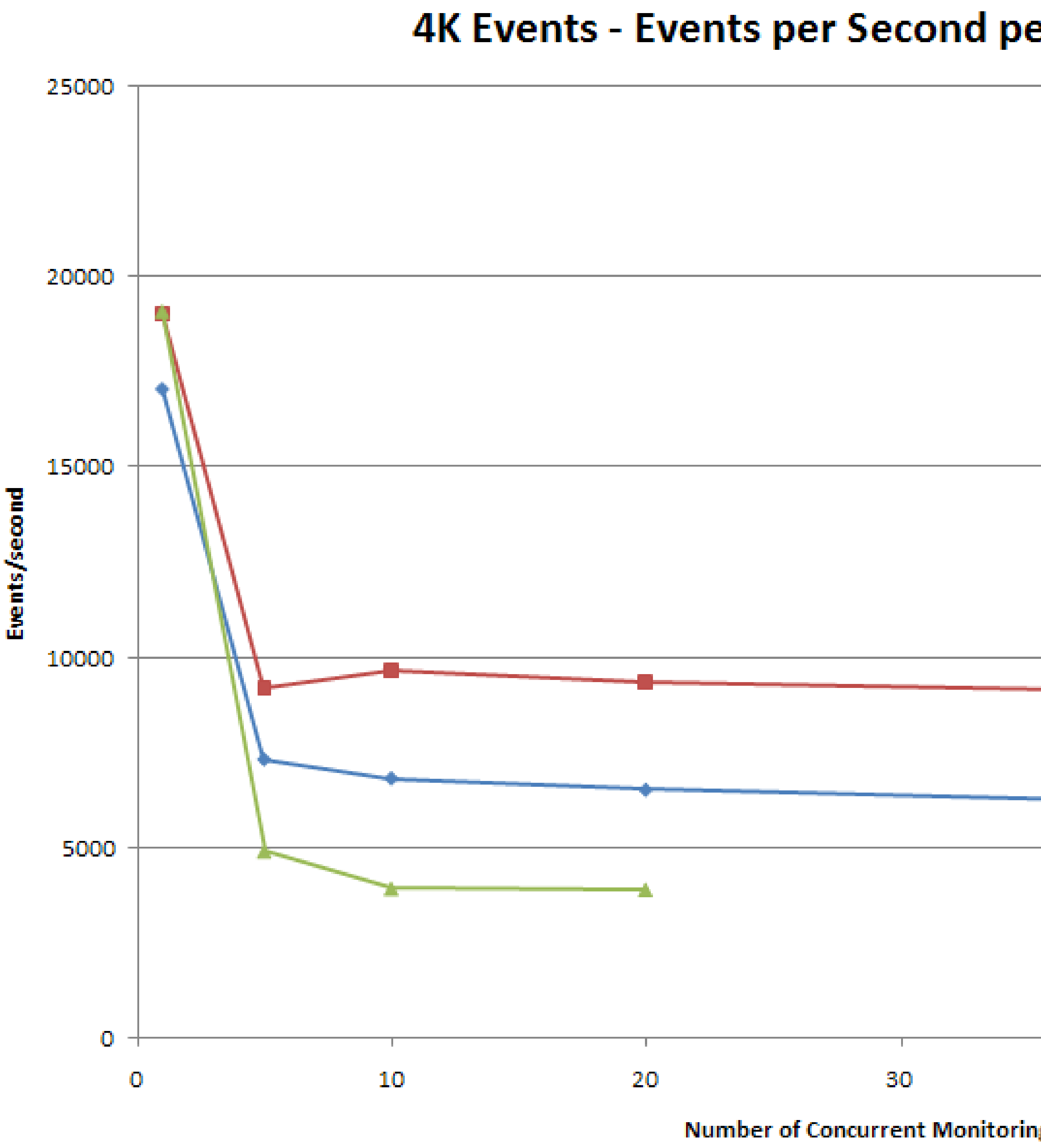
In order to fully include the effect of event forwarding in the measurements, only the rate at which the bottom most (so-called leaf) task of a Monitoring tree could process events has been recorded. This naturally is the worst-case event rate of a Monitoring tree, all parent tasks necessarily have a higher event rate.

