November 7, 2007

Review Documentation for the CMS Storage Manager

W. Badgett, K. Biery, H. Cheung, J. Kowalkowski, E. Sexton-Kennedy

Fermilab, Batavia, IL, USA

M. Klute, C. Paus

MIT, Cambridge, MA, USA

Storage Manager Working Group

Abstract

The CMS Storage Manager system is designed to store raw data on a storage medium close to the experiment, give real-time access to a fraction of the collected data for monitoring purposes, and collect non-event monitoring data. The design specifications for the system are a total storage capacity of 250 TB and a maximum throughput of 1 GB/s. This note presents the hardware and software architecture of the Storage Manager system, describes future upgrade plans and operational needs are discussed.

1 Introduction

The Large Hadron Collider (LHC) at CERN is scheduled to begin operation in 2008 and will deliver proton-proton collisions to the CMS detector at a center-of-mass energy of 14 TeV and a crossing rate of 40 MHz. This rate must be reduced significantly to comply with the limitations of the computing resources to process the collected data for physics analysis. At the same time interesting physics signals must be triggered with high efficiency. CMS uses a two level trigger system to reduce the rate to approximately 100Hz. The Level-1 trigger is based on custom hardware and designed to reduce the rate to about 100 kHz, corresponding to 100 GB/s, assuming an average event size of 1 MB. The High Level Trigger (HLT) is purely software-based and must achieve the remaining data reduction by executing quality algorithms with an average event processing time of 40 ms. Events accepted by the HLT are send to the Storage Manager system, where they are organized in streams according to the specific physics content as identified by the HLT filter decision. The HLT output rate is expected to be 100-300 Hz and the event size 1-2.5MB. We therefore designed a system which is able to handle 1 GB/s throughput with a buffer of 3 days, equivalent to 250 TB. Beside collecting event data, the Storage Manager is responsible for collection of Data Quality Monitoring (DQM) information produced in the HLT. Events and DQM information are written to disk and a fraction of the information can be served to consumer clients. The raw data files on disk are then moved to the Tier0 computing farm for further processing.

The primary functions of the Storage Manager are the following:

- Receive events from HLT.
- Write events to disks at up to 1 GB/s.
- Transport files to the Tier0 at up to 1 GB/s.
- Buffer data to up to 3 days.
- Serve events and monitoring elements to online monitoring clients.

2 Storage Manager Software

2.1 Storage Manager Functions

The primary function of the Storage Manager (SM) is to collect (event) data from each Event Filter Farm HLT (High Level Trigger) process, and store them in files for Tier-0 to pickup and process. The event data files are written to a set of output streams according to the event trigger bits and the SM configuration information.

The Storage Manager also acts as an event server, and a DQM (Data Quality Monitoring) data server for consumer clients. The DQM data is produced in the Filter Farm HLT processes which has access to the full L1 trigger rate of events before the HLT. The consumers analyze event data or DQM data and are mainly used for monitoring the data, or for calibration and alignment.

The DQM data or monitoring elements are mainly histograms filled with data for a particular period of time, nominally one luminosity section (LS) or about 93 seconds during normal running. Each specific histogram is produced in each HLT process but contain different events for a given LS. So for each LS, all these histograms need to be added together. Besides gathering the histograms from each HLT process together in the Storage Manager, the SM has the additional function of adding up histograms of the same type from each HLT process. The summed histograms are then written out and passed to the DQM system.

These functions of the Storage Manager are represented schematically in Fig. 1.

2.1.1 Software Responsibilities

The Storage Manager project software task has well defined responsibilities. The functions of the code needed are listed below.

- Serialize data into a byte stream for network transfer.
- Create and maintain the serialized data format.
- Provide input modules to deserialize and reform the data for the framework.

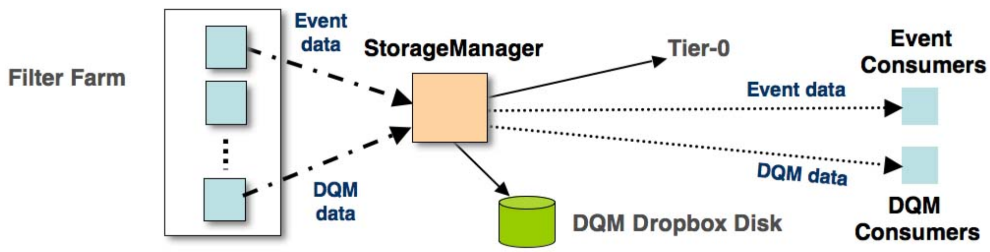

Figure 1: Functional schematic.

- Receive data from all HLT processes in the Filter Farm.
- Output event data in streams based on trigger bits.
- Communicate and interact with Tier-0 for data transmission to Tier-0.
- Serve event data and DQM data to consumers.
- Receive DQM data, sum histograms and output to DQM Dropbox.

Much of the Storage Manager software is well integrated within the standard CMS offline framework. In fact the base classes for the serialized data input source ad serialized data output modules are provided and maintained by the CMS framework group.

2.1.2 Requirements and Dependencies

The Storage Manager runs in the CMS online system which is implemented within the XDAQ framework. The Storage Manager application must be a XDAQ application operating within the CMS DAQ and configured, controlled, and monitored by the CMS Run Control.

The Storage Manager must be able to handle the required data rates and the primary event data collection and output must not be compromised by the other functions of the Storage Manager.

The event data and DQM data consumers must be able to receive the data using a standard CMS EDM offline framework job, or a CMS XDAQ consumer application. Remote consumers must be able to receive data from the Storage Manager.

The intention of the Storage Manager group is to reuse as much of the code in the CMS offline framework, the CMS DAQ, and the DQM framework. This integration means that the SM code is highly dependent on these three subsystems.

2.2 Architecture

The approximately 1000 Filter Farm nodes will be split into 6–8 separate slices or subfarms. Each subfarm will have its own Storage Manager application as illustrated in Fig. 2. A separate XDAQ SM Proxy Server application collects events at a low rate from the Storage Managers of each subfarm to serve to event consumers. The SM Proxy Server application also collects DQM data and sums the DQM histograms before outputing these summed DQM data to the DQM Dropbox disk. The SM Proxy Server application sits on the DAQ private network boundary so it can serve remote event consumers and remote DQM consumers.

Each Storage Manager application also has event data and DQM data server functions, however in normal running the Storage Managers will be isolated from almost all consumers (and from every remote consumer) by the SM Proxy Server.

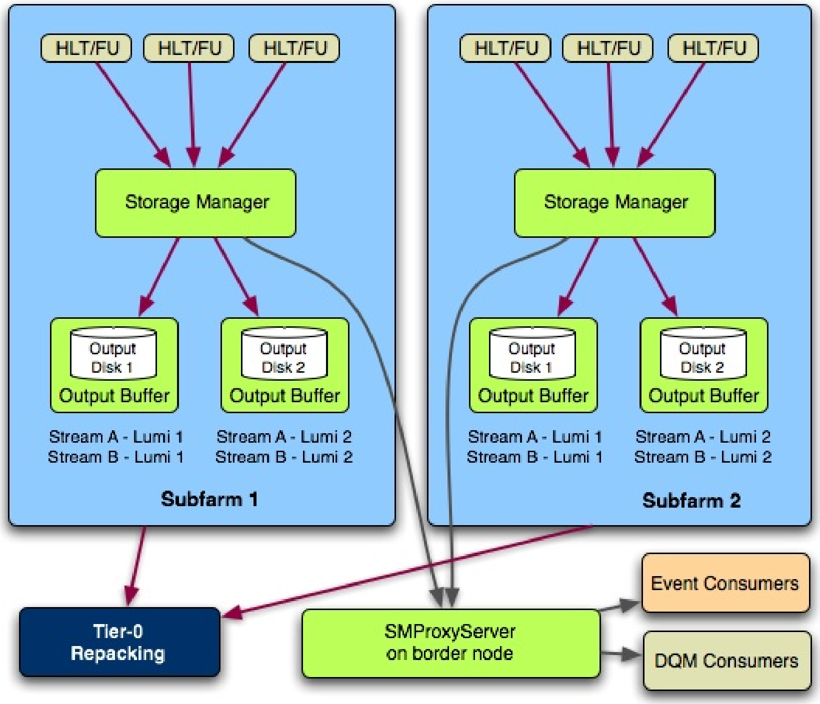

Figure 2: Architecture with subfarms.

