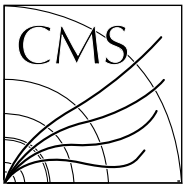
Available on CMS information server

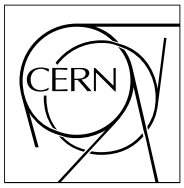
CMS NOTE 2001/037



The Compact Muon Solenoid Experiment

CMS Note

Mailing address: CMS CERN, CH-1211 GENEVA 23, Switzerland



July 16, 2001

CMS Data Grid System Overview and Requirements

Koen Holtman[1)], on behalf of the CMS collaboration

Abstract

This document gives a comprehensive overview of the data grid system that CMS intends to operate around December 2003. This CMS data grid system will support both CMS production and analysis and will seamlessly tie together resources at CERN and at international CMS regional centers.

This document focuses on the relation between the CMS software components and the grid software components operating inside the 2003 CMS data grid system. In addition, the document includes overview and reference material to introduce the members of the grid projects (GriPhyN, PPDG, and the EU DataGrid) to the CMS data handling environment.

This document contains a *snapshot*, taken in 2001, of the vision that CMS has of the intended software capabilities of its production data grid system in 2003, and the expected scaling towards 2007. To capture the expected level of complexity, the vision is sometimes worked out to considerable detail, even though some of these details are likely to be adjusted in future. Though the vision captured in this document will likely evolve, this document does yield the current requirements for the grid projects that CMS is involved in as a 'customer', the requirements for the grid components which are to be created from now until the end of 2003. The major CMS software milestones affecting the grid projects are the 'delivery of baseline core software' milestone for December 2002, this includes the choice and integration of grid components into the baseline software, and the '20% data challenge' milestone, for which the work starts in January 2004 with milestone completion in December 2004. The 20% data challenge includes the full range of distributed operations required for the analysis of CMS data under realistic conditions as occurring during LHC operation from 2006 onwards.

The primary audience for this document are the participants in the grid projects that CMS is involved in.

[1)] Division of Physics, Mathematics and Astronomy, California Institute of Technology

Contents

The Computing Model represents the architecture of a system of four interconnected resources: computing and network hardware; data; software; and people. These must function effectively in the context of more than a billion physics events per year, more than a thousand physicists located at over one hundred institutes, with a detector and physics complexity unprecedented in High Energy Physics.

– From the executive summary of the CMS computing technical proposal, December 1996 [8].

1 Introduction: how to use this document

The primary audience for this document are the participants in the grid projects (GriPhyN [1], PPDG [2], and the EU DataGrid [3]) that CMS [4] is involved in. Parts of this document will also be useful to CMS physicists to get an overview of the expected capabilities and limitations of the CMS data grid system.

This document can be read as a comprehensive overview of the CMS data grid system requirements, but it can also be used as a reference guide. Because of this dual purpose, the document consists of more or less self-contained sections, which can be read partially and in any order.

Note to computer scientists: It will be useful to read section 2 first as an introduction to CMS and its physics. Much of the material in section 2 applies also to other high energy physics experiments. The terminology section 7 might be useful as a reference while reading other parts of this document.

Note to CMS physicists: Section 2 can be skipped on first reading. It might be useful to read section 7.2 first as an introduction to grid terminology. This document uses some terms which are different from those normally used internally in CMS. Specifically:

Usual CMS term	This document
Object	Data product

Acknowledgments

Editorial help in preparing this document was provided by Takako Hickey and Stephan Wynhoff. Thanks go to the many members of the CMS collaboration and the grid projects who provided comments, corrections, and background material. Particular thanks go to the organizers, participants, chair, and minute takers of the workshop on CMS and the grid at the Catania CMS week in June 2001.

2 Brief overview of CMS and its physics

2.1 CMS

The CMS experiment is a high energy physics experiment located at CERN, that will start data taking in the year 2006. The CMS detector (Figure 1) is one of the two general purpose detectors of the LHC accelerator. It is being designed and built, and will be used, by a world-wide collaboration, the CMS collaboration, that currently consists of some 1800 scientists in 144 institutes, divided over 31 countries.

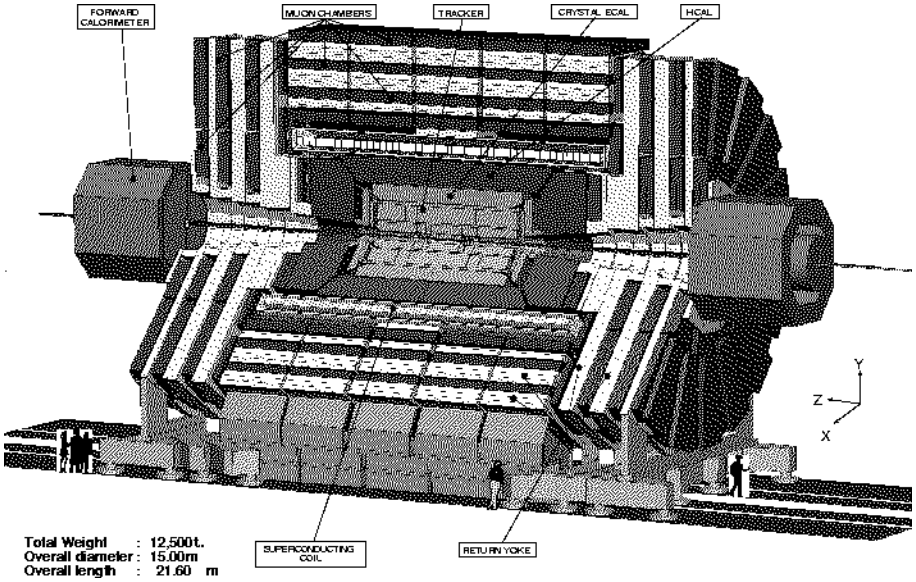


Figure 1: The CMS detector (cutout view).

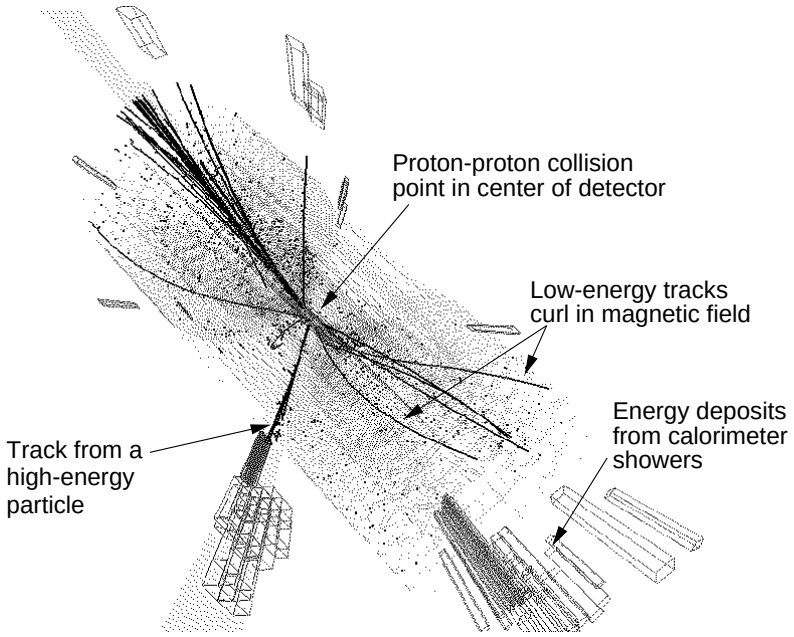


Figure 2: A CMS event (simulation at low luminosity).

In future operation, a large number of particle bunches, each containing some 10^{11} protons, circulate inside the LHC accelerator. At a rate of 40,000,000 times per second, the LHC accelerator lets two bunches of particles coming from opposite directions cross through each other inside the CMS detector. In every bunch crossing in the

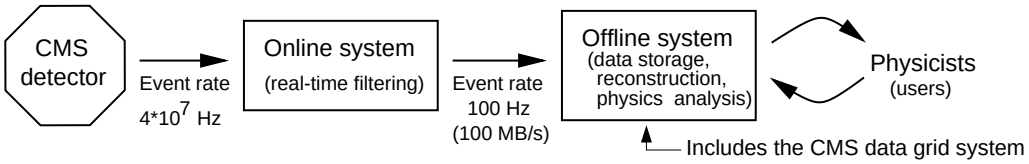


Figure 3: The CMS online and offline systems and their environment.

detector, on average 20 collisions occur between two protons from opposite bunches. The collision phenomena that occur during a single bunch crossing are called an *event*. Figure 2 shows an example of an event, with the collision products emanating from the collision point. Note that the picture of the collision products is complex, and represents a lot of information: CMS has 15 million individual detector channels. The measurements of the event, done by the detector elements in the CMS detector, are called the *raw data*. The size of the raw data for a single CMS event varies around 1 MB.

CMS has the high event rate of 40,000,000 events per second because, like in many high energy physics experiments, the most important phenomena that the physicists want to observe will occur with only a very low probability in any single event. Of the 40,000,000 events in a second, some 100 are selected for storage and later analysis. This selection is done with a fast real-time filtering system (figure 3). The raw detector data of each selected event is stored as a set of 'data products'. In this document, the term **data product** is used for a small self-contained piece of data. In CMS terminology (section 7), data products are usually called 'objects'. As the word 'object' is heavily overloaded we tried to avoid it in this document.

Data analysis is done by CMS physicists working around the world. The 1 MB raw event data for each event is not analyzed directly. Instead, for every stored raw event, a number of summary data products called *reconstructed data products* are computed.