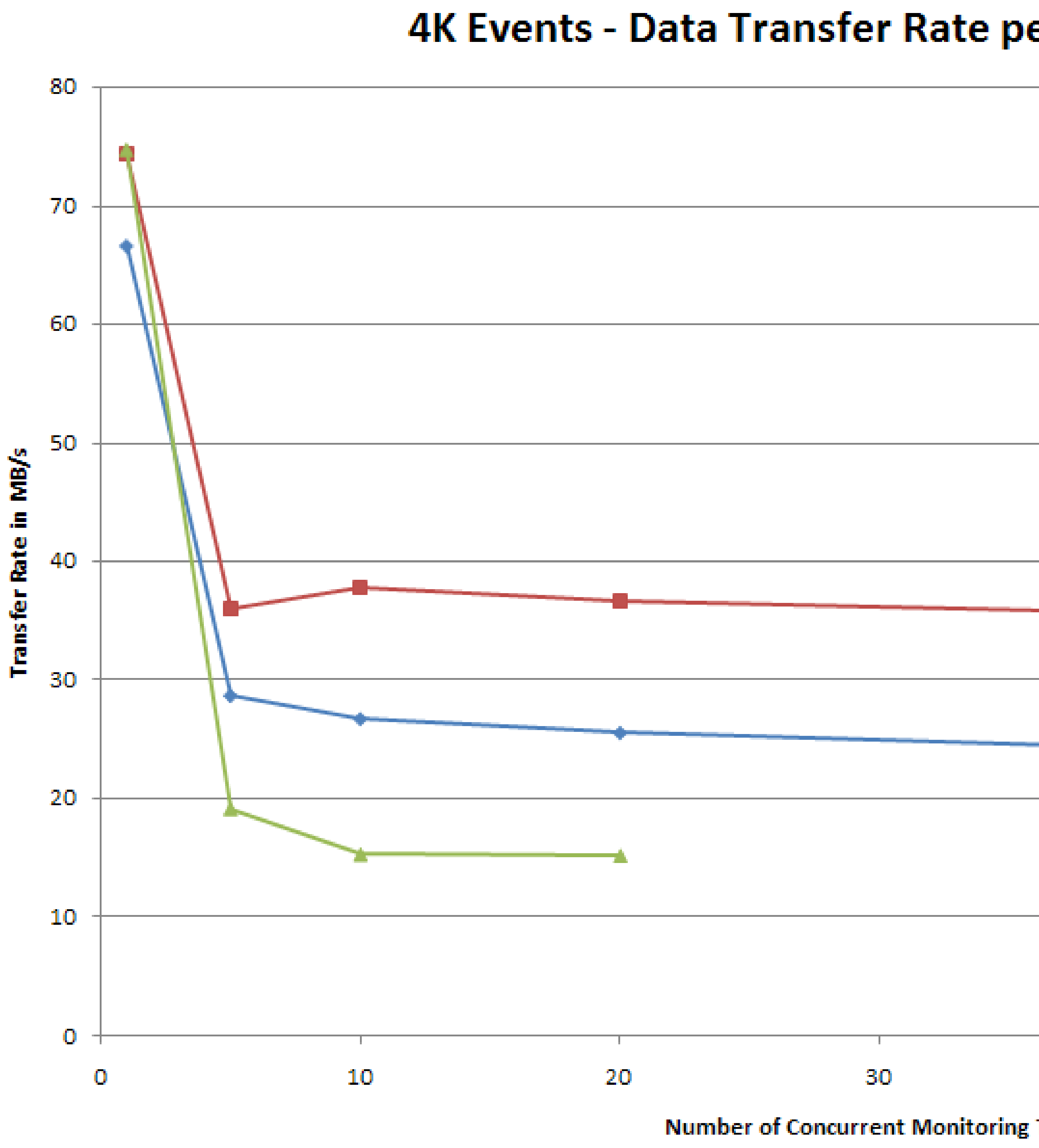
As the EMon component can be used to sample events from several different points in the data flow chain of the final experiment, the size of the events communicated via the EMon framework varies in a range between 4K and 2 MB. Accordingly both ends of this event size range have been tested during the 2006 tests.