2.3 Software Components

In this section we describe each of the software components that form the Storage Manager software project. These are represented by the circles/ovals in Fig. 3. The hardware view from Fig. 14 is used in Fig. 4 to illustrate where the software components run.

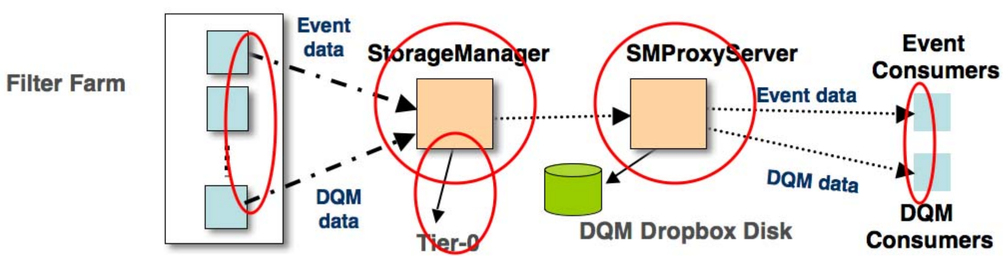

Figure 3: Functional schematic with subfarms.

2.3.1 Output and Input Modules

The Storage Manager group provides the output module for the HLT process. In the output module the event data is serialized into a “blob” of bytes for transfer over the network to the Storage Manager. The actual transfer is done by the Resource Broker running the on the HLT Filter Farm node via the I2O facility in XDAQ. This is illustrated in Fig. 5. The “data blob” is written to shared memory which is managed by the resource broker.

Since the maximum size of I2O frames is 256 KB, and the actual I2O frame size used in the CMS online is smaller (64 KB), the resource broker splits the data for each event into as many fragments as needed, wraps each fragment with appropriate I2O headers and sends them to the Storage Manager. The code for doing the split of the “data blob” was provided by the Storage Manager group though it now resides in the Resource Broker software package and maintained by the CMS DAQ group.

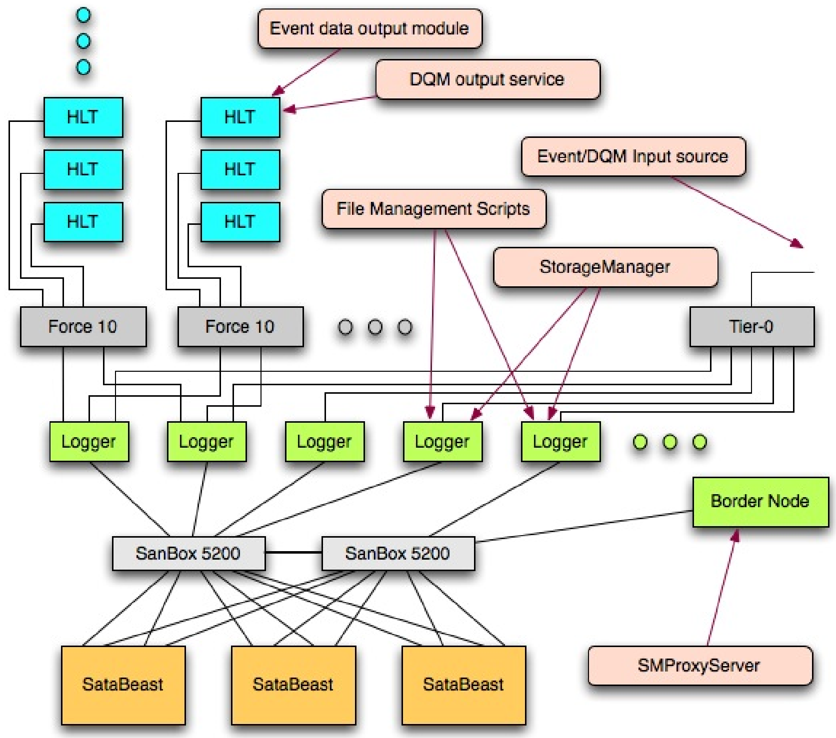

Figure 4: Hardware view showing where the software components run.

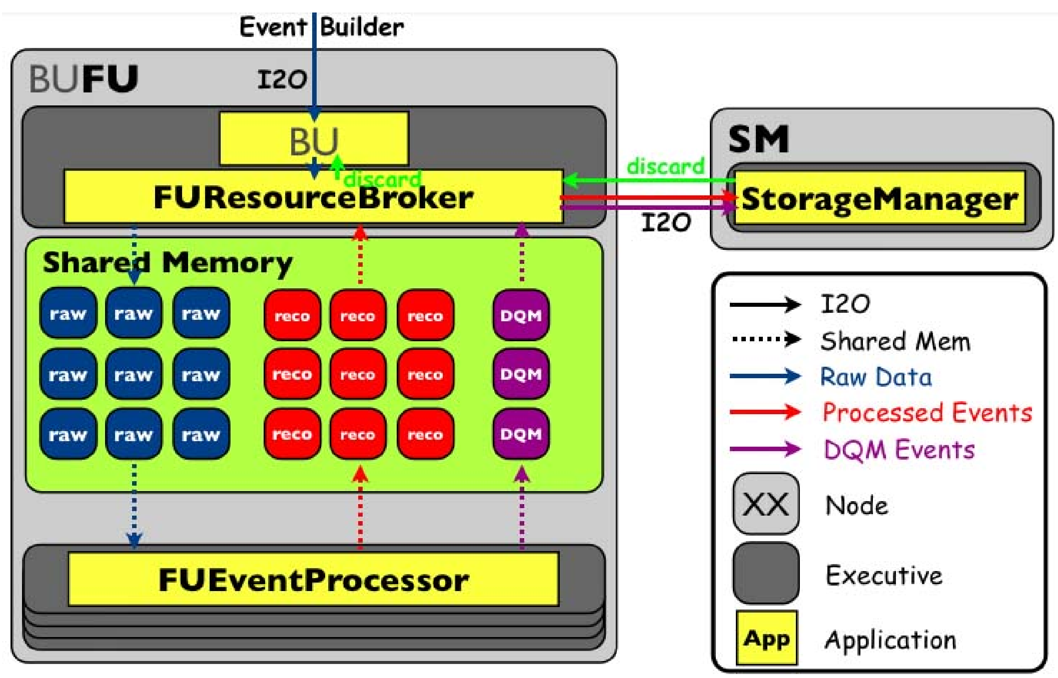

Figure 5: Schematic showing the mechanism for data transfer between the HLT process (FUEventProcessor) and the Storage Manager.