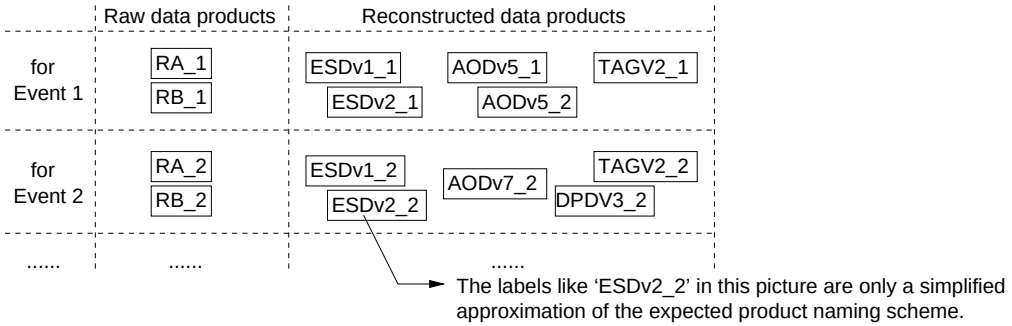


Figure 4: Different raw and reconstructed data products that exist at some point in time for two events. Each event has a fixed number of raw data products, but a variable number of reconstructed data products.

Figure 4 shows the different data product values that may exist at one moment in time for two different events. For each event, the reconstructed product values have been computed, in some way, from the values of the raw data products of these events. Reconstructed data products range in size from a 500 KB full reconstructed tracks data product to a 1 KB tag data product.

Collections of reconstructed data products will be replicated widely, depending on needs and capacities. The original raw data will stay mostly in CERN's central tertiary storage system, though some of it may be replicated also. Tertiary storage will likely be a robotic tape store, though maybe by 2006 some other bulk storage medium (DVD?) will have replaced tape. Due to the slowness of random data access on tape robots, the access to raw data will likely be severely limited.

2.2 Physics analysis

By studying the momenta, directions, and other properties of the collision products in the event, physicists can identify particles and learn more about the exact nature of the particles and forces that were involved in the collision.

One of the prime objectives of CMS is to confirm the existence of a particle called the Higgs boson, which is the

origin of mass. In the framework of the Standard Model of particle physics, particles acquire mass through their interaction with the Higgs field. This implies the existence of a new particle: the Higgs boson H^0. The theory does not predict the mass of the Higgs boson. CMS has been optimized to discover the Higgs in the full expected mass range 0.08 TeV $< M_H <$ 1 TeV.

It is predicted that a Higgs boson decays almost immediately after creation and cannot be observed directly. One possible way that a Higgs boson is predicted to decay is into two Z bosons which then further decay into two charged leptons each. Therefore, one way in which a Higgs boson analysis effort can start is by isolating the set of events in which four charged leptons were produced. Not all events in this set correspond to the decay of a Higgs boson: there are many other physics processes that also produce charged leptons. Therefore, subsequent isolation steps are needed, in which *background* events, in which the leptons were not produced by a decaying Higgs boson, are eliminated as much as possible. Background events can be identified by looking at other observables in the event data like the non-lepton particles that were produced, or at the momenta of the particles that left the collision point. Once enough background events have been eliminated, some important properties of the Higgs boson can be determined by doing a statistical analysis on the remaining events.

The data reduction factor in this type of CMS physics analysis is enormous. The final event set in the above example may contain only a few hundreds of events, selected from the 4×10^{14} events (10^{16} collisions) that occurred in one year in the CMS detector. This gives a data reduction factor of about 1 in 10^{12}. Much of this reduction happens in the online system (figure 3) before any data is stored, the rest happens in a more interactive way, using the CMS offline system.

Physics analysis on the offline system is an iterative, collaborative process, in which subsequent versions of event feature extraction algorithms and event selection functions are refined until their effects are well-understood. The grid jobs run during this process can be compared to the compile-and-run steps in iterative software development. The grid job 'locate the Higgs events and calculate the Higgs mass from them' is highly atypical: it is the final job at the end of a long analysis effort. A much more typical job run by an individual physicist in an analysis effort is: 'run this next version of the system I am developing to locate the Higgs events, and create a plot of these parameters that I will use to determine the properties of this version'. Another typical class of grid resource usage is not initiated by individual physicists, but by system operators on behalf of a larger group. A typical system operator command to the grid is: 'now that the new version of this standard event feature extraction algorithm of this physics group is finished, run the algorithm over this large set of events and store the results for quick access by everybody in that group'. Workload characteristics are discussed further in section 4.9.4.

2.3 Data handling problems and opportunities

The nature of physics analysis implies several problems and opportunities for physics data handling systems like the CMS data grid. These problems and opportunities have been extensively studied in the RD45 project and elsewhere [30] [22] [23] [12] [26] [15] [25] [21] [27] [28] [16] [20]. Some of the main points are as follows.

- Data volumes are huge, but the size of an individual data product is relatively small (1 KB-1MB).
- Workloads are dominated by reading.
- Given a sufficiently powerful and pervasive versioning mechanism, many sets of data product values can be treated as read-only after creation. This makes data replication much more tractable than in many other application areas.
- The stepwise refinement of algorithms and event selection functions leads to a workload profile where series of jobs are run over exactly the same input dataset, with each job in the series containing the next refined version of the code or of some parameter. So there are obvious opportunities for the caching-type optimizations when delivering input datasets to subsequent jobs.
- Because of the large data reduction factor in several important types of CMS physics analysis, the input datasets in such analysis efforts will represent very sparse subsets of the set of events over a period of detector running. The sparseness increases as an analysis effort progresses. This has important consequences for data handling. In particular, a system that uses a fixed partitioning of data product values over large files, and then stages all data needed by a job by staging completely all the large files containing the needed data products, will be too inefficient. To achieve the desired efficiency for the CMS workload, it is necessary to perform actions like copying a sparse subset of the data product values in some files into new files [15] [21]. For the foreseeable future, the baseline approach to these copying operations is that they are performed manually by CMS users or are initiated by CMS software components.

3 CMS offline hardware and software systems

To place the CMS data grid system and the grid project requirements in context, this section gives a brief overview of the major CMS hardware and software systems which together form the CMS offline system. Figure 3 shows the relation between the whole CMS offline system and other CMS systems.

3.1 Hardware systems

The CMS offline hardware will consist of computer installations arranged in a hierarchical structure as shown in figure 5. This figure was adapted from [10]. In year 2007, the offline hardware consists of a large central computing facility at CERN, at tier 0, five 'regional centers' at tier 1, and about 25 still smaller centers at tier 2. Below that, there are institute servers at tier 3 and physicist's desktop workstations at tier 4. At the moment of writing, only a handful of sites are performing CMS regional center tasks. Going towards 2007, the number of deployed centers will slowly increase [13].

Section 4.9.2 has some numeric details on the CMS hardware.

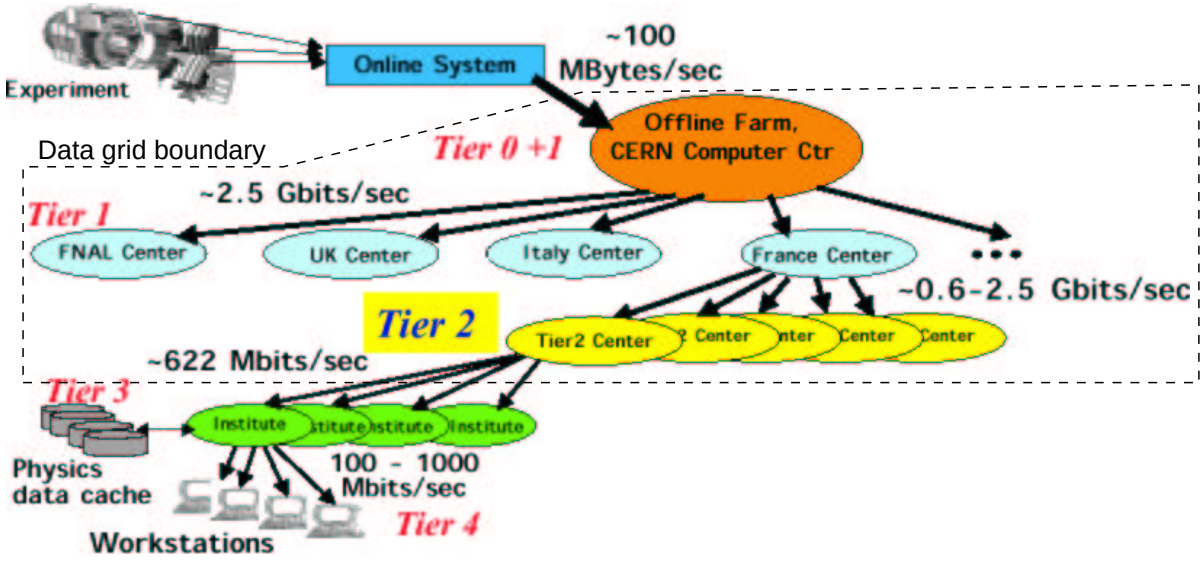


Figure 5: CMS hardware installations and network links (for 2007)

Around 2003, only some of the centers in figure 5 will already be in operation: the expected number is about 2 tier 1 centers and 6 tier 2 centers [13]. The scalability requirements for the grid components delivered to CMS by the end of 2003 are driven by the 20% data challenge milestone effort in 2004: the components need to scale to supporting a hardware configuration that has 20% of the projected 2007 capacity. The testbed configuration for the data challenge will exploit several shared resources in addition to the CMS-operated tier centers.

As a baseline requirement, the CMS data grid system need only use and manage system resources in tiers 0, 1, and 2. Not all of the resources in these tiers will be managed and used by the data grid system. The tier 0–2 site resources are common to the whole CMS collaboration. The tier 0–2 sites are assumed to be administrated well enough that any CMS physicist will trust that any data stored on these sites is labeled correctly, and will remain labeled correctly when updated. There is also trust that all certified CMS software is installed correctly on these sites, and that this software will reliably run to produce the requested result. From a trust standpoint all these sites are interchangeable, so that the CMS data grid system schedulers to not need to take trust issues into account when deciding between tier 0–2 sites. The tier 3 and 4 sites on the other hand are dedicated to a few users or to a single user: these will in generally be trusted only by specific users to perform some specific tasks correctly. Use of the tier 3–4 sites as common resources is outside the scope of the baseline CMS data grid system. However the tier 3–4 sites will be used to run physics analysis tools that will interact with the CMS data grid system, for example by submitting jobs to it, or by exporting data from it.

