

The CMS PhEDEx System: a Novel Approach to Robust Grid Data Distribution

Tim Barrass, Dave Newbold
University of Bristol, UK

Lassi Tuura
Northeastern University, USA

The CMS experiment has taken a novel approach to Grid data distribution. Instead of having a central processing component making global decisions on replica allocation, CMS has a data management layer composed of a series of collaborating agents; the agents are persistent, stateless processes which manage specific parts of replication operations at each site in the distribution network. The agents communicate asynchronously through a blackboard architecture; as a result the system is robust in the face of many local failures. CMS' data distribution network is represented in a similar fashion to the internet. This means that the management of multi-hop transfers at dataset level is simple. This system allows CMS to seamlessly bridge the gap between traditional HEP "waterfall" type data distribution, and more dynamic Grid data distribution. Using this system CMS is able to reach transfer rates of 500Mbps, and sustain transfers for long periods in tape-to-tape transfers. The system has been used to manage branches of large scale Service Challenges for LCG this year.

1 Introduction

Historically High Energy Physics (HEP) experiments have been able to rely on manpower-intensive mechanisms for distributing data to geographically dispersed collaborators. The new era of CERN Large Hadron Collider (LHC) [1] computing necessitates a move to more scalable data distribution and management systems. In the near future these systems must bridge the gap between "traditional" HEP data distribution—large scale scheduled transfers between major sites—and more dynamic data distribution, which enables optimised replication in response to user demand.

In general existing HEP distribution networks are very structured, often characterised by simple tree-like topologies in which nodes are persistent, but routes are typically not in constant saturated use. This will change in the LHC era. Moreover, access to and production of data is currently undertaken at large regional centres or computing centres at Institutes or Universities. This will not change, but to access the computing power required to analyse LHC data HEP collaborations must make additional opportunistic use of resources made available for scientific analysis in general. Data distribution in this type of environment is more suited to recent Grid [2] or peer-to-peer filesharing technologies, which often develop random topologies to avoid network segmentation.

The LHC distribution environment is there-

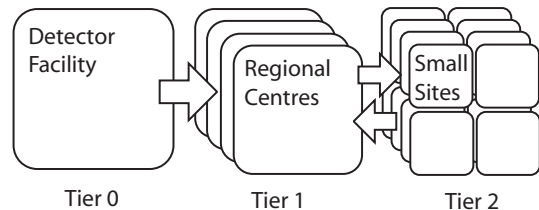


Figure 1: Schematic representation of the tiered data flow common to many HEP experiments. Raw and reconstructed detector data is produced at the detector facility and flows to regional centres, where it is safely stored, and large scale analyses are undertaken. Further processed data is transferred to smaller sites for smaller scale analyses; in addition, smaller sites produce simulated data of use to the whole collaboration which is cached at the regional centres.

fore a blend of static core infrastructure with an associated infrastructure of transiently available resources. Static infrastructure transfers are continuous, sourced from a central site through successive "tiers" of sites, hosting resources of decreasing capacity (fig. 1) [3]). LHC experiments will use this core infrastructure to store multiple remote, secure tape copies of invaluable experimental data in real time, relying on the core infrastructure's more robust and available services.

1.1 CMS and PhEDEx

CMS [4]— one of the four LHC experiments currently being built at CERN— has among the most demanding requirements of data management ???. In a given year 5 PB of raw data alone will need to be distributed over the WAN. Additionally, a further 5 PB of processed data, and a similar quantity of simulated raw and processed data must also be transferred. CMS has made important progress toward meeting these demands by developing a robust data management layer. This data management layer has a simple twofold goal: to manage the prioritised transfer of files from multiple sources to multiple sinks; and to provide answers to questions of transfer latency and speed to enable scheduling of transfer operations.

The data management layer, named PhEDEx, for Physics Experiment Data Export [5], is composed of a series of autonomous, robust, persistent, stateless processes. We term these processes agents, as their design is inspired by agent frameworks [6–8]. These agents share information about replica and transfer state through a blackboard architecture [9]. The blackboard also hosts some higher-level information about network routing, dataset subscriptions and other infrastructural information.

The PhEDEx distribution topology reflects the tiered structure common to many HEP experiments. Existing Grid middleware has tended to view such structures as completely connected, and all transfers as simply single file point-to-point [10]. More recent iterations of these applications seek to improve the robustness of these operations [11] by taking the design of PhEDEx as their starting point.

The key problem for a HEP data distribution layer is to seamlessly blend very high priority, continuous large-scale data transfers between highly available resources, with smaller scale, lower priority transfers between smaller sites for analysis, re-processing or re-analysis, and therefore make the infrastructure manageable by a small number of people. The data distribution layer needs to do this in an environment in which even the relatively static, stable core infrastructure is considered unreliable and must therefore handle failover and retry of transfers to a given destination. It is also conceivable that high-priority transfers might get dynamically reallocated in the face of failure of a regional centre, guided by collaboration policy.

Note that the distribution layer is only one

part of HEP experimnt data management system: in addition robust bookkeeping mechanisms are required that allow analysts to determine what data has been created, and trace an audit trail for all such data.

2 Overall use cases for HEP data distribution

In HEP there are two broad use cases for a data management layer: “push” and “pull”. Push indicates the transfer of invaluable raw experimental data from the detector to tape store at regional centres. “Pull” represents a request for or subscription to data. In both cases the data distribution layer must manage the transfer of either complete or incomplete datasets. In the latter case, new files must be automatically harvested and propogated to the relevant sites.