The output module is a plugin module of the CMS offline framework and can be used in either a standard CMS offline job (using cmsRun), or within the online HLT process. The online HLT process uses an instance of CMS offline framework event processor running in an asynchronous mode, and basically has the same features as an offline event processor except its execution can be controlled by the CMS Run Control.

The output module inherits from an output module class provided by the CMS offline framework group, thus the main maintenance load is with the data serialization. The output module is actually a templated class and another output module using the same template is also maintained that writes out streamer files instead of to shared memory. This additional streamer file output module is used internally for testing purposes.

The Storage Manager group provides a DQM output service plugin module. The output service uses the standard offline framework hooks for its processing schedule, and the main function is again to serialize the DQM data and write to shared memory.

The Storage Manager group also provides two input source plugin modules, one for streamer file input and one for network input. Both inherit from a base input source provided and maintained by the CMS framework group. The streamer file input source is used primarily by the Tier-0 group but is also used internally for testing purposes. The network streamer input module is used for event data consumers for online monitoring. These input sources does the deserialization needed to convert the serialized data back into CMS framework event objects.

The network input source used by the online consumers use cURL to interact with the HTTP server of the Storage Manager application, and uses a HTTP GET command to receive data in a binary stream.

These input sources are not used inside the Storage Manager itself. No deserialization is done in the Storage Manager, and it does not use the CMS framework event processor. Details the Storage Manager input and processing is given below in the section on the Storage Manager application.

2.3.2 Serialization and Deserialization

A serialized data format was created for the serialized data. Originally this format was to be a completely temporary format used only for the network transfer of event data to the Storage Manager. The Storage Manager would then write out the standard CMS ROOT files. During the testing of the first prototype of the Storage Manager we found that the deserialization performance was too poor to write out standard CMS ROOT files. Instead in the second prototype we wrote out flat binary files, called streamer data files, these are basically the concatenation of each of the event serialized data messages sent over the network.

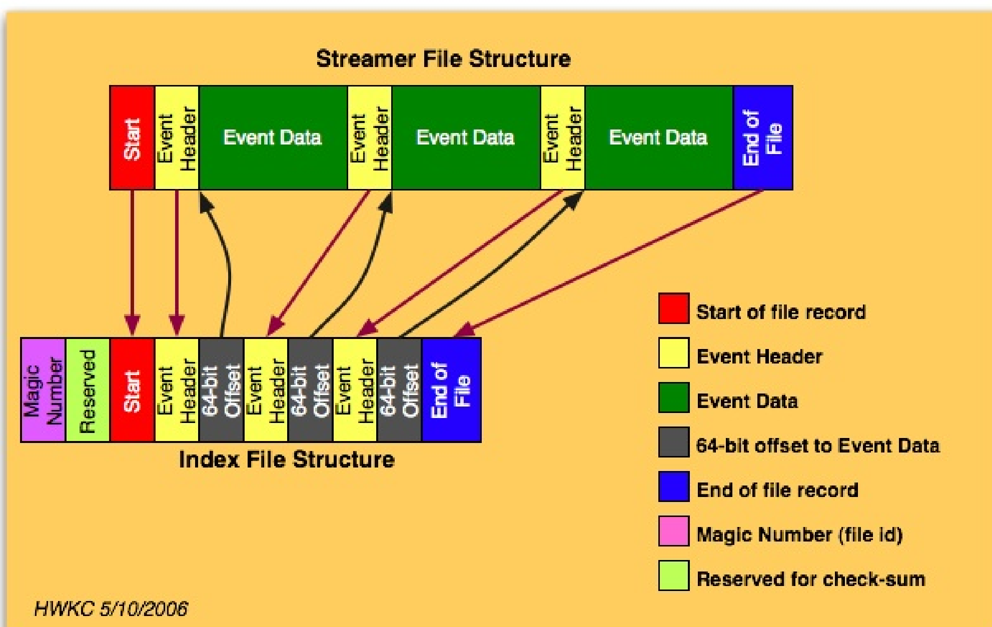

Figure 6: Schematic showing the format of Streamer files and index files.

The streamer file format is illustrated in Fig. 6. The file consists of a start of file record, and an event record for each event, and an end of file record is placed at the end of the streamer file. The event record consists of an event header and the event data.

The actual formats of the start of file record, the event header and data, and the end of file record are given schematically in Fig. 7. The formats of the start of file record, and the event record are exactly the same as the format of messages used to transfer data from the HLT process to the Storage Manager. This has the advantage of having less code for the Storage Manger group to maintain.

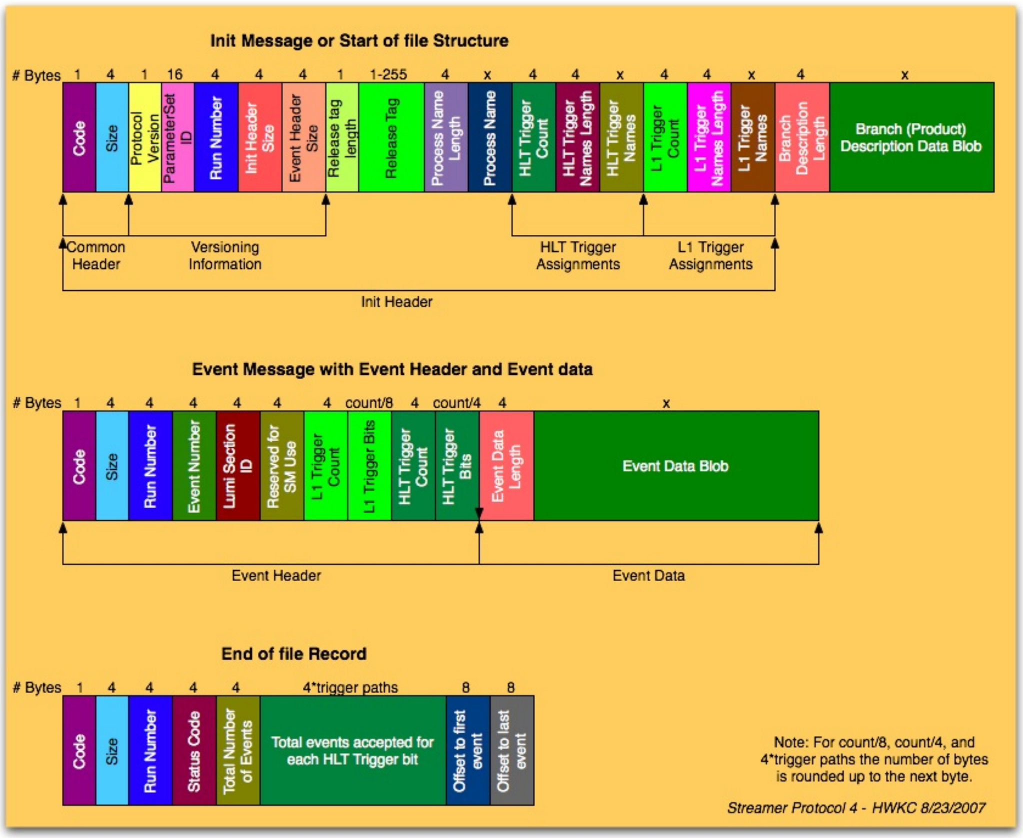

Figure 7: Schematic showing the format of serialized data messages.