3.2 Software systems

Seen from the highest level, the CMS experiment has a long data processing chain that goes from detector signals all the way to plots in publications. The chain is not static, but the elements of it are developed and refined by physicists over time. Several software systems are involved in handling the whole chain. An important question is where exactly the boundaries between these software systems should lie. This document gives the following baseline answer for the boundaries of the CMS data grid system. The CMS data grid system handles the middle part of the CMS data processing chain, from the point when raw data flows out of the online system to the point when the intermediate results become small enough to be stored and handled by conventional tier 3–4 systems (figure 5). In physics data terms this latter boundary is the point where things like *ntuples*, *tags*, *DPDs*, and *histograms* appear. The proposed boundaries limit the complexity of the interactions that need to be supported by the CMS data grid system, and yield fairly narrow interfaces between the grid components in this system and the rest of the CMS software.

These are the main different offline software systems that CMS intends to operate. All of these combine CMS and non-CMS software components.

The **CMS data grid system** will support both CMS production and analysis and will seamlessly tie together resources in the tiers 0–2. The CMS data grid system is the most important system for the grid projects (GriPhyN, PPDG, and the EU DataGrid) in which CMS is involved. Whereas some other CMS software systems will also use grid technology, the CMS data grid system contains the grid components that should be targeted by the grid projects for which CMS is a 'customer'.

The **end-user physics analysis tools** are the software systems that the average physicist spends most time interacting with. These tools run on tier 3 and 4 systems, and are installed and maintained by local system administrators or by the end users themselves. Mobile computing devices forming a 'tier 5' might also run analysis tools with a limited functionality. Such tier 5 devices might also run lightweight analysis clients that connect to analysis servers running on tier 3 and 4 systems. The physics analysis tools are used for the interactive analysis on limited amounts of physics data (stored locally), with interactive cycle times in the second–hour range. These tools might also support advanced visualization and GUI functions. Because physicists spend much of their time interacting with end-used physics analysis tools, these tools are to physicists what text editors are to programmers: people get very used to doing things in a specific way with a specific tool, and tend to resist a forced switch to another tool. Some people prefer tools with GUIs based interfaces, others prefer command line or EMACS-style interfaces, and debates about which tool is best are vigorous. It is unclear at the moment whether the CMS physicists will all use the same monolithic analysis tool in 2003, will all use a single standardized set of analysis tools, or will all use different but overlapping sets of analysis tools. In the baseline vision of this document, physicists will generally interact with the CMS data grid system via special-purpose modules in their analysis tools, thereby obtaining the user interface they like best. These modules will in turn interact with the grid via a standard, narrow interface.

The **CMS idle CPU scavenger grid** is a grid software system which is separate from the data grid. The main purpose of the idle CPU scavenger grid is to make use of any otherwise idle compute power in CMS tier 3–4 systems and the CMS detector online compute farm. Otherwise idle compute power in the tiers 0–2 might also be absorbed by the CMS CPU scavenger grid. The use case that underlies the CMS idle CPU scavenger grid, absorbing otherwise idle compute power, exists widely in the grid community, so it is expected that no LHC-specific grid software development needs to be done by the grid projects, in order for CMS to successfully build and deploy its future CPU scavenger grid system. At the time of writing, CMS has not yet deployed a comprehensive idle CPU scavenger grid system on its tier 3–4 resources. Compared to the CMS executables running in the CMS data grid system, the executables running in the CPU scavenger grid will be written to be much less demanding in terms of the facilities offered by their execution platform. In particular the executables will do all their I/O in terms of local files, will not require large input datasets, and will not require access to 'grid-wide' metadata repositories or other services. These executables will be upgraded infrequently, maybe every few months, and can be distributed in the form of self-contained auto-installing packages. The CMS CPU scavenger grid will mainly run detector simulation codes, whose output will be uploaded at regular intervals into the CMS data grid system. The CPU scavenger grid will not be able to run all CMS detector simulation codes: many of these will need the more powerful services offered by the CMS data grid system to execute.

4 The CMS data grid system

This section describes the CMS data grid system that CMS intends to operate around December 2003, and the expected scaling towards 2007. To capture the expected level of complexity, the vision is sometimes worked out to considerable detail, even though some of these details are likely to be adjusted in future.

4.1 Software components in the CMS data grid system

The CMS data grid system contains a large number of software components from different sources.

1. The system contains **generic off-the shelf components**, like operating system systems and Internet protocol stacks.
2. The system contains **commercial software components selected by CMS**, for example a commercial database component.
3. The system contains **generic grid components**, as may be standardized by the Global Grid Forum (GGF), and as currently produced by projects like Globus [31] and Condor [32]. While the CMS data grid system relies on such generic grid components, it is not the primary role of the GriPhyN, PPDG, and the EU DataGrid to develop such generic grid components.
4. The system contains **data grid components** produced by the grid projects, GriPhyN, PPDG, and the EU DataGrid, for which CMS is a 'customer'. These components are more specifically devoted to the data-intensive science application requirements of high energy physics and of the other grid project customers.

   CMS itself might also produce some data grid components. It should be noted however that, in terms of manpower and deliverables, the CMS grid component implementation effort will be small compared to the efforts of the grid projects in this layer. Most of the grid-related manpower that CMS has available will be devoted to interacting with the grid projects as a customer, to work towards the identification, definition, testing, integration, and deployment of common data grid components in the grid projects.
5. The system contains **HEP-specific non-grid software components**, produced in HEP collaborative projects like GEANT 4 [34] and ANAPHE [35].
6. The system contains **CMS-specific non-grid software components**. The most important of these are CMS **physics algorithms** which encode CMS's knowledge about its detector and the way to do physics with it: by 2006 these algorithms will represent many hundreds of man-years of development effort. Another important component is the **CMS software framework** in which these physics algorithms run. The currently deployed CMS software framework is called CARF, the next major version is called COBRA. The CMS software framework isolates the grid components from many of the details of CMS physics algorithm invocation and CMS physics data handling. The grid components in turn have to isolate the CMS software framework from many of the details of grid resource management, distributed job execution, and wide-area data handling.

The main focus of this document is on the exact relation between the data grid components and the CMS-specific software components. The interfaces between these components are discussed in considerable detail. For many interface calls, quantitative aspects like the calling frequency are specified. From these numbers, some scalability requirements for the grid components can be derived.

4.2 Storage and handling of data product values

Note to CMS Objectivity/DB users: This section describes a long term data handling vision that would, among other things, support use of the current Objectivity product. However this long term vision is not coupled very strongly to current Objectivity details. The mapping of this section to current CMS data handling terminology is as follows:

This section	Current CMS data handling terminology
File	Objectivity database file or ZEBRA file
Data product	Object or record
A CMS file catalog	An Objectivity federation catalog file and boot file in the local area, maybe combined with some CARF metadata attached to that catalog
Object persistency layer	Objectivity client library and some parts of CARF, or ZEBRA library
Object persistency server	Objectivity AMS server

A **data product value** is a physical representation of the value of a data product. Data product values can be stored, and read into the memory space of CMS executables. CMS has a very specific architectural vision for the storage and handling of data product values. This vision is illustrated in figure 6. The main points are as follows.

Data product values are always stored in **files**, usually there are many related data product values in a single file. For a single data product, its data product value can exist in many files at the same time. The reading and writing of data product values from and to files is handled by an **object persistency layer** inside each CMS grid executable. This layer is part of the CMS software framework. Inside the CMS data grid system, in the time frame now-2003, the CMS software components are responsible for mapping and re-mapping data product values to files. The grid components needs not be aware of the fact that some files will hold subsets of data product values which are also present in other files.

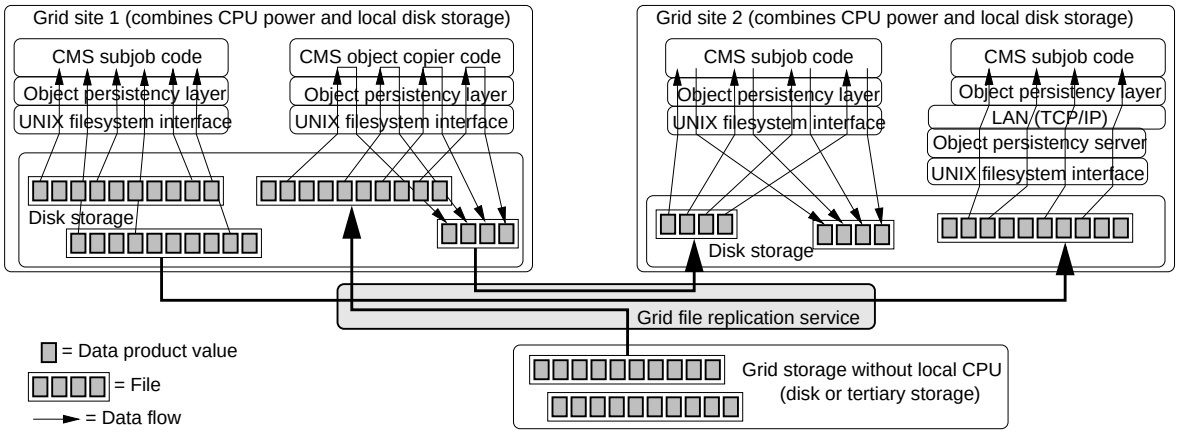


Figure 6: Storage and handling of data product values in the data grid system

The data grid software components are responsible for wide-area data handling and replication in terms of files. The services offered by the grid components provide **logical grid file** and **physical grid file** abstractions. A logical grid file is a file that exists in the grid in a location-independent way. It has one or more physical grid files associated with it, each of these is a physical representation, stored at a certain location, of the logical grid file. The grid software components provide a **grid-wide file replica catalog service** which maintains the mapping from logical to physical files. Not all physical files of a logical file are necessarily identical all of the time: due to lazy replication policies some of the physical files may be 'out-of-date' representations.