9.5.2 Test Results

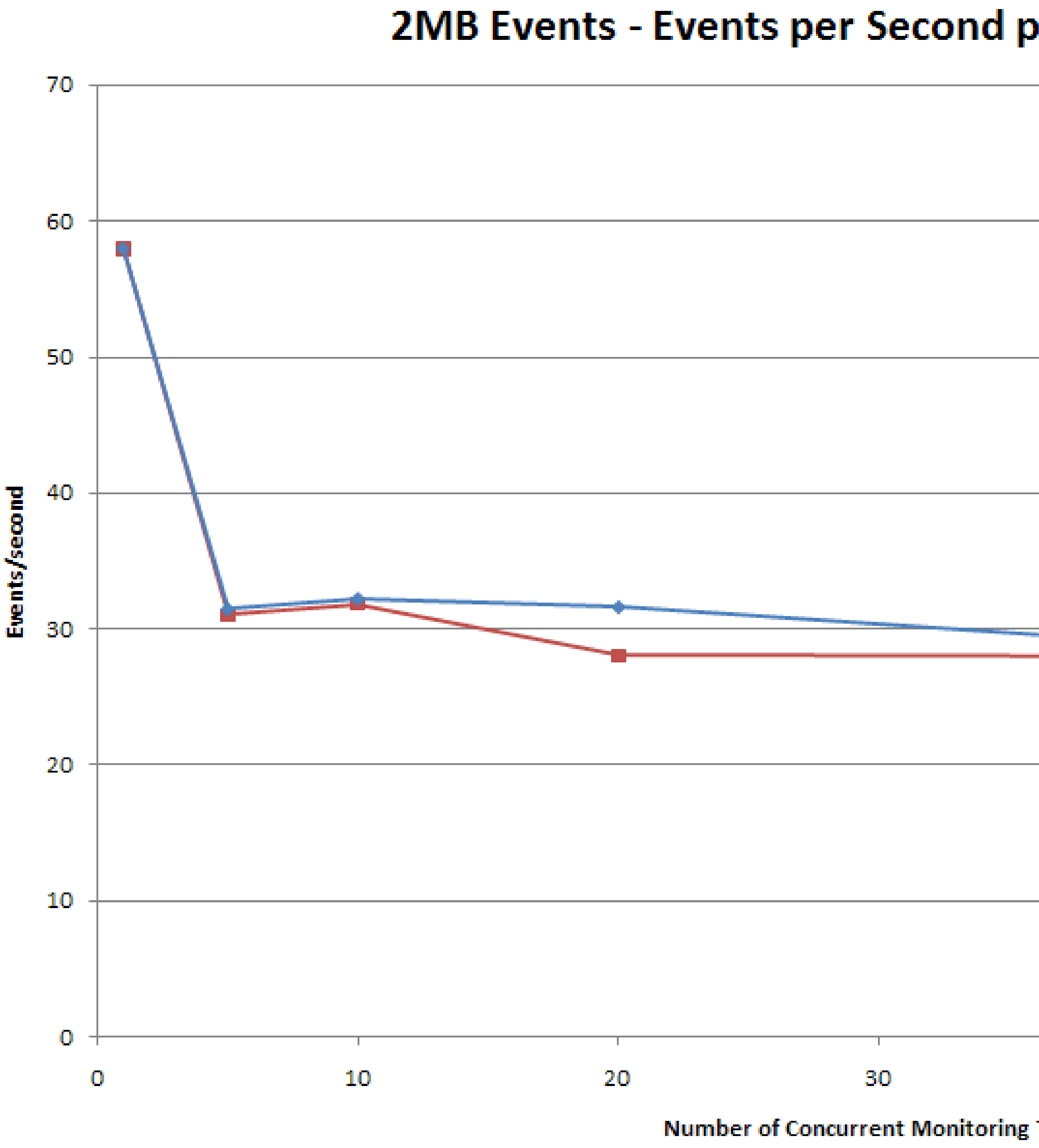
These figures show the rate at which the leaf monitoring task of a task could receive events in events/second for an event size of 4K and for different values of the tree type k. There is a remarkable difference in the event rate between a tree consisting of a single task and a tree consisting of 5 tasks. A single task can receive 4 K events at a rate of about 19,000 per second. In a k=1 tree consisting of 5 tasks, this value drops to roughly 50 % or 9,500 events per second. This drop likely results from the overhead involved with the forwarding thread and the process of copying events for child tasks. This value of 9,500 events per second remains however constant with an increasing number of tasks connected to the tree. As in cases k>1 each task has to forward each event to several child tasks, this event rate is naturally lower, the higher the value of k is. With values of k=2 and k=5, the event rate again stays constant when increasing the number of concurrent tasks above a value of 5. The performance loss of k=2 when compared to k=1 is roughly 20 %. The performance loss of k=5 compared to k=1 is roughly 50 %.