At the start of a run, each HLT process sends an INIT message to the Storage Manager. This INIT message is the same as the start of file record. For each run, the data for the first received INIT message are saved for output as the start of file record for each streamer file. The data for each event is transmitted by the HLT processes to the Storage Manager in the form of an event message which is the same as the event record. It consists of the event header and the event data. The event data is just the serialized "data blob" mentioned above preceded by its size in bytes.

Besides the streamer files, the Storage Manager also writes out one index file per streamer file. The format of these index files are given in Fig. 6. The index file contains the same start of file and end of file records as the streamer file, however for each event it contains only the event header and the byte offset for that event's data in the corresponding streamer file. The index file is intended for use during Tier-0 processing to aid in random access reads of streamer files, and is thus only temporary. The index file has the same name as the streamer file but with an extension of ".ind" instead of ".dat".

The transmission of DQM data from the HLT processes to the Storage Manager is done using serialized DQM data messages. The format of these messages are given in Fig. 8. The DQM data are essentially ROOT histograms, and integers, floats, and std::strings stored as TObjects. These data are organized into a number of (top-level) folders and subfolders.

The DQM data for each top-level folder is serialized into a "blob" of bytes and placed in its own DQM data

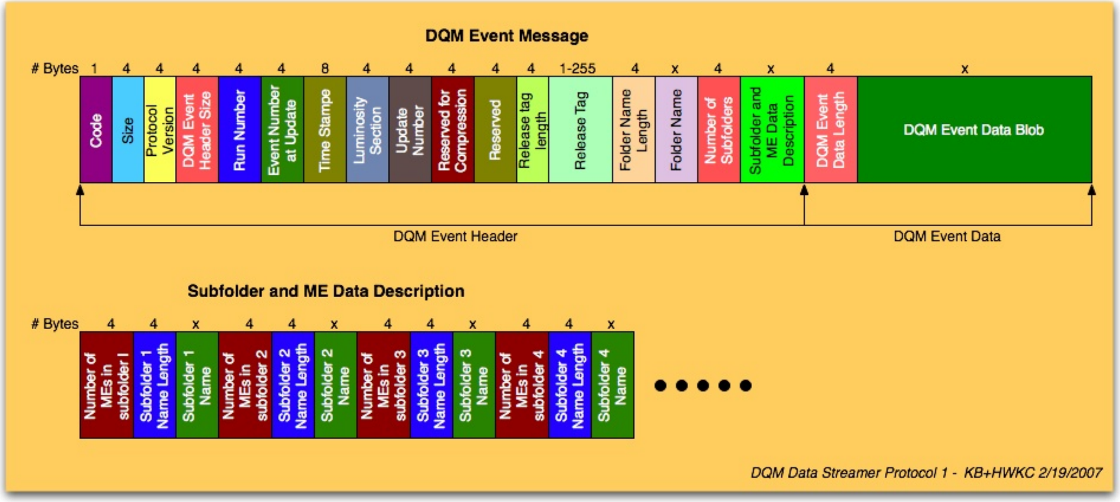

Figure 8: Schematic showing the format of serialized DQM messages.

message, and preceded by some header information as shown in Fig. 8. The DQM data is sent at each "update interval". Usually this update interval is at the end of a luminosity section (nominally 93 seconds).

Both the serialization of event data and the DQM data is done using ROOT. The event data in the CMS framework is stored using ROOT TTrees, and the serialization is done using the class descriptions of the event data objects in ROOT for the particular version of the offline CMSSW version used. Only a subset of the event meta data is written out to the streamer file. The chosen meta data are put into a data structure for which ROOT can generate an automatic class description.

The start of file record in each streamer file includes the version of CMSSW used and the "branch descriptions". These branch descriptions include the identifier of each of the event data branches. However the class descriptions themselves are not stored in the start of file record. To deserialize the data back to the original data objects we load the class descriptions for each of the data branches using the appropriate CMSSW libraries. To be safe this means that the version of CMSSW used to deserialize the streamer file should be the same as that used for writing that streamer file.

There is no provision for schema evolution for the streamer files since they are temporary data formats. However this has implications for the online event data consumers.

The DQM data contain only histograms and simple types and std::strings, no special class descriptions are needed to deserialize these data.

2.3.3 Storage Manager Application

The functions of the Storage Manager XDAQ application is listed below.

- Receive event data and DQM data in the form of I2O frames and reform the fragments.
- Output event data streamer files based on the per run configuration and the per event trigger bits.
- Communicate with Tier-0 for the transfer of streamer files to Tier-0.
- Take online consumer registrations and serve event data and DQM data to registered online consumers.

After the experience with the first prototype of the Storage Manager it was decided that no (real) processing is done in the Storage Manager application since even the deserialization was found to be too CPU intensive for the Storage Manager to keep up with the incoming data rate.

Figure 9 illustrates the design of the Storage Manager application. The StorageManager XDAQ application runs within the xdaq.exe executable in the main thread, but basically all the processing is done in other threads of this application. There are three transport callback threads, one for I2O, one for XOAP, and one for HTTP. Additionally

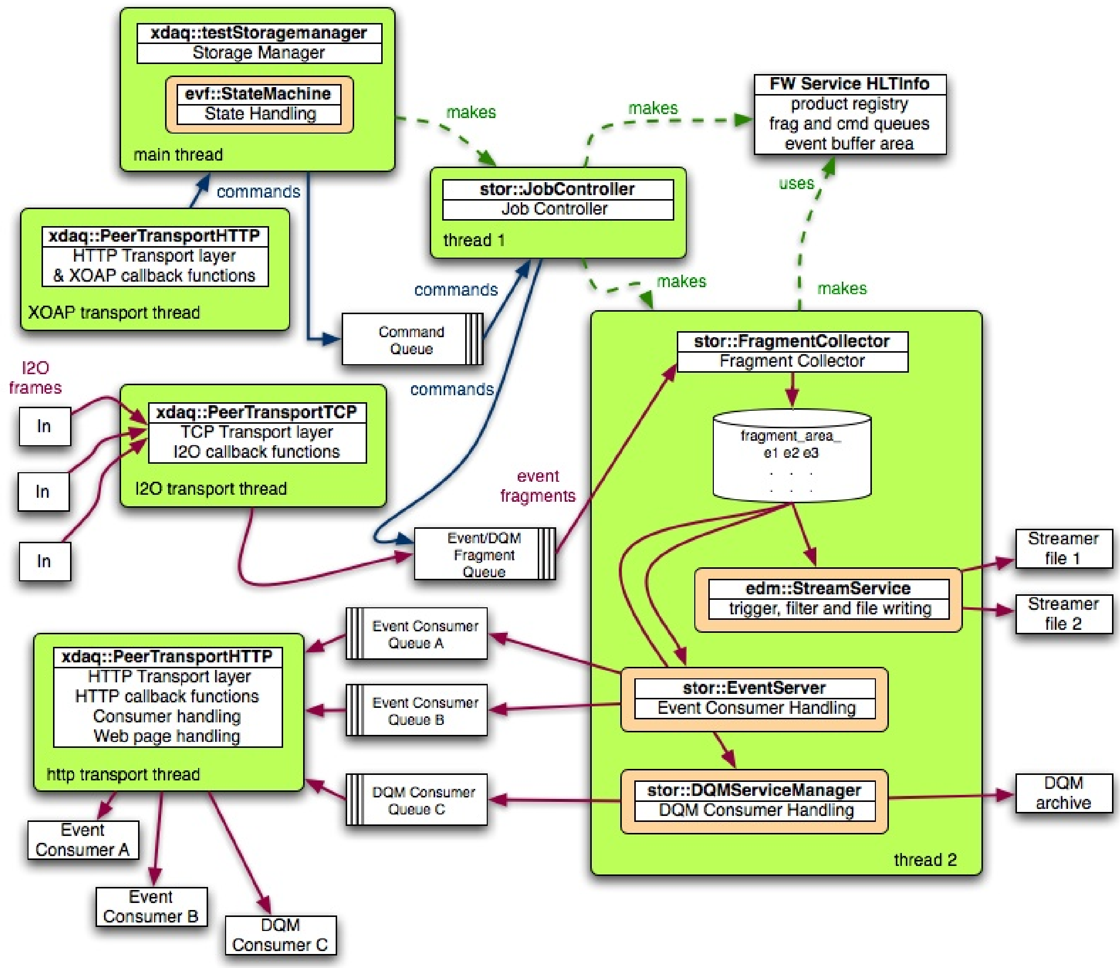

Figure 9: Design of the Storage Manager application.

there are two other threads started originating from the StorageManager application itself, the JobController thread, and the FragmentCollector thread. The functions of these processing threads are given below.