CMS currently uses two file types to store data product values in production: Objectivity/DB database files produced by a persistency layer based on Objectivity, and ZEBRA files produced by the Fortran ZEBRA library – this latter type is being phased out. By the end of 2002, CMS expects to select a final object persistency solution to be used from then onwards past 2006. However at some time during the lifetime of the experiment (up to at least 2030), external developments might force or motivate CMS to change again from the persistency solution selected in 2002.

Irrespective of specific implementation of the object persistency layer, the following architectural choices and constraints apply for the storage and handling of data product values in files in the CMS data grid system, for the time frame starting now and extending past 2006.

1. Data product values are stored in logical grid files. Each logical grid file will contain one or more (maybe a huge number) of data product values. Support for file sizes larger than 2 GB by grid components is desirable, but not absolutely needed in the time frame now-2003.

2. The choice and implementation of the persistency layer, which is responsible for reading objects out of files and writing objects into files, is always under complete control of the CMS software team: the grid projects

are not involved and also cannot impose design constraints for this layer. The grid components do not need to manipulate or interpret the file contents.

3. The grid components need to maintain a global view of which logical files exist, and where all physical files are.

4. The CMS components will call on the grid components to create and delete logical files, and perform wide-area replication actions on (sets of) physical files.

5. Data import/export operations between the CMS data grid and outside systems happens in terms of logical files.

6. For any CMS executable running on a grid site, the object persistency layer and the CMS software framework only implement, as a baseline, access to physical grid files which are local to that grid site. (The current Objectivity-based layer does provide the option of implementing access to physical files at other grid sites, but the data transport protocol that is used is too inefficient to be of much use in the wide area. It is therefore not expected that this option of remote file access will be used much, if at all, in production by CMS grid executables.)

7. File access by the CMS object persistency layer is always performed on physical grid files via the UNIX filesystem interface, using regular POSIX I/O calls. In particular a grid-provided file storage layer that would require the object persistency layer to be linked with special 'grid versions' of the POSIX open(), read(), . . . calls is incompatible with the CMS architectural principles, because CMS might choose in future to use commercial software components in its persistency layer that do not allow such re-linking with special open() versions. (The current Objectivity-based persistency layer does allow for some degree of re-linking, but a future product that CMS could consider might not.)

8. Two cases for file access by the persistency layer are possible.

   - First, the I/O calls could be done directly from the executable containing the CMS physics algorithm, on a filesystem mounted on the machine that runs the executable. Note that the mounted filesystem is not necessarily a local disk of that machine, it might need to be a much larger filesystem local to the site but not local to the machine. Access to files on a large filesystem local to the site, but not local to the machine, is needed in particular in the CMS use case of doing 'random' type access to sets of 'pileup events' with a size of 1 GB – 1 TB: such sets are simply too large to stage to the local disk of every machine in a site.
   - The second case of doing the file access is shown in the rightmost part of figure 6: The executable connects to a CMS-provided object persistency server via the site LAN, and the actual filesystem I/O calls are done by object persistency server, on a filesystem that is mounted on the machine that the persistency server runs on.

   CMS does not guarantee that both forms of access will always be implemented as equivalent options by all its persistency layers. Rather, in both the short term and the long term, CMS requires that both these forms are efficiently supported by the grid components in the CMS data grid system. Supporting only one of these forms in an efficient way is not acceptable, as this would constrain the choices that CMS has for evolving its persistency layer.

9. CMS has a strategy of moving code to data. However some of the time, data will also have to move to a location where code is run.

   The preparations for running a CMS executable on a site may involve creating local physical files by replicating files from other sites, or even (for efficiency reasons) first invoking a CMS object copier tool on a remote site to copy selected data product values into a new, smaller logical file, and then creating a physical copy of this new, smaller file on the local site. Both these options are illustrated in figure 6. For the foreseeable future, the baseline approach to data product level copying operations is that they are performed manually by CMS users or are initiated by CMS software components.

   Not all data used by a CMS executable will necessarily be available locally at the start of this executable. During the running of a CMS executable, the executable, or a CMS object persistency server used by the executable, might also call on the grid to resolve a logical grid file that is needed by the executable to a close physical grid file. If a close enough physical file is not found, a file replication operation might be initiated to create one. The CMS executable will be idle while the file is being replicated over. It is not expected from

the grid components that they offer services to reclaim such idle CPU time. In stead CMS is responsible for making the performance tradeoff, on a file to file basis, between using dynamic staging of a file which introduces idle time on the CPU that is allocated to the running of the CMS executable, and using pre-staging of the file even while it is not known for sure that the executable will need it. CMS might at one point in the future seek help from the grid projects in developing heuristic stage-ahead strategies to hide some of the latency of dynamic staging.

10. The CMS executables will use two different ways of mapping logical grid file names to the names and locations of physical grid files.

    - The first way is by doing a lookup in the grid-wide file replica catalog service.
    - The second way, which also requires some support from the grid software components, is via a **CMS file catalog**. A CMS file catalog is a special persistent data structure, existing as several logical grid files, which is maintained by CMS software components in the data grid system. Apart from references to physical grid files, the file catalog can also contain specialized indexing metadata that allows fast access to the data product values in these physical grid files, bypassing much of the process of generating a logical grid file name and then having it mapped to a physical grid file. Many CMS file catalogs will be present in the CMS data grid system.

      For the grid components in the CMS data grid system, the support requirements for CMS file catalogs are as follows. Because the CMS file catalogs contain physical file names and locations, the grid components might need to update such file catalogs when moving or deleting physical files. These constraints on the data grid components will be expressed as part of CMS-set replication consistency management policies. These policies may imply for example that, before a physical grid file is deleted, a CMS grid software component needs to be called to ensure that the information about this physical grid file location is removed from one or more CMS file catalogs. Policies might also require that, when a new physical grid file is created at some site by automatic replication, a CMS grid software component is called to register the presence of this file into one or more CMS file catalogs.

    CMS does not guarantee that both above forms of obtaining name and location information about needed physical files will always be implemented as equivalent options by all its persistency layers. Rather, in both the short term and the long term, CMS requires that both these forms are efficiently supported by the grid components in the CMS data grid system.

11. For the duration that a CMS executable might be accessing a particular physical file, this file needs to be 'pinned', that is protected from moving, deleting, or overwriting by grid components. The grid components are responsible for maintaining the pinning status of each physical file. File operations initiated via the grid components which would conflict with the pinning status need to be delayed or refused. CMS executables will require an interface to create file pins or inquire about the file pinning status. Note that multiple CMS executables might have a physical file pinned at the same time: it is the responsibility of these executables, not the grid components, to coordinate any possible update operations that they do on the file.

12. Many physical files in the grid will exist under CMS-set replication and consistency policies that allow grid components like 'garbage collectors' or 'load balancers' to automatically replicate or delete these files. Such automatic file management actions will be always be constrained by file pins, and performing them will also often require calls to CMS software components so that CMS file catalogs or other types of metadata can be updated.

4.3 Data replication model

As mentioned above, the grid components in the CMS data grid system provide logical and physical grid file abstractions. The grid software components provide a grid-wide file replica catalog service which maintains the mapping from logical to physical files. Not all physical files of a logical file are necessarily bytewise identical all of the time: due to lazy replication policies some of the physical files may be 'out-of-date' representations of the logical file.

Data replication in the CMS data grid system is not done on a file-by-file basis, but in terms of coherent groups of files called **file sets**. A file set is a set of logical grid files. A single file can be in many file sets: file sets can overlap each other and be contained in each other. A grid-wide **file set catalog service** is used to maintain information about all file sets that exist in the grid. This service is provided as a software component by the grid projects. The contents of the catalog, the file sets themselves, are created and updated by CMS software components that

interact with the file set catalog service. Each file set entry in the catalog has a unique name that is assigned by CMS components. Apart from a list of logical file names, the file set entry also records a CMS-defined **consistency management policy** that will govern replication actions on the file set.

Examples of file sets are: 1) a single file, 2) all private files of some user, 3) the Objectivity database files containing all raw data products for some period of detector running, combined with several database files with indexing metadata, which are needed to access the raw data products in these files, 4) the files which represent a particular CMS file catalog.

Both private and public file sets will be defined. For public file sets, CMS production managers and physics groups will often create metadata on a per-file set basis. This metadata can be searched by CMS physicists to find the file sets they need. CMS might make use of grid level metadata keeping services to make maintaining this metadata easier. However it remains the responsibility of CMS to encode and manage this metadata, and it will be CMS production managers who decide to what extent they will adopt common grid metadata management services.

Examples of operations on file sets are: 1) replicate the contents of the file set to some grid site, 2) export the contents to a site outside of the grid, 3) clone the file set contents to create a new 'private' copy of the file set under a new name, 4) query if site X has a replica of file set Y, and whether this replica is still 'current', with the concept of currency being defined by the consistency management policy of the file set, 5) find all grid sites which have a 'current enough' copy of some file set available on local storage.

It is envisaged that a grid-level file replication service can be used to perform operations like the ones above on file sets. This service has the perform replication actions within the constraints of several different CMS-defined consistency management policies. These policies can be quite complex, encoding specific freshness constraints between the replicas of different files in a set. Examples of policies are: 1) this file set is local to a single grid site and never replicated, so each of the logical files in the set has exactly one physical file associated with it on that site 2) this file set consists of read-only files, so files can be replicated always in any order, 3) the files in this set are read/write files, but there is a clearly defined master site where all the file writes are done, new or changed files can be replicated to other sites at any time, 4) there is one master site where file updates are done, but replicas can only be made at particular times when a CMS component marks the set of files at the master site as stable and mutually consistent. The exact policies that CMS needs support for are not yet defined at this moment, the above examples merely indicate the expected level of complexity from now till 2003. In theory, the CMS components could define overlapping file sets in the file set catalog and attach mutually conflicting consistency management policies to them: it is the responsibility of the CMS software components to ensure that this never happens.

All CMS file replication policies for the foreseeable future will share some common characteristics. For all of these policies, each individual logical file is declared to be either a) read-only or b) to be read/write with a single well-defined 'master site' that is responsible for all the writing. Note that the master site can change over time.