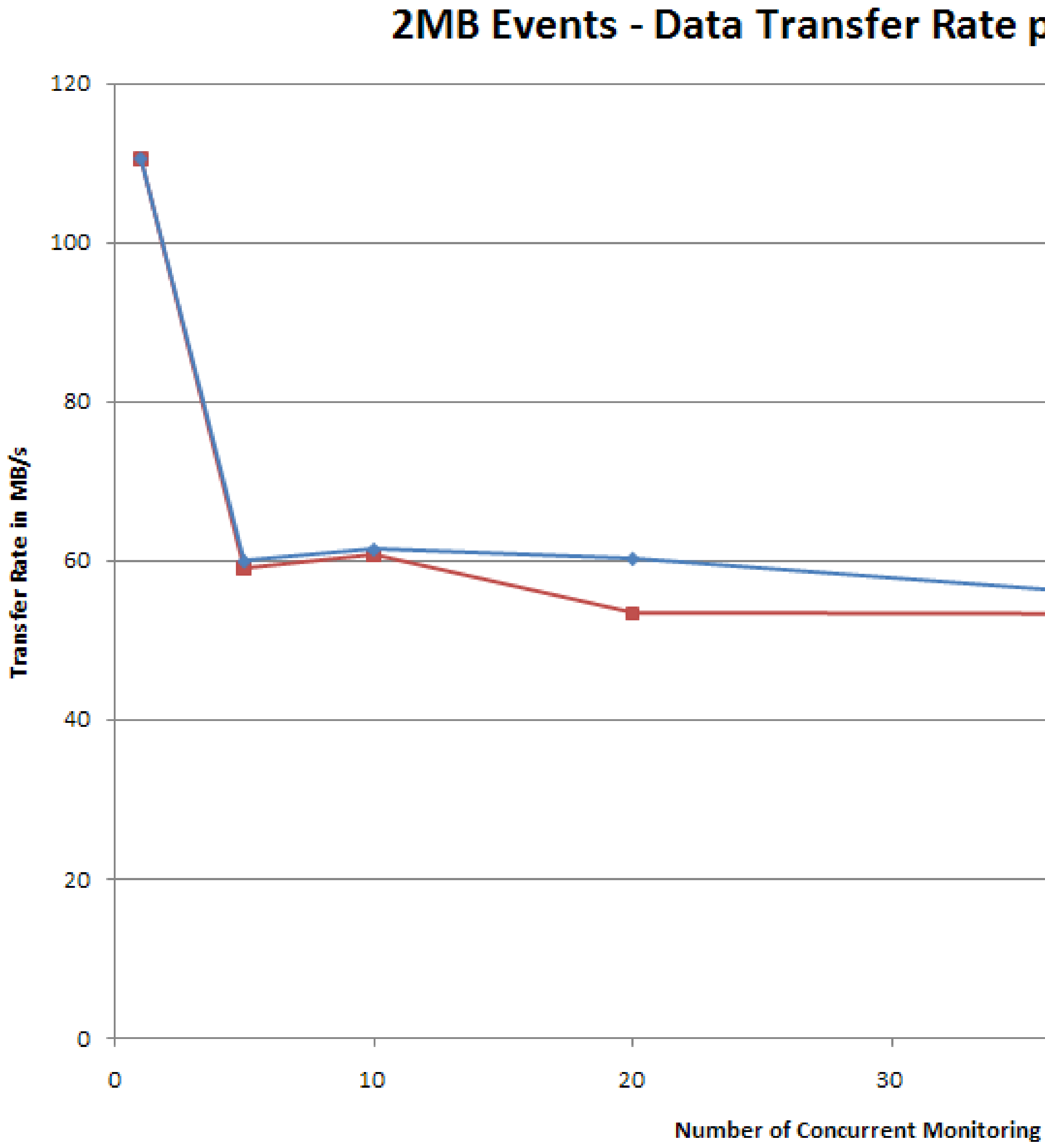
For k=1, the transfer rate per task levels off at roughly 35 MB/s which is about 280 MBit/s.