- *XOAP transport thread.* Commands to the Storage Manager are sent via XOAP (XML) messages, and various XOAP callback functions are defined in the Storage Manager application to handle these commands. These commands include commands to configure, start, pause, and end (halt) the Storage Manager application.
- *I2O transport thread.* The incoming event data and DQM data I2O frames trigger I2O callback functions that run in the I2O transport thread. The I2O callback functions are defined in the Storage Manager application and essentially just place the appropriate data fragments with identifiers into the event/DQM fragment queue.
- *HTTP transport thread.* HTTP callback functions are also defined in the Storage Manager application and handle various tasks. The tasks include generating web pages with various monitoring data about the Storage Manager, called application web pages and these are the most direct means of monitoring each XDAQ application besides looking at that application text console. Other HTTP callback functions handle online consumer registration and sending data to the online consumers.
- *JobController thread.* The JobController creates shared data structures like the fragment queue and creates and starts the FragmentCollector thread. The Storage Manager can issue commands to the JobController thread by placing them in the command queue. In the running state the JobController thread just reads the command queue and reacts to what is read. The command queue is a blocking queue so that if empty a read will block until something is placed in the command queue. The command queue is protected by a mutex.

  The JobController thread can control the FragmentCollector thread by placing commands in the (event/DQM) fragment queue. The fragment queue is a blocking queue like the command queue and also protected by a mutex.
- *FragmentCollector thread.* In running mode the FragmentCollector thread reads the fragment queue and processes event fragments, DQM data fragments, and also commands like an end of run or a halt. The data fragments are collected and reformed into the serialized event data or DQM data.

  Once reformed the serialized data for an event is written out to streamer files based on the trigger bits for the event and the initial per run configuration of the Storage Manager. The output files can be written out in streams, for example a physics stream, a calibration/alignment stream, and a debug stream. The data for each luminosity section for each stream is written to one file. The data for the event is also passed to the event server which determines if it needs to be placed in any online consumer queues.

  Reformed DQM data is passed to the DQMServiceManager which can be configured to sum up histograms for the same update. Once the histograms are summed they are written out to the DQM Dropbox disk for pickup by the DQM system. These data are also passed to the DQMEventServer which determines if it needs to be placed in any DQM consumer queues.

The state of the StorageManager application is kept in a Finite State Machine. This is implemented using FSM code provided and maintained by the Event Filter Group which is part of the CMS DAQ group. This provides a uniform set of states and allowed transitions for XDAQ applications within the Event Filter Group. The state chart for the StorageManager application is illustrated in Fig. 10.

The online event and DQM consumers interact with the Storage Manager event and DQM servers through HTTP GET and POST commands, and data are transferred using a binary stream. Online consumers register with the Storage Manager and get data via a pull mechanism, polling at a maximum rate of a several Hz. The event and DQM servers limit the data rate and bandwidth to online consumers to ensure that the primary function of the Storage Manager is not compromised.

Copies of the data are minimized in the Storage Manager during its execution loop. The received data fragments remain in the XDAQ memory pool until all fragments are received. At this point the data is copied and concatenated together to reform the serialized "data blob" before being written out to streamer files. The memory for this event data in the XDAQ memory pool is then released, but the event server retains a copy of this event data. Pointers to this single copy of the data are used in the consumer queues.

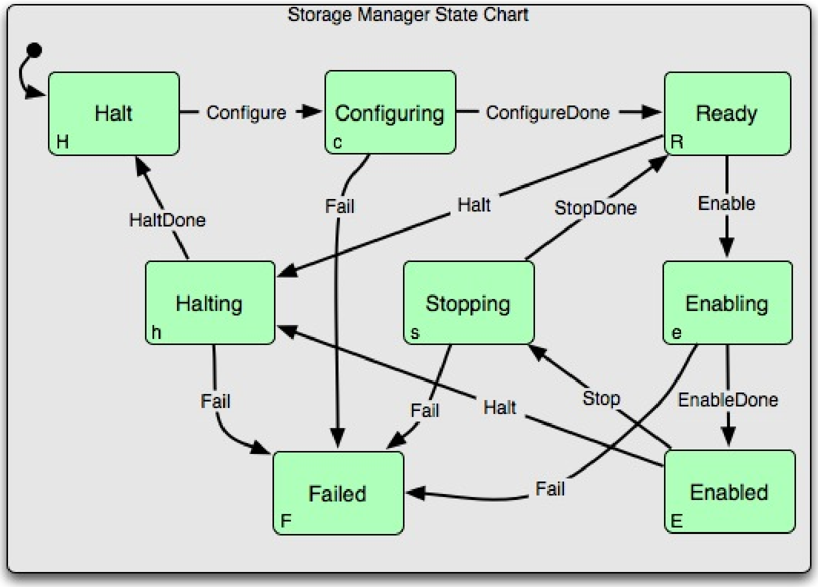

Figure 10: Storage Manager State Chart

2.3.4 Storage Manager Proxy Server Application

When there are multiple StorageManager instances, one for each subfarm, online consumers must be able to get events from multiple Storage Managers. This is one of the functions of the SMProxyServer XDAQ application. The design of the SMProxyServer is illustrated in Fig. 11.

The SMProxyServer gets event data and DQM data from each of the StorageManagers like an online consumer. The SMProxyServer also registers itself with each of the StorageManager applications. However unlike a regular online consumer, the SMProxyServer–StorageManager connection operates in push mode. For example when the event server of a StorageManager instance receives an event that should go to the SMProxyServer, it pushes the event data immediately to to the SMProxyServer instead of putting it on a consumer queue. The event server limits the rate of the pushed data.

As much of the StorageManager code as possible is used in the SMProxyServer to reduce the amount of code that needs to be maintained. Instead of the JobController thread and the FragmentCollector thread the SMProxy-Server only has a DataProcessManager thread. This DataProcessManager thread manages the registration of the SMProxyServer with all StorageManager instances, and processes commands in the command queue.

The SMProxyServer functions as an event and DQM server to serve online consumers. In normal running the online consumers would actually only connect to the SMProxyServer application. This isolates the StorageManager applications from the online consumers. The SMProxyServer is located on the border of the DAQ private network so that remote online consumers can also connect to it.

Finally the SMProxyServer receives DQM data from each of the StorageManager applications and sums up the same histograms from each StorageManager for the same update and saves them to the DQM Dropbox disk.

The SMProxyServer application uses the same state machine code as the StorageManager application and the state chart is the same as that given in Fig. 10.