4.4 Job model

Physicists get work done on the CMS data grid by submitting **jobs** to it. A CMS grid job generally consists of many **subjobs**. By 2003, some very large jobs might even contain hundreds of thousands of subjobs. Each subjob consists of the running of a single **CMS executable**, with a run time of seconds up to hours. The executable runs as a UNIX process. The process may be multi-threaded, but in general the threads together will only use the CPU power of a single CPU. The number of CPUs to be allocated to a subjob on execution is one of the CMS-supplied parameters of the subjob.

There are many different types of subjobs. For example, some subjobs execute analysis code specified by the physicist when the job was submitted. Some execute standard physics algorithms taken from a software repository, to create new data product values that might potentially be shared by other physicists. Some extract selected data products from a large file set into a smaller file set. Not all of the subjobs of one job will necessarily run in the same security context: for example subjobs which contain only certified data extraction code or certified physics algorithms might run in a security context that allows them to create or update file sets that are shared among many grid users.

CMS subjobs do not communicate with each other directly using an interprocess communication layer like MPI. In stead all data is passed, asynchronously, via file sets. A subjob generally has one or more file sets as its input, and will generally create or update at least one file set to store its output. Subjobs can add new files to output file sets while they are running. Some of the file sets used in a job may be temporary sets of files, existing only to pass data within the job. Within a job there is a sometimes complicated, but always acyclic, data flow arrangement between

subjobs. This arrangement can be described as a data flow graph in which file sets and subjobs appear alternately. The data flow arrangement inside a job is known to the grid components, in particular to the grid schedulers and grid execution services, so that they can correctly schedule and sequence subjob execution and the data movement within a job.

Several subjobs may have the same file set as input. It also is possible that several subjobs, subjobs which can potentially run concurrently according to the data flow constraints specified for the job, are using a single file set as a joint space to place their output in. In general this requires that all the files in this file set have their master copies on the same site, with these subjobs also running on that site. The subjobs will coordinate their I/O actions on the contents of the file set using concurrency management services offered by the CMS persistency layer.

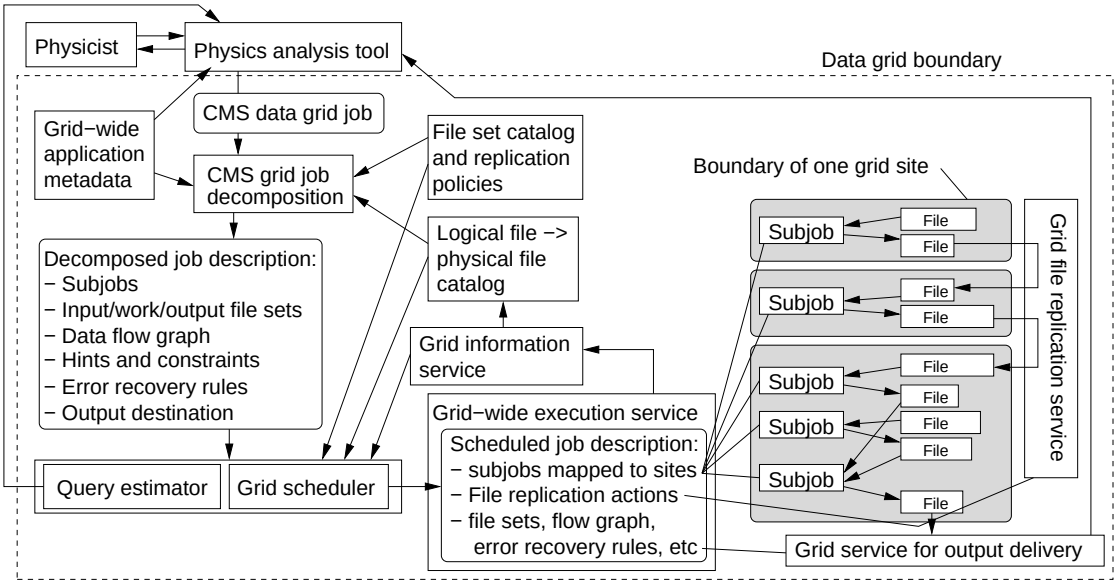


Figure 7: Creation and processing of a single CMS data grid job

Figure 7 shows the model for the creation and processing of a single CMS data grid job. The job is created by a physicist using a physics analysis tool, and then submitted to the grid. The job description is passed to a **CMS grid job decomposition service**, which is a grid component provided by CMS. This service decomposes the job into subjobs, and creates the job data flow graph. Often a job can be decomposed in many ways: the decomposition service interacts with several catalogs and other grid services to determine in which way the job should be decomposed to achieve good efficiency given the current state of the grid.

The resulting **decomposed job description** contains a logical, location independent, specification of all actions that have to be performed by the grid to execute the job. The subjobs in the decomposed job description are not yet mapped to any grid site, and the input, output, and data handling are specified in terms of file sets of (location-independent) logical grid files. It is the job of the **grid scheduler** to optimize the job by taking the decomposed job description and mapping the subjobs to specific grid sites. The grid scheduler also generates any necessary file replication actions to ensure that the subjobs can access (physical replicas of) their specified input, output, and working file sets on local site storage via their persistency layer. Note that while running, the subjobs might also interact with the grid to dynamically stage additional files, whose identities were not visible in the decomposed job description.

The decomposed job description might also be delivered to a **query estimator**, which computes an estimate of the job runtime and resource usage when it would be scheduled. It is expected that an estimation of CPU and I/O resource usage can be fairly accurate: the query estimator can use estimates about subjob CPU and I/O profiles which are supplied in the decomposed job description. Estimating wallclock runtime will be more difficult, so a **job progress meter** service and the ability to abort a pending or running job will likely be useful. When submitting a job, the user may include several **hints** to the grid to help it in optimizing resource usage. An example of a hint would be “several other jobs using the same input data set will be submitted in future”.

The grid scheduler transforms the decomposed job description into a **scheduled job description**, which is passed to the **grid-wide execution service**. An important part of the job description is a set of *error recovery rules*, which are supplied by CMS as part of the decomposed job description. These rules can be used by the execution service to recover automatically from certain classes of hardware and software failure. An example of an error recovery rule

is: ‘if the machine that is running the subjob at point X in the data flow graph crashes, then one can automatically roll back to a consistent state, representing the state before the subjob was started, by calling the CMS-provided cleanup script with the name Y and parameters Z’. Because of the limited operator manpower compared to the number of hardware and software components that may fail, CMS requires the grid-wide execution service to be able to support quite complex automatic error recovery strategies.

Often, the bulk of the CMS **job output** remains inside the grid, as a new or updated file set. However, one or more subjobs in a CMS grid job might also deliver output (usually in the form of files) directly to the physics analysis tool that started the job: output delivery is asynchronous and should be supported by a grid service.

Besides jobs, users and system operators can also submit **commands** to the grid, to achieve something that cannot be expressed as a job. An example is the command create a replica of some file set at a particular location and to keep it there for the next month.

4.4.1 Job and command submission interface

To submit jobs and commands, the CMS physics analysis tools, usually running on local workstations outside the grid, will use a grid project defined job or command format and call on a grid project defined API. Some job or command descriptions may include very large pieces of data, for example compiled code and large algorithm parameter sets. These could be represented as the names of files on the local filesystem of the process doing the API call.

Some jobs which are submitted may be waited for interactively, with the CMS tool keeping an ‘open line’ to the grid to receive information about progress and termination. Other jobs may be run in batch mode, with progress and termination being monitored using separate tools.

4.4.2 Subjob overheads

Subjobs of course have startup and initialization overheads, but these will generally be negligible if the size of the data products read by the subjob is larger than about 100 MB. Subjob granularity will generally be at this level or even more coarse, so that startup overheads are kept relatively low.

There are various potential I/O overheads when subjobs read input data product values from files. A complete discussion of this topic is beyond the scope of this document, some sources are [15], [22], and [23]. Some specific considerations follow. It is desirable, and often possible, to store data product values in files in such a way that the reading of product values from files produces sequential access patterns on the disk hardware [15] [17]. The large size of job input data sets, and the ‘streaming’ sequential nature of data product value access, imply that in-memory caching of product values is not a very important optimization technique in the CMS data grid. The I/O overheads in accessing data product values are mainly determined by the storage system hardware. The use of Objectivity/DB as a persistency layer does not introduce overheads significantly different from those of other persistency layer implementations that provide cross-platform format compatibility. Indexing overheads to find product values inside files will always be relatively small.

4.5 Weak forms of equivalence between datasets

One cannot overestimate the cost (in time, manpower, hardware resources, wrong results and loss of credibility) that could result from false “equivalences” among datasets. In many cases CMS physicists will insist on byte-wise equality between subsequently delivered datasets.

In some cases, weak forms of equivalence between a dataset that is requested and the one that is delivered are permissible, especially if a weakly equivalent dataset can be delivered much faster. Examples of these cases are the selection of random initial event sets an analysis effort, and the selection of sets of simulated events. However, research by the grid projects into computer support for such data selection with weak forms of equivalence is not being requested by CMS.

4.6 Requirements for the handling of platform differences

A platform is a combination of hardware, OS, compiler, libraries, etc. used to execute job or algorithm code. Platform differences may result in small deviations in job output. The CMS data grid will have to include some

facilities that assist physicists in studying and handling platform differences. In particular it should be possible to constrain job scheduling in such a way that subsequent jobs are guaranteed to be run on the same platform.

4.7 Security and resource sharing policies

The code and data in the CMS data grid need to be protected against unauthorized access. There are two main cases. First, all data needs to be protected from access by people outside the CMS collaboration. Second, when doing development work, a physicist will consider all intermediate results to be private. Private data needs to be inaccessible, even to some degree invisible, for others. End results might be 'published' by extending the access permissions to a group of people or to the whole collaboration. Access logs and workload statistics that are gathered for grid scheduling and monitoring also need to be protected against unauthorized access.