4K Events - Events per Second pe
25000
20000
15000
10000
5000
0
Events/second
0
10
20
30
Number of Concurrent Monitorin

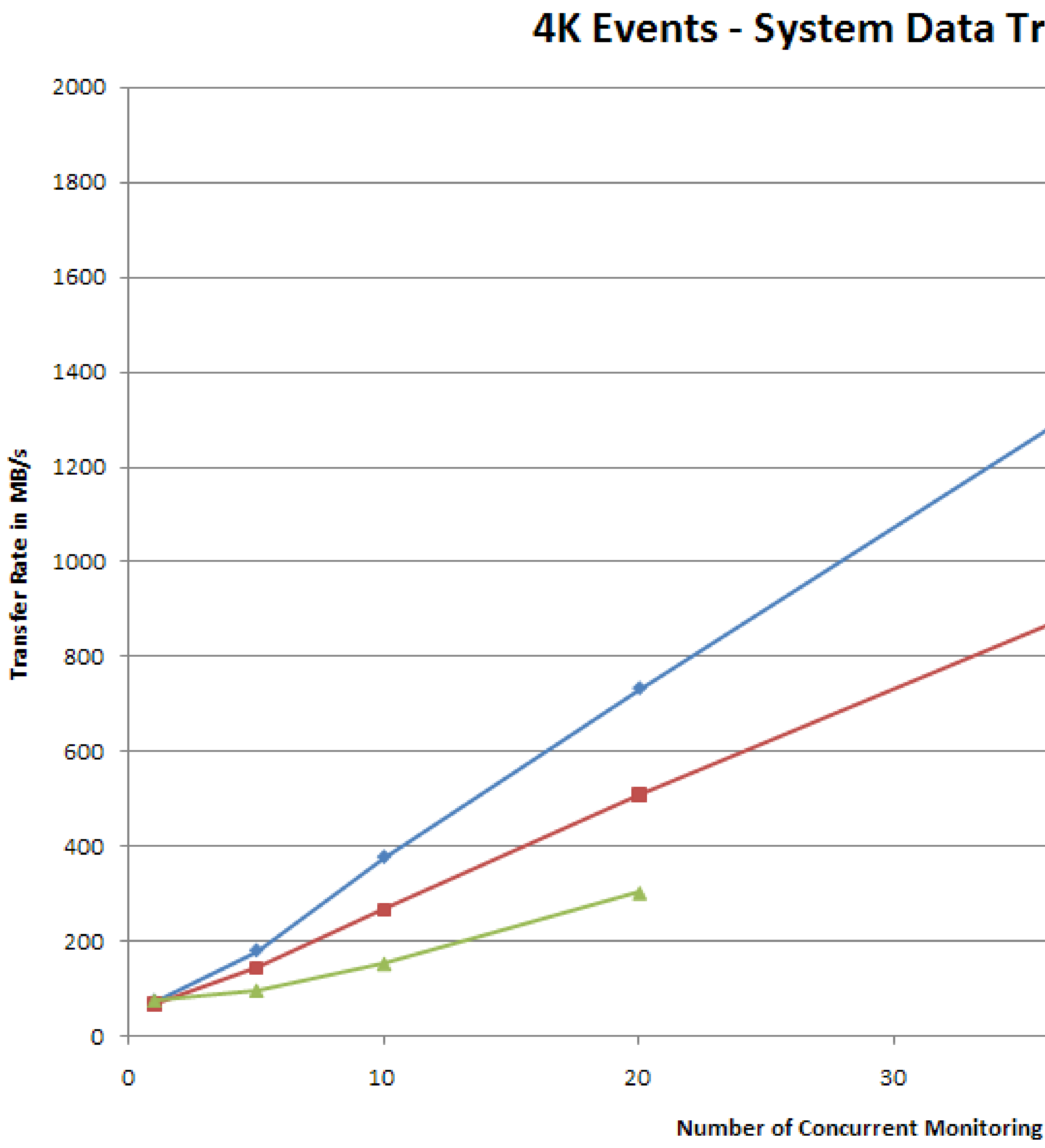


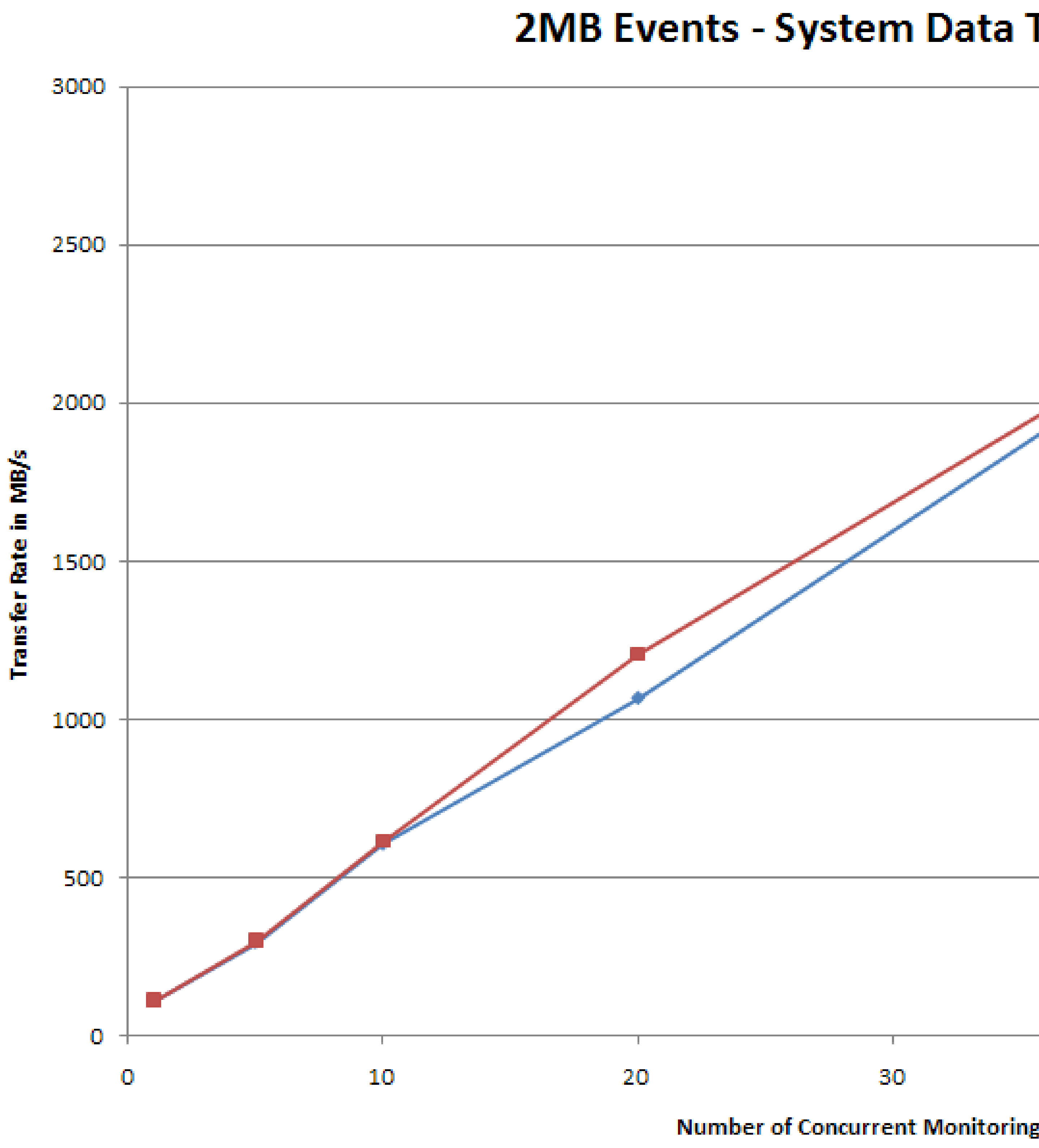
In a second test series, an event size of 2 MB has been used. Again the event rate drops about roughly 45 % from 58 to about 31 events per second as soon as tasks are involved with event forwarding. Again this value stays constant when the number of concurrent tasks is further increased.