2.4 File Processing, and Communication and interaction with Tier-0

For each stream and each StorageManager instance one output file is written per luminosity section. To optimize concurrent writing to and reading from the disk file system the files for all streams for the same luminosity section are written round robin to the set of disks.

Events arrive asynchronously to the Storage Manager depending on the processing time for each event in the HLT process. On the arrival of an event from a new luminosity section a new streamer file is created for this luminosity section, and an entry is created in the online database for this file. The arrival of an event from the next luminosity

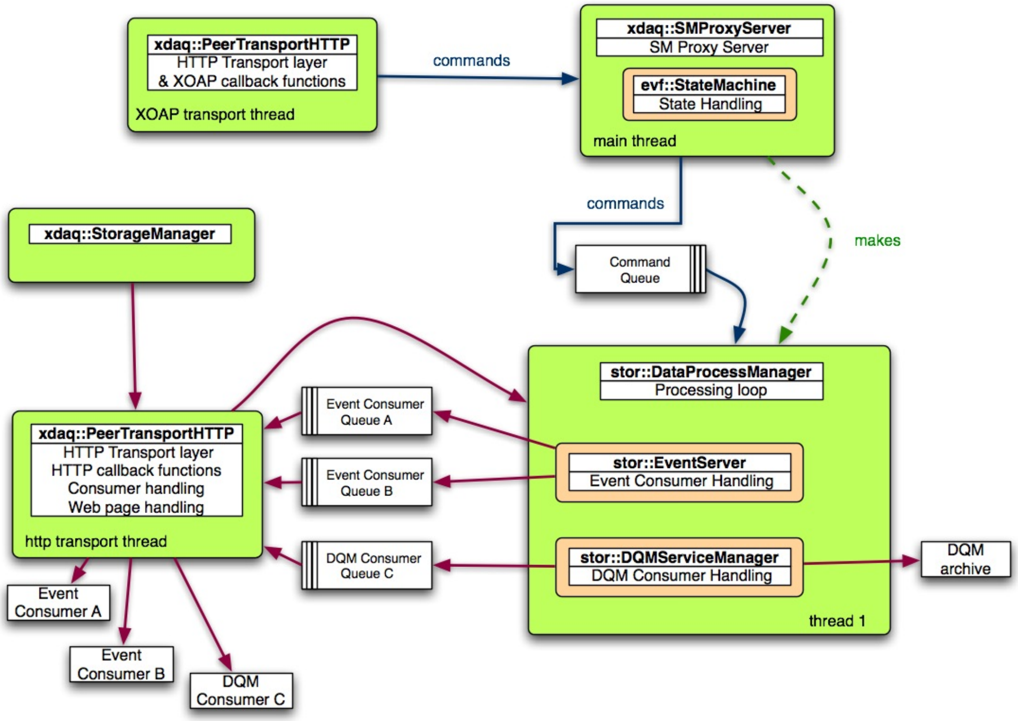

Figure 11: Design of the Storage Manager Proxy Server application.

section to a StorageManager will start a timeout counter. Once this timeout is reached all files for the previous luminosity section is closed. The database entry for each of the closed files are updated. The information updated includes the number of events in the file and a file CRC checksum. The files are moved to a staging subdirectory and the Tier-0 system is notified using a shell script provided by the Tier-0 group.

Currently the database creation, update and the Tier-0 notification is done using PERL scripts called by the StorageManager instance. In a planned future update the database interaction will be done using standard XDAQ database tools, and the Tier-0 notification script could be triggered by the database instead of the StorageManager.

A set of PERL scripts monitors the status of each of the closed streamer files via the database and manages the removal of streamer files. In a future update the removal of streamer files could be triggered by the database for some specific update.

All scripts used in the Storage Manager and maintained by the Storage Manager group reside in CVS along with all the C++ code.

2.5 Control, Monitoring, and Error Handling

2.5.1 Control

Control of the Storage Manager is done by the CMS Run Control using XOAP messages. These XOAP messages triggers the state transitions. If problems occurs during the state transition the Storage Manager will go into the Failed state instead as illustrated in the state chart, see Fig. 10.

The Storage Manager is also controlled by configuration data that are read by the Storage Manager during the transition from Halt state to the Ready state. A new run is always started when the Storage Manager is configured or reconfigured. The configuration data determines what streams the Storage Manager will write out and the trigger selection for each stream. The configuration data also specify which disks will be written to, and govern the setup of the event server and DQM server, as well as various other parameters that govern the behavior of the StorageManager and SMProxyServer applications.

2.5.2 Monitoring

There are different levels of monitoring implemented for the Storage Manager system, from the lowest level closest to the StorageManager and SMProxyServer XDAQ applications, to higher levels where the monitoring data has to pass through additional software.

Monitoring of the software at the lowest level is via the XDAQ application console and the XDAQ application web pages. The console just contains text that is output from the application. Different levels of text are generated depending on their information content or severity, *i.e.* debug, informational, warning, and error levels. The text that appears from an application can be filtered and also directed to specific consoles. These text messages are created using the standard offline framework message logger integrated with the online messaging system.

Each XDAQ application contains a simple HTTP server that can serve web pages to a browser. The StorageManager and SMProxyServer applications both have a number of monitoring web pages defined. These monitoring pages are dynamically generated when a browser requests a specific web page. The monitoring pages include information on each HLT node sending data like the node address, data rate, and latency. For the Storage Manager itself, some information that is currently included are the status of the XDAQ memory pool, the total maximum, minimum, and average data rate into and out of the StorageManager, the average event size, the number of events received, and statistics for the each streamer file written or being written. Other web pages give the status of connected consumers and the status of the event and DQM servers.

Although a lot of information on the software status is available at the lowest level if needed, typically we minimized the processing the Storage Manager has to do to processes incoming web page requests. We make use of other monitoring features of XDAQ. A list of important properties of the Storage Manager is defined at configuration time and the Storage Manager updates the information for this list. The list is accessible by other online applications like the CMS Run Control. The values of the list of quantities are published in the online XDAQ monitoring system and any number of applications can access this information.

Standard web pages are generated using the monitoring lists for each XDAQ application. Each online subsystem has a single web page summarizing the status of that subsystem, called its “Page One”. The Storage Manager Page One has information mainly about the status of streamer files, and the status of the output disk system. An example of the Storage Manager Page One is given in Fig. 12.

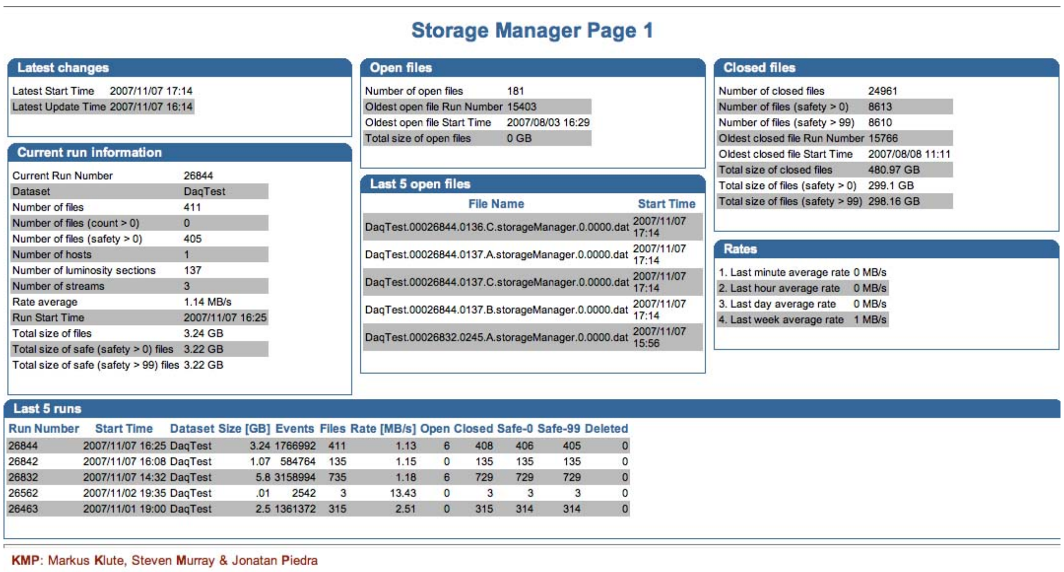

Figure 12: Image of the Storage Manager Page 1 monitoring web page.