Some facilities are needed to ensure that grid resources are shared fairly. The grid schedulers have to implement allocation policies that ensure reasonable sharing of resources amongst concurrently running jobs. Query estimation services will play an important role because they allow users to avoid the unintentional abuse of resources. An accurate accounting of resource usage is needed, to drive the social feedback loop that ensures fair sharing inside the CMS collaboration. At the moment of job submission the grid does not need to implement very fine-grained constraints on resource usage by individuals, though some active protection is needed against the accidental or intentional execution of jobs or commands with a 'denial of service' effect. Some tier 1 and 2 centers might have policies that reserve the use of certain local resources to a subset of all physicists in CMS.

4.8 Interface for the invocation of CMS executables on grid sites

A CMS subjob consists of the running of a single CMS executable on a grid site. The executable runs as a UNIX process. The process may be multi-threaded, but in general the threads together will only use the CPU power of a single CPU. The number of CPUs to be allocated to a subjob is one of the CPU-supplied parameters of the subjob.

A CMS executable might incorporate user code that was finished only seconds before the submission of a grid job or command that requires the use of this code. The exact way in which such code is delivered to the grid still needs further study. Some options are: source, shared library, complete CMS executable, and an abstract identifier that can be submitted to a compilation service, together with a platform identifier, to obtain a platform-specific CMS executable.

CMS executables will occasionally dump core, hang, terminate with nonzero exit status, or produce unexpected error messages on their standard output and standard error streams. The grid components invoking the executables need to be written to cope with such situations, both by supporting automatic error recovery strategies, and by giving CMS grid users or operators the ability to diagnose and decide if no automatic strategy applies. When a CMS executable crashes or is aborted by the grid components, the grid components might need to run a CMS-provided cleanup tool, for example to delete temporary database files and release temporary database locks.

To prevent accidental or intentional data corruption due to the execution of code that is developed dynamically by end user physicists, any CMS executable that incorporates such code needs to be run in a 'sandbox' type environment as far its UNIX filesystem I/O privileges are concerned. The code should not be able to read, modify, or delete files containing private, unpublished data of other users or groups (see section 4.7). The code should also not be able to modify or delete files containing public or published data. The I/O sandbox can be set up by the grid just before invoking the executable. The CMS object persistency server, if used on a grid site, will have the ability to set up an internal sandbox file I/O environment for any executable that contacts it, after obtaining the necessary information from the grid security layers. The server itself has full I/O privileges for all files.

A CMS executable generally needs to be invoked by calling a CMS tool which first sets up the right runtime environment for the executable, and then runs the executable itself. The CMS software runtime environment will remain under development from now throughout the lifetime of the experiment, and this makes high flexibility in setting up the runtime environment for CMS executables critical. As a simple use case, two different CMS executables running at the same time on the same CPU might require different versions of standard shared libraries like glibc. CMS currently uses a tool called SCRAM [33] to create the required flexibility in the runtime environment, SCRAM does this by dynamically configuring the environment variables (in particular the library path) before the executable is invoked. Because of the complexity of the CMS software environment, it is unclear whether SCRAM could be replaced by a possible future common grid job environment management tool. Taking the current use of SCRAM as a starting point, the following requirements for the grid components and the operation of CMS data grid system sites are obtained. First, CMS needs to have the ability to configure dynamically any given part of the

execution environment except for the UNIX kernel and kernel services. As a baseline, the runtime environment setup tool that invokes the executable must have a recent mirror of the CMS production software repository available on a locally mounted filesystem. This software repository will have a size of at least a few GB, containing elements like shared libraries, configuration files, and setup scripts. The synchronization frequency of the mirror should be at least once every day. Some CMS grid subjobs might require the availability of a very 'fresh' mirror of the repository, such constraints will be expressed as part of the job description and the grid scheduler needs to take them into account when mapping subjobs to sites.

4.9 Quantitative aspects

This section provides some quantitative estimates for data, hardware, and jobs. These estimates form the scalability requirements for the CMS data grid system. Most of these estimates were taken from [13], and are estimates for the year 2007.

The scalability requirements for the grid components delivered to CMS by the end of 2003 are driven by the 20% data challenge milestone effort in 2004: the components need to scale to supporting a hardware configuration that has 20% of the projected 2007 capacity.

4.9.1 Product sizes and processing times

Table 1 gives the estimated sizes and processing times for the most frequently occurring data products. See section 2.1 for an explanation of the raw data products for an event. The data products derived directly from raw data are generally called **ESD** (event summary data) products by CMS physicists, those obtained from ESD products are generally called **AOD** (analysis object data) products. Less frequently occurring data products are expected to have largely similar characteristics. The *time needed to derive* in this table is the running time of an algorithm that has the product as output; and, the *time needed to process* is the running time of a job that has the product as input.

		CPU power needed to		2007 CPU time needed to	
Product type	Size	derive	process	derive	process
All raw products for one event	1 MB	–	3000 SI95s	–	15 s
One ESD	500 KB	3000 SI95s	25 SI95s	15 s	0.125 s
One AOD	10 KB	25 SI95s	10 SI95s	0.125 s	0.05 s

Table 1: Estimates for CMS data product sizes and processing times

The times in seconds are based on the estimated capacity of a single CPU used around 2007, which is 200 SI95.

Starting in late 2006, CMS expects to store raw products for 10^9 events per year. Around 2006 it also expects to run simulations of events in the detector, storing simulated event data at a rate of $5 * 10^8$ events per year, with each taking 2 MB of storage. See section 6.1 for information about the scale of the current data handling effort in CMS.

Creation of ESD and AOD products is partly a *chaotic* activity, in which new products computed with new algorithms and parameters could be requested at any time by some specialized physicists. However, there are also *production runs*, which are highly structured data product creation efforts run by system managers. In production runs, the system managers will usually run grid jobs to compute and store two specific ESD and AOD products for every event in a large set. Such a set generally is the set of all events taken over a period of weeks or months, or a well-defined subset thereof. Production runs are expected to use a significant fraction, probably more than half, of all grid resources in the year 2007. In later years, as faster hardware makes resource constraints less of an issue, the chaotic activity is expected to take up larger and larger fractions.

4.9.2 CMS grid hardware capacity

The CMS grid hardware will consist of computer installations arranged in a hierarchical structure as shown in figure 5 of section 3.1. Figure 5 also shows recent estimates of the network link capacities between the sites in 2007. It should be stressed that the actual link capacity available to CMS in year 2007 cannot be estimated very accurately, mainly because of uncertainties about long-term developments in the international telecom market.

As a baseline requirement, the CMS data grid need only use and manage the system resources in tiers 0, 1, and 2. The institute servers and workstations will be used for data storage and processing outside the grid, and also to

prepare and submit grid jobs. Table 2 gives hardware capacity estimates for individual tier 0–2 centers. Note that 'tape' in this table could also be another tertiary storage medium, depending on technology developments. The archival storage capacity will grow over time, in order to satisfy the constraint that at all raw data needs to be kept forever.

Tier	CPU capacity	Nr of CPUs	Active tape	Archival tape	Disk
0 (CERN)	455,000 SI95	3000	1540 TB	2632 TB	796 TB
1	105,000 SI95	750	590 TB	433 TB	313 TB
2	26,000 SI95	180	none	50 TB	70 TB

Table 2: Estimates for 2007 CMS hardware capacity needs, taken from pages 2–4 of [13], and an estimate of the numbers of installed CPUs needed to fulfill the CPU capacity needs. The disk space used as a cache for the active tape is included in the disk column above ([13] does not count this disk space). The year 2007 represents the first full year of CMS detector running, so the above estimates reflect the capacity needs to store and process one full year of CMS data. To obtain the total installed hardware capacity from these needs, they need to be multiplied with overhead factors to take into account allocation inefficiencies and downtime for maintenance [13]. The tier 0 numbers in this table are the total capacity needs for CMS offline data processing at CERN

4.9.3 Hardware failure characteristics

The CMS data grid needs to provide fault tolerance services which take the following into account.

CMS expects to buy most of the grid hardware described in Section 4.9.2 on the commodity market, and expects to operate it without expensive round-the-clock service contracts. CPU servers can easily be down for days. Much of the disk capacity might actually be on hard disks inside the CPU server boxes, not on more dedicated disk servers. Most of the disks will not have backups and might not even use a high RAID level. The disks can also be down for days or suffer failure with no hope of data recovery.

CMS also expects to configure and operate a small amount of all hardware as high reliability and availability systems which have round-the-clock operator support. Such systems will be located at the tier 0 site and probably also at all tier 1 and tier 2 sites. The grid can use these systems to store metadata, catalogs, and transaction state.

4.9.4 Workload characteristics

Multi-user physics analysis workloads have a complicated structure. The properties of the workload are determined by three major interacting factors: the methodology of high energy physics as an experimental science, the way in which physicists collaborate and divide their work, and the need to maximize the utility of the available computing resources. The workloads have a mix of **production** and **analysis** activities. In the terminology of this document, a production activity corresponds to the submission of a grid job which contains at least thousands of subjobs and creates a large set of output files. The completion of such a job might take days or even months. Analysis corresponds to the submission of grid jobs by individual physicists – this activity is much more 'chaotic' than production, with jobs generally having a runtime of a few hours.

This document takes the HEPGRID2001 model [14] as the baseline model for the workload that needs to be handled efficiently by the CMS data grid system around 2006. Some statistics about the HEPGRID2001 workload are in figure 8, for further details this document refers to [14]. The HEPGRID2001 workload model is a model in terms of 'virtual' data product access, not a model in terms of file sets. A workload model at the level of file sets can be generated by extending the HEPGRID2001 model with a simulation of the CMS mapping of data products to files sets. The creation of such an extension is a planned activity. It should be noted however that the CMS understanding of its future strategy for mapping data products to files could evolve considerably over the next few years, based on experience with the future data analysis activities of individual physicists.

It should be noted that the HEPGRID2001 workload represents a *lower bound* for the number of 'chaotic' analysis jobs submitted by end user physicists. Section 4.9.5 gives both lower bounds and values needed to support very high levels of 'chaotic' interactive use. The MONARC project [11] is another reference for models of grid workloads.