These last two figures show the system transfer rate of the whole Monitoring tree. Here the number of tasks receiving events has been multiplied by the worst-case transfer rate of the leaf task. In a centralized solution with the event sampler or a central distribution component representing a bottleneck, this transfer rate would remain constant while the transfer rate per task would linearly drop with an increasing number of tasks. As in the tree-based approach of EMon, the bandwidth of machines running Emon tasks is used for distribution, the system transfer rate linearly increases with an increasing number of tasks. Using an event size of 2MB and the bandwidth of 50 machines running monitoring tasks, the total transfer rate of a sampling channel reached roughly 2.6 GB/s which is about 20 GBit.


4K Events - System Data Tr
Transfer Rate in MB/s
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
10
20
30
Number of Concurrent Monitoring


2MB Events - System Data T
Transfer Rate in MB/s
3000
2500
2000
1500
1000
500
0
0
10
20
30
Number of Concurrent Monitoring

9.5.3 Summary and Conclusions

The tests have proven the scalability of EMon's tree-based approach. They have however also shown some points that may require further improvement. One should further investigate the exact reasons for the drop in the event rate when comparing a single task to a tree consisting of five tasks and see how this drop in the event rate can be attenuated.

Complete: ▭
Responsible:
Author: Haleh Hadavand 19 Jan 2007
Contributors: Doris Burckhart, Haleh Hadavand, scholtes
Last significantly modified by: scholtes 12 Feb 2007
Not yet reviewed

Chapter 10

Summary, Conclusions and Follow-up Actions

10.1 Summary, Conclusions and Outlook

10.1.1 Introduction - under construction

10.1.2 Summary and Conclusions - under construction

The conclusions remain to be written, once the documentation about Lvl2 and DBStressor is included.

In LST06 for the first time the DAQ/HLTsystem was run successfully with full configuration from databases, and at large scale </o:p>

- Separate tests for Level2 and for EF were run </o:p>
- Configuration timing were found to be reasonable and apparently not dominated by database access </o:p>

A number of bugs found and - fixed "online" or immediately after the LST </o:p>

</o:p>

The complete DAQ/HLT system with ROS, Level2, EB and EF without algorithms was run on up to 600 nodes </o:p>

Monitoring and Run Control timings OK </o:p>

Need to work on Fault Tolerance throughout the system </o:p>

Still improvements necessary in the area of Farm Tools, PartitionMaker </o:p>

Operational Monitoring Tools to be put in place </o:p>

We

10.1.3 Outlook

More complete tests foreseen on the hardware installed at Point1 </o:p> with the adequate network structure within racks </o:p>