The monitoring of the Storage Manager hardware will be covered in the hardware section.

2.5.3 Error Handling

There is currently little error handling in place for the Storage Manager. This will form a large part of the ongoing and future work of the Storage Manager group. Fatal errors will cause the Storage Manager to go to the Failed state and the situation causing the problem is analyzed.

There are unit tests in place for the data serialization and deserialization code, and for the streamer message creation and access. These unit tests are run during each CMSSW pre-release and the production release to ensure that these code function properly. The tests of the StorageManager and SMProxyServer applications relies on a number of applications and already constitutes an integration test. These tests are currently done manually on a test system consisting of two nodes. Currently no scale tests are done as part of the software release procedure.

CMS has had a number of DAQ global run exercises and the Storage Manager is an integral part of every global run exercise. The Storage Manager system is scale tested during these global exercises. Any problems or issues in each run are diagnosed and corrected before the next global run exercise. Additional functionalities or features of the Storage Manager may also be tested in the global run exercises and in test running between these exercises.

Currently we are still collecting data on the types of error conditions we face from the global run exercises. We will be evaluating these data to form our fault tolerant and recovery policy. This policy will be an integrated policy that fits in with the CMS online plan for fault tolerance and recovery.

2.6 Status, Software Schedule, and Software Organization.

The Storage Manager project started slowly in June 2005 with just two people from Fermilab. The number of people in the Storage Manager group doubled with time and the first prototype was produced for the Magnet Test and Cosmic Challenge (MTCC I) in May 2006. During this testing it was found that the deserialization step in the Storage manager was too slow to write out ROOT output files thus the decision was made to write out the binary streamer files as a temporary format for use within the Tier-0 processing.

Another lesson learned was that the Storage Manager group needed a real presence at CERN at the experiment during data taking. The MIT group joined the Storage manager group and strengthened the group enormously in this area. The MIT group took over the operations responsibility, including the interaction with Tier-0 and the hardware.

A second prototype Storage Manager writing streamer files was produced for MTCC-II with more optimized concurrent writing to and reading from the disk system. The event server functionality was commissioned during MTCC-II. Remote consumers were served by an adhoc proxy server written to run on an Apache HTTP server. The performance goals for the Storage Manager were met in MTCC-II.

The Storage Manager group was further strengthened with additional manpower to finish the production version for the first CMS global run exercise in May, 2007. The SMProxyServer was written but is still not commissioned in the experiment. An additional feature request of handling DQM data in the Storage manager was taken and completed. A series of performance tests in joint exercises with the Tier-0 team was done early in 2007 in preparation for the May global run exercise.

During the 5 global run exercises conducted so far the Storage Manager goals were met and performed well. There were of course a number of smaller issues and problems during the exercises or additional fault tolerant features needed. The Storage Manager team were able to respond to problems usually in a timely manner.

The schedule of software releases for the Storage Manager is govern by the schedule of global run exercises and CMS Cosmic Runs (CCR) as well as the actual start of collision data. These are given in Fig. 13 for the near future. Most of the future work for the Storage Manager involves commissioning the SMProxyServer, including further testing of the DQM data collection and summing, scale tests of the Storage Manager, improving monitoring, robustness of the code and error handling. In additional documentation of the Storage Manager will also be completed.

The Storage Manager team resident at Fermilab will continue to do the bulk of the software development and code maintenance work, while the MIT team will continue to be responsible for the interaction with Tier-0 including development and maintenance of the code related to this task, with operations, and with the Storage Manager hardware.

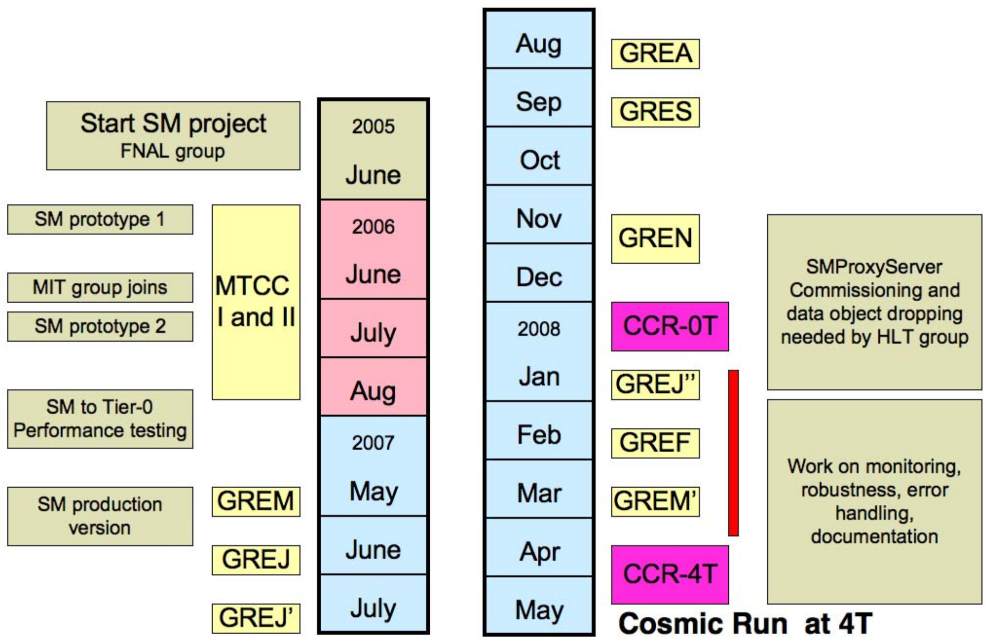

Figure 13: Schedule for CMS global data taking exercises and Storage Manager software work.

3 Storage Manager Hardware

3.1 Overview

An overview of the Storage Manager (SM) hardware configuration is given in Figure 14. The HLT nodes are grouped in subfarms and connected to eight gigabit ethernet switches (Force10). Data from the HLT is sent to 8 SM nodes (Logger). The Logger nodes are also connected to a GigE switch (Tier0) to provide connectivity to the Tier0 system at CERN. The connection to the NexSan SATABeasts is provided through two Fibre Channel QLogic SanBox 5200 switches. Each NexSan SATABeast unit has two controllers with two fiber connections. If one controller fails, traffic can be switched to the second controller. The total network connectivity from the Logger nodes to the storage units is 8 time 2 Gb, such that the system is limited by the performance of the SATABeast controller. According to specification each NexSan SATABeast should be able to maintain 400MB/s write and 800MB/s read speed, if equipped with a sufficient number of disks.