Time span covered by workload generator in the HEPGRID2001 model	1 year
Number of physicists submitting jobs	500
Number of jobs in workload	341/day
Average size of a job input set	10^7 products
Average size of a job input set	1.3 TB
CPU capacity needed to analyze requested products in jobs	960,000 SI95
RAW products requested by all jobs	$4.5*10^9$/year
ESD products requested by all jobs	$3.0*10^{11}$/year
AOD products requested by all jobs	$9.4*10^{11}$/year
Average number of times that a single product is requested	40
CPU capacity needed to derive all requested products once	433,000 SI95
Different data products derived at least once	31/event/year
ESD products derived if all derived only once	$4.3*10^9$/year
AOD products derived if all derived only once	$2.7*10^{10}$/year
Size of RAW products	1000 TB/year
Size of ESD products derived if all derived only once	2166 TB/year
Size of AOD products derived if all derived only once	269 TB/year

Figure 8: Exact statistics for the workload generated by the HEPGRID2001 workload generator, adapted from [14].

4.9.5 Other parameters

This section estimates several other parameters important for the grid. For every parameter, the first value given is the expected value that needs to be minimally supported for the data grid system to be useful to CMS. The second value, between parenthesis, is the expected value that is needed to support even very high levels of chaotic use by individual physicists.

- Number of users: 500 (2000)
- Number of simultaneously active users: 100 (1000)
- Number of jobs submitted per day: 250 (10,000)
- Number of jobs being processed in parallel: 50 (1000)
- Number of events visited in a job: 1–10^9 (1–10^9)
- Job turnaround time: 30 seconds (for tiny jobs) – 1 month (for huge jobs) (0.2 seconds – 5 months)
- Data product value size: 1 KB – 100 MB (50 byte – 2 GB)
- Data products uploaded: 10^{10}/year (10^{11}/year)
- Frequency of bulk data upload operations: 5/day (50/day)
- Frequency of upload operations of a single file: 500/day (20,000/day)
- Number of file sets that serve as input to a subjob: 0-10 (0-50)

In general, CMS physics analysis workloads will expand to fill all available hardware capacity. Given this, and combining the estimate of about 250 jobs per day with the available capacity, the following estimates are produced. In these estimates the jobs fill their expected available share of CPU capacity, but not yet their expected available share of local area disk I/O capacity.

- Average number of data products accessed by a job: 10^7 (250,000)
- Average size of the data set accessed by a job: 1.3 TB (30 GB)
- For every data product that is ever computed, the average number of times that it is accessed in the workload: 40

Note that the three parameters above have very wide value distributions, so mere averages are not that meaningful when taken on their own. The output of the HEPGRID2001 workload generator can be used to obtain an indication of the value distributions of these parameters.

4.9.6 Expected evolution of parameters

The system requirements for year 2007 will evolve over time to track predictions of how much hardware capacity can be economically available in year 2007.

From year 2007 to at least year 2030, throughout the lifetime of the experiment, the required and realized system capacity will definitely increase to exploit price/performance improvements in hardware. For CMS, increased capacity will mean an increased ability to study physics effects, so increases will always be sought. The CMS grid architecture should thus allow for scaling of at least a few orders of magnitude beyond the year 2007 storage and CPU capacity requirements.

5 Requirements for the grid projects

This document defines the current requirements for the grid projects that CMS is involved in as a 'customer'. It should be noted that these requirements are based on the current CMS vision of its grid system, and this vision will certainly evolve over the next few years, partly as a result of the interaction with the grid projects. At the highest level, the requirement is simply that the grid projects should deliver software components to CMS which can be used by CMS in the construction of the CMS data grid system.

Concerning the grid components which are to be created by the grid projects, and delivered to CMS from now until the end of 2003, section 4 defines many requirements, and, even more importantly, many architectural constraints which these components need to take into account. An exact specification of the components to be delivered is beyond the scope of this document. The creation of such exact grid component specifications is considered to be joint future work between the grid projects and their customers, with the grid projects taking the lead in this effort.

The remainder of this section discusses a number of high-level requirements issues.

5.1 Important CMS milestones

The major CMS software milestones affecting the grid projects are the 'delivery of baseline core software' milestone for December 2002, this includes the choice and integration of grid components into the baseline software, and the '20% data challenge' milestone, for which the work starts in January 2004 with milestone completion in December 2004. The 20% data challenge includes the full range of distributed operations required for the analysis of CMS data under realistic conditions as occurring during LHC operation around 2007. The challenge does not necessarily require the commissioning of 20% of the computing capacity dedicated to CMS, since common time-shared facilities, such as the T0 prototype center at CERN, will be exploited. In connection to this challenge, the Grid components delivered to CMS by the end of 2003 need to scale to supporting a hardware configuration that has 20% of the projected 2007 capacity.

5.2 Division of labor

The following summarizes and expands on the division of labor between CMS components and grid components, for the time frame now-2003, as discussed in section 4.

1. Tasks for components provided by the grid community and grid projects.

    - Basic management and access interfaces for grid resources such as storage systems, CPUs, and the network.
    - Queuing of grid jobs, grid job execution management, integration with local site job submission systems.
    - Distributed job scheduling: optimize job execution by efficiently allocating subjobs to sites, moving code to data if possible, by taking into account factors like data location and site loads, and by generating efficient data replication actions to pre-stage data when necessary.
    - Error recovery services during job execution, which are configured with CMS-provided error recovery rules and scripts.
    - Resource management, monitoring, accounting tools and services.
    - Query estimation based on a decomposed job description and the current state of the grid.
    - Efficient wide-area data transfer in terms of files.
    - Access by CMS executables to physical grid files on site-local disk systems via POSIX calls on the regular UNIX filesystem interface.
    - File catalog services mapping logical to physical files.
    - File set catalog services.
    - File replication services in terms of file sets with the ability to implement CMS-configured consistency management policies.
    - Data management services: services to configure and maintain backups or mirrors to ensure the long-term availability and integrity of precious data.
    - Resource optimization: longer term data migration (often based on user hints or initiated by system operator commands) to balance the use of resources in different grid sites.
    - Grid wide authentication and authorization services, security infrastructure.

2. Tasks for CMS components and commercial components selected by CMS.
    - Physics analysis tools that provide user interfaces to the grid services for end-user physicists, interfaces in terms of the high-level physics application semantics.
    - Persistency layer that maintains data product values in files.
    - Optimizing the strategy for mapping data product values into files.
    - Local and remote extraction and packaging of data products to/from files.
    - Configuration management for CMS data products and metadata.
    - Generation and maintenance of configuration meta-data for each file set, creation of services which allow CMS physicists to find the data they need using application-level queries.
    - Efficient job decomposition into subjobs.
    - Mapping of the application-level job description to a grid-level description containing the names of input, work, and output file sets.
    - Configuration of resource usage and access policies.
    - Creation of error recovery rules and scripts.
    - Making the tradeoff between pre-staging and dynamic staging of files, initiation of dynamic file staging operations in CMS executables.
3. Tasks for which the level of involvement of the grid projects is yet uncertain.
    - Management and operation of a distributed CMS software repository.
    - Creating the correct runtime environment for CMS executables at each site.
4. Tasks which are not needed at all.
    - Computer support for data set selection with weak forms of equivalence.

6 Appendix: Current CMS grid needs and activities

The preceding sections talk about the CMS data model and data analysis needs in 2003 and towards 2007. This section discusses current and near-future needs and activities, and contrasts these to the future needs.

6.1 Large-scale detector simulation efforts

Currently CMS is performing large-scale simulation efforts, in which physics events are simulated as they occur inside a simulation of the CMS detector. These simulation efforts support detector design and the design of the real-time event filtering algorithms that will be used when CMS is running. The simulation efforts are currently in the order of hundreds of CPU years and terabytes of data [29]. These simulation efforts will continue, and will grow in size, throughout the lifetime of the experiment. The simulation efforts and the software R&D for CMS data management are seen as strongly intertwined and complementary activities. In addition to performing grid-related R&D in the context of several projects, CMS is also already using some grid-type software in production for its simulation efforts. Examples of this is the use of Condor-managed CPU power in some large-scale simulation efforts, and the use of some Globus components by GDMP [18] [19], which is a software system developed by CMS that is currently being used in production to replicate files with simulation results in the wide-area.

CMS simulation efforts currently still rely to a large extent on hand-coded shell and Perl scripts, and the careful manual mapping of hardware resources to tasks. As more grid technology becomes available, CMS will be actively looking to use it in its detector simulation efforts, both as a way to save manpower and as a means to allow for greater scalability. On the grid R&D side, the CMS simulation codes could also be used inside testbeds that evaluate still-experimental grid technologies. It thus makes sense to look more closely here at the exact properties of the CMS simulation efforts, and how these differ from the use of the CMS data grid around 2006.

Current CMS simulation runs have a batch nature, not an interactive nature. Each simulation run generally at least takes a few days to plan, with several people getting involved, and then at least few weeks to execute. At most some tens of runs will be in progress at the same time. So there is a huge contrast with the year 2006 situation, where CMS data processing requirements are expected to be dominated by chaotic physics analysis jobs generated by hundreds of physicists working independently. Also, in contrast to the year 2006 workloads, requests for the data in sparse subsets of (simulated) event datasets will be rare, if they occur at all. Simulated event sets can be, and are, generated in such a way that events likely to be requested together are created together in the same database file or set of database files.

6.2 CMS software packages

Two distinct software packages are currently used in the CMS detector simulation efforts described above. The division corresponds to two phases in the simulation process.

6.2.1 Physics simulation and detector simulation phase: CMSIM

The Fortran-based CMSIM software takes care of the first steps in a full simulation chain. CMSIM uses flat files in the ZEBRA format for its output. In CMSIM, each simulation run produces one output file based on a set of runtime parameters. CMSIM is very portable and can be run on almost any platform. Installing the CMSIM software is not a major job.

CMSIM will be phased out in the next few years. At the start of 2002 a transition to the C++-based OSCAR simulation package is expected, this package uses more up-to-date simulation codes from C++-based next-generation GEANT 4 physics simulation library, and will use the CMS object persistency layer (currently Objectivity) for data storage.