- using the pre-series machines, then also new 8-core nodes </o:p>
- With all four trigger slices configured from TriggerDB </o:p> (In LST one e-gamma slice from DB - all slices only with jobOptions for Level2) </o:p>
- Configure LVL2 and EF concurrently </o:p> in the same partition </o:p>

Investigate details of trigger algorithm setup timing at configuration transition, esp. CPU intensive parts </o:p>

Rack-specific configuration of CORAL to be integrated into the partition </o:p>

- DbProxy node name depends on rack - had to use inelegant scripts in LST </o:p>

Further tests at SLAC (DbProxy specific) and Manchester</st1:place></st1:city> (trigger specific) and Point1 (integration)

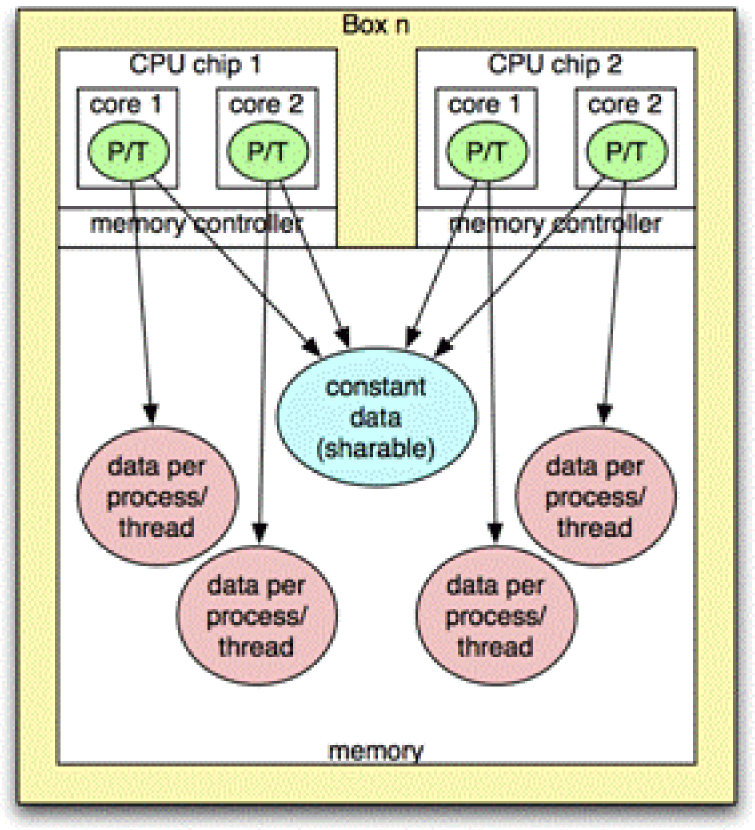
HLT configuration via DbProxy caches was a success </o:p>

- We have a proven way to scale from one to many HLT racks </o:p>
- SLAC group now working on a direct DbProxy to Oracle connection </o:p>
- DbProxy used for data varying for each run: conditions + TriggerDB </o:p> Geometry and magnetic field map are taken from files </o:p>

</o:p>

Cashing:

- At node level, caching could be done as well, but we have nodes with 2*4 cores each already today - more cores to come </o:p>
- need to optimise memory consumption and initialisation effort - options: </o:p>
- global (initialization) thread + multiple event threads share constant data (geometry, fieldmap,...) </o:p>
- initialization process + multiple event processes could use shared memory segment for constant data </o:p>
- options for sharing are now studied in Athena architecture team, for L2, EF, and possibly T0 </o:p>
- Athena object store (StoreGate), if in shared memory, could be loaded with ready-made object data from a file in one go to save a lot of initialization time </o:p>



</o:p>

Complete:
Responsible:
Author: Doris Burckhart 18 Dec 2006
Contributors: Doris Burckhart
Last significantly modified by: Doris Burckhart 18 Dec 2006
Not yet reviewed

–