3.2 Hardware Components

3.2.1 Logger Nodes

Logger nodes are standard Dell PC's PowerEdge 2950 with two dual-core Xeon processors and 4 GB RAM. Each node is equipped with a 4-port GbE card (Silicom PEG4i) and a fiber channel host bus adapter (QLogic QLE2460-CK). An additional PCI-e slot is available for an extra 4-port GbE card if needed. The same PC's are used for the CMS HLT farm.

3.2.2 Fiber channel switch

The fiber channel switches are used to connect the Logger nodes and the storage. Two QLogic SanBox 5200 switches are available. Each chassis includes sixteen 2Gb ports, plus a four-pack of high-speed 10Gb ISL ports of which one is used to connect the two switches.

3.2.3 Storage Array

The storage arrays are the SATABeast from NEXSAN. Currently, 3 SATABeasts are available with ten 750 GB Sata disks per array resulting in 21 TB storage. Each system has two controller with two 2 Gb ports. The total number of disk slots available is 42 and the system can support Sata disks of up to 1 TB size. NexSan SATABeasts have been successfully deployed for the CDF Consumer Server Logger system in 2006.

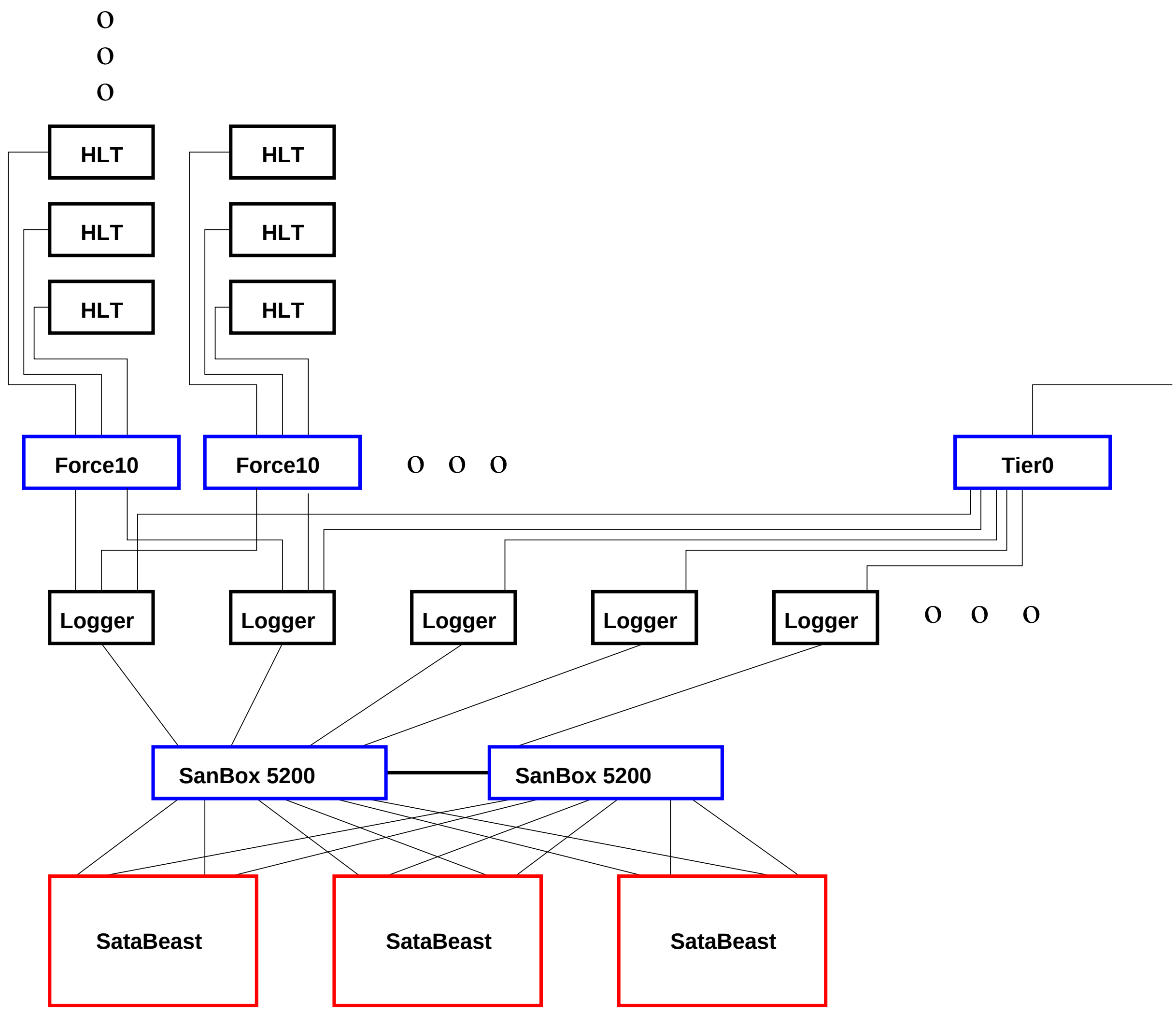

Figure 14: *Storage Manager system overview.*

3.3 Expected Performance

The proto-type system, currently installed in Cessy, has a storage capacity of 21 TB and is expected to be limited to 400MB/s per SataBeast for concurrent reads and writes.

The HLT subfarm and the logger nodes are connected with 8 times 2 GbE links which can be extended to 8 times 6 GbE links using a second Silicom card. An additional switch could be used to connect all Logger nodes with all HLT subfarm to allow for easier system configurations.

The Tier0 switch and each Logger nodes will be connected with 2 GbE links.

The proto-type system will already be able to provide a throughput of 1 GB/s. In 2008, shortly before we can expect the first collisions we plan to increase the storage capacity more than 100 TB by filling all available 126 slots with SATA disk.

To increase the storage capacity to the design capacity of 250 TB, additional SataBeast have to be added. With 3 additional storage arrays the system can be increased to more than 200 TB. This addition will also increase the IO bandwidth.

3.4 Commissioning Plans

The proto-type system has been installed during the summer 2007 and is currently being configured. Connectivity and small scale system tests will be carried out in November 2007. End of November the system will be used for a Global Run commissioning exercise. End of December performance tests will be carried out to benchmark the system performance.

3.5 Cost and Upgrade Plans

Table 1 shows the costs for the proto-type system with the exception of the Logger nodes, Silicom cards and fibers, which where ordered together with the nodes used for the HLT farm or a database project. In total \$80k were spend on the proto-type system. Costs and cost projections for Storage Manager hardware are summarized in Table 2. With an additional \$51k, the storage capacity can be increased to more than 100 TB. To increase the storage capacity and the connectivity even more, additional SataBeasts are needed. In order to increase the capacity to more than 200 TB an investment of \$117k is needed. Given a limited lifetime for disks and host bus adapter, we assume that about 20% have to be replaced after 3 years. The cost projections are given assuming spring 2007 prices for the SATABeast and 750 GB SATA disks.

year	cost	product
2007	$66k	3 SataBeasts with 10 disks
2007	$6k	2 QLogic switches
2007	$8k	8 QLogic HBAs
2007		8 Dell PowerEdge 2950
2007		8 Silicom PEG4i
2007		20 2m fiber

Table 1: Proto-type system costs.

year	cost	product
2007	$80k	proto-type system
2008	$51k	96 SATA disks
2009	$66k	3 SataBeasts with 10 disks
2009	$51k	96 SATA disks
2010	$17k	Hardware replacement

Table 2: Storage Manager system costs.