6.2.2 Reconstruction and analysis phase: ORCA

In the second phase of a full simulation chain, the simulated detector output is constructed, processed and analyzed as if it were the output of the real CMS detector. The major part of the processing (called reconstruction) is done using the ORCA package. Besides ORCA, ‘desktop’ analysis tools are used to analyze further some relatively small datasets created by ORCA, but these desktop activities are outside the scope of this document. ORCA

data storage is done using the CMS object persistency layer, which is currently based on the Objectivity database system. As a first step in the use of ORCA, the CMSIM `fz` format output files are read and their contents are stored as objects in the database. Compared to CMSIM, inside ORCA there is a much more complicated pattern of data handling in which intermediate products appear [24]. The ORCA software is under rapid development, with cycles of a few months or less. ORCA is only supported on Linux and Solaris, and currently takes considerable effort and expertise to install. Work on more automated installation procedures is underway.

More details on ORCA are in [24] and [9].

7 Appendix: Terminology

To make this document more self-contained, some terminology is defined here.

7.1 CMS data terms

This section defines some CMS data related terms as used in document. These are terms that often appear in other CMS and high energy physics documents as well. However, other documents may use these terms in slightly different ways.

- **Object**. The smallest unit of data in CMS is referred to as an *object*. Objects are *persistent* pieces of data on storage. They are also *atomic*, so containers with objects are never called objects themselves. In this document, the term **data product** is used in stead of object, because the word 'object' is too over-loaded.
- **Event**. In the context of the storage and analysis of CMS detector data, an event is defined as the collision phenomena that occur during a single bunch crossing. An event is not any particular piece of data in a database, rather it is a distinct *real world phenomenon* that can be measured by the CMS detector, and about which data can be kept in database. In other contexts, in particular in detector simulations, an event can also be a single individual collision during a bunch crossing.
- **Event ID**. An *event ID* is a small piece of data that uniquely identifies a particular event. CMS has not yet decided what data type will be used for event IDs. In this document, integers are used. In the production CMS system, most likely a 64-bit encoding of some numbers will be used.
- **Event data**. A piece of *event data* for event e is a chunk of data, an object or data product, that describes some aspect of event e.
- **Raw data, raw object**. *Raw data* is a type of event data. The raw data for event e consists of all measurements made of event e by the detector at the occurrence of the event. In the current CMS data model, the raw data consists of a fixed set of persistent objects called the *raw objects*. Each of these raw objects contains a set of binary readout channel values, maybe encoded with zero-suppression. The raw objects will not necessarily be stored in an object database, it is conceivable that some type of data streaming into binary files will be used.
- **Simulated data**. Event data created by a simulations of collisions inside the CMS detector. Simulated data generally consists of two components, the *Monte Carlo truth* which records the exact physics processes that were simulated, and the *simulated raw data* which records the simulated detector response, in a format that is very similar, or equal to, the real raw data format. The estimated size of a 'full simulation' for one event is 2 MB of such simulated data. Some 'fast simulation' codes may produce smaller amounts of data per event, for different types of studies.
- **Reconstructed data, reconstructed object, reconstruction algorithm**. *Reconstructed data* is another type of event data. Reconstructed data for event e consists of *reconstructed objects* that describe event e. A reconstructed object is computed by a *reconstruction algorithm* that takes as input a set of (maybe simulated) raw objects and/or other reconstructed objects belonging to the same event. The reconstruction algorithm also takes parameters, which influence its functioning.

7.2 Grid terms

For more complete explanations of the grid, see [5]. For self-containment of this document, this section briefly describes a few grid terms used in the grid projects.

- **Grid**. The following description of a grid is adapted from [5]. The term 'the Grid' was coined in the mid-1990s to denote a proposed distributed computing infrastructure for advanced science and engineering. The real and specific problem that underlies the grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organization (VO).

- **Data grid**. The original grid concept focused primarily on the sharing of CPU resources. The term ‘data grid’ was created recently to denote a grid system that has an additional strong emphasis on the sharing of large amounts of data and data storage resources.

8 Some other documents that may be of interest

The documents [13] and [14] contain important material (hardware size details and workload details) that is only covered in a sketchy way in this document. These two documents should be combined with this document to get a complete picture of the CMS data grid system.

The following documents may also be of interest. The CMS Computing Technical Proposal [8], written in 1996, is still a good source of overview material. More recent sources are [6], which has material on CMS physics and its software requirements, and [12] which has a longer version of section 2 of this document, and many details about HEP data access patterns and their optimization, though for this document wide-area issues are out of scope. Recent estimates for CMS networking are in [10]. More information about LHC computing estimates is in [7].

References

[1] http://www.griphyn.org/

[2] http://www.ppdg.net/

[3] http://www.eu-datagrid.org/

[4] http://cmsdoc.cern.ch/

[5] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. To be published in Intl. J. Supercomputer Applications, 2001. Available at http://www.globus.org/research/papers.html#anatomy

[6] CAFE Group. Requirements from Physics Analysis Process to Software Architecture. http://cafe.web.cern.ch/cafe/

[7] S. Bethke et al. Report of the steering group of the LHC computing review. CERN/LHCC/2001-004, CERN/RRB-D 2001-3, 22 February 2001. Available from http://lhc-computing-review-public.web.cern.ch/lhc-computing-review-public/

[8] CMS collaboration. CMS Computing Technical Proposal. CERN/LHCC 96-45. December 1996.

[9] ORCA Tutorial at FERMILAB. http://cmsdoc.cern.ch/orca/FermilabTutorialWeek2000/

[10] P. Capiluppi. CMS World Wide Computing. Reported at the LHCC Comprehensive Review of CMS Software and Computing. October 2000. CERN. http://cmsdoc.cern.ch/cms/software/reviews/LHCC_review_oct_00/

[11] Monarc project web page. http://monarc.web.cern.ch/MONARC/

[12] K. Holtman, Prototyping of CMS Storage Management, Ph.D. thesis (proefontwerp), Eindhoven University of Technology, May 2000. see http://home.cern.ch/kholtman/ for a link to the pdf version.

[13] Ian Willers. CMS Interim Memorandum of Understanding: The Costs and How They are Calculated. CMS Note 2001/035.

[14] Koen Holtman. HEPGRID2001: A Model of a Virtual Data Grid Application. Proc. of HPCN Europe 2001, Amsterdam, p. 711-720, Springer LNCS 2110. Also CMS Conference Report 2001/006. Web site: http://kholtman.home.cern.ch/kholtman/hepgrid2001/

[15] K. Holtman, Clustering and Reclustering HEP Data in Object Databases, Proc. of CHEP '98, Chicago, USA. http://kholtman.home.cern.ch/kholtman/chep1/art1.web.html

[16] K. Holtman, H. Stockinger. Building a Large Location Table to Find Replicas of Physics Objects. CHEP'2000, Padova. http://kholtman.home.cern.ch/kholtman/olt_long.ps , http://chep2000.pd.infn.it/abst/abs_c074.htm

[17] K. Holtman, P. van der Stok, I. Willers. Automatic Reclustering of Objects in Very Large Databases for High Energy Physics, Proc. of IDEAS '98, Cardiff, UK, p. 132-140, IEEE 1998. http://kholtman.home.cern.ch/kholtman/ideas/autorecl_ideas98.html

[18] GDMP project web page. http://cmsdoc.cern.ch/cms/grid/

[19] Asad Samar, Heinz Stockinger. Grid Data Management Pilot (GDMP): A Tool for Wide Area Replication. IASTED International Conference on Applied Informatics (AI2001), Innsbruck, Austria, February 19-22, 2001.

[20] Heinz Stockinger, Asad Samar, Bill Allcock, Ian Foster, Koen Holtman, Brian Tierney. File and Object Replication in Data Grids, to appear in 10th IEEE Symposium on High Performance and Distributed Computing (HPDC2001) , San Francisco, California, August 7-9, 2001.

[21] K. Holtman, P. van der Stok, I. Willers. Towards Mass Storage Systems with Object Granularity. Proceedings of the Eighth NASA Goddard Conference on Mass Storage Systems and Technologies, Maryland, USA, March 27-30, 2000. http://kholtman.home.cern.ch/kholtman/mss.ps

[22] Using an object database and mass storage system for physics analysis. CERN/LHCC 97-9, The RD45 collaboration, 15 April 1997.

[23] Julian J. Bunn, Harvey B. Newman, Richard P. Wilkinson. Final Report of the Globally Interconnected Object Databases (The GIOD Project). October, 1999. http://pcbunn.cithep.caltech.edu/pubs/publications.html

[24] David Stickland. The Design, Implementation and Deployment of Functional Prototype OO Reconstruction Software for CMS. The ORCA project. CHEP'2000, Padova. http://chep2000.pd.infn.it/abst/abs_a108.htm

[25] M. Schaller. Reclustering of High Energy Physics Data. Proceedings of SSDBM'99, Cleveland, Ohio, July 28-30, 1999.

[26] M. Schaller. Persistent ATLAS Data Structures and Reclustering of Event Data. Ph.D. thesis, Geneva, September 1999.

[27] A. Shoshani, L. Bernardo, H. Nordberg, D. Rotem, A. Sim. Multidimensional Indexing and Query Coordination for Tertiary Storage Management. Proceedings of SSDBM'99, Cleveland, Ohio, July 28-30, 1999.

[28] A. Hanushevsky. Developing Scalable High Performance Terabyte Distributed Databases. Proceedings of CHEP'98, Chicago, USA.

[29] http://cmsdoc.cern.ch/cms/production/www/html/general/index.html

[30] http://wwwinfo.cern.ch/asd/rd45/index.html

[31] http://www.globus.org/

[32] http://www.cs.wisc.edu/condor/

[33] http://cmsdoc.cern.ch/cgi-cmc/scrampage

[34] http://wwwinfo.cern.ch/asd/geant4/geant4.html

[35] http://anaphe.web.cern.ch/anaphe/