For some data— notably the invaluable raw detector data— the collaboration chooses custodial regional centres and effectively “pushes” the data onto them. Furthermore, for efficient operation it must be possible to clear newly-created files from buffer space at data sources. To do this it is vital to know whether a given file has been made “safe”. For CMS raw data “safe” is defined as “in tape storage at the detector facility and at least one regional centre”. It is this use case, and therefore the explicit necessity of managing end-to-end file state, that is the source of the differences between PhEDEx and existing file distribution systems. To manage file state the PhEDEx agents collaborate in the transfer and migration of files, and are given the responsibility for cleaning buffer space when they determine that files are safe, as governed by collaboration policy.

The pull use case can represent a scale of activities from on-going requests for data by sites as large as Universities and Institutes for group analysis to small-scale one-shot operation, in which say a single physicist is interested in some part of a dataset, and wishes to transfer it either to their University or even their laptop. These transfers are typically of lower priority than the transfers of raw detector data.

Note that the names of these use cases are not intended to describe low-level transfer activity. In all cases we implement point to point transfers as third party- although in the majority of cases they are instigated by an agent running near the destination rather than the source.

The simplest requirements placed on the system are that it should be manageable at a high

level by an individual, and that low level deployment and support take close to zero effort. To meet these requirements we devised an architecture based on quite simple, focused, robust processes that were able to handle failover and retry of transfers autonomously, and which could collaborate to solve the problem of transferring data over multiple hops to multiple destinations.

The system should also scale by the number of files in transfer. It should have a robust, internet-like design where no one part can bring entire transfer network down. The system should be able to saturate any hardware (network, storage) on which it is deployed. It should also operate completely asynchronously from other systems, for example: data bookkeeping systems; data production processes; or analysis tasks running on batch computing nodes.

Within PhEDEx we have met, or will soon meet many of these requirements.

We do not discuss the co-scheduling of replication with jobs here. In this environment the cost of replication is very much higher than the cost of running a job, and therefore it makes sense to distribute data throughout the collaboration and then send jobs to the data.

3 Current PhEDEx architecture

HEP environments are heterogeneous at all levels of operation, although some standards in transfer tools and storage management are beginning to emerge. To develop PhEDEx we rely on a core set of tools and services that are most widely available. These tools and services are broadly categorized as either storage management or transfer tools.

For storage management we rely on SRM (Storage Resource Management) [12], which provides generic access to any storage resource. Superficially SRM provides a defined two step interaction through which a file can be obtained from any storage resource by using a Storage URL (SURL), which contains the hostname of the resource and some identifier that can be interpreted by the resource. During an SRM transaction the client presents an SRM server with the SURL; the server returns a Transport URL (TURL) which indicates the current (possibly temporary) location of the file and a transport protocol that can be used to access it. This means that someone wishing to access a file at a given site does not need to know on which or what type of physical resource the file resides.

In conjunction with SRM, local file catalogues map some (collaboration level) global replica set identifier to some (local) identifier that can be used to access the file. These catalogues are not strictly seen as part of PhEDEx, as they are shared with other activities at each site—analysis tasks, for example. In this context “catalogue” is quite a generic term: it could refer simply to the internal catalogue maintained by the filesystem, or to something more complex like a MySQL database that serves file information for multiple tasks, including distribution.

A number of tools are used for point-to-point transfer of data. WAN transfers are made with tools that overlie GridFTP [13]— basically FTP with X509 certificate authentication. Such tools are globus-url-copy [14] and srmcp [12]. PhEDEx also uses a number of site-local transfer and tape management commands. This variety of tools is accommodated with a plug-in design that allows the system to provide generic components that can be easily configured to use local tools when necessary.

3.1 System design

PhEDEx design and development is guided by a number of architectural principles. The key principle is that complex functionality should be kept in discrete units, and that the handover of responsibility for sections of the transfer workflow should be handed between those units using as few messages- containing minimal information- as possible [15]. As the system comprises active components it is reasonable to model these units after agents rather than as passive stateless services.

The PhEDEx design is characterised by layered abstractions intended to improve system robustness, much as abstractions like the OSI (see e.g. [16]). Our experience with data management tools has led us to view many operations and tools as unreliable. To manage this we wrap unreliable tools and systems in more robust layers to build a reliable data transport system.

Additionally PhEDEx allows site managers to keep local information local, and allow as much local management of files as possible. This makes the system robust during local infrastructural change, but means that local and global information can become unsynchronised after deletion operations. This is not considered unscalable, as the straightforward deletion of files is rare in HEP systems: although data can be obsoleted, it is rarely deleted. More problematic

<p>Replica management Simple allocation of files to destinations based on subscriptions. Determination of closest/ best replica for transfer. More grid-like "global" replica management based on demand.</p>
<p>Dataset/ chunk level transfer Monitor transfers at chunk level, notify sites on progress. Activate and deactivate chunks. Alter routing dynamically to avoid problems. Handle automatic harvesting of files and bulk transfer requests for existing data.</p>
<p>Reliable routed, or multi-hop, transfer Efficient handover of responsibility from node to node in a transfer chain. Manage clustering of tape stages and migrations.</p>
<p>Reliable point-to-point, or single hop, transfer Failure recovery and retry of transfers.</p>
<p>Unreliable point to point transfers and technologies srmcp, globus-url-copy, lcg-rep, dccp srm, gsiftp, dCache</p>

Figure 2: PhEDEx' layered architecture. Each layer is considered more reliable than the one below.

is the loss of files due to hardware failure; in this case PhEDEx currently relies on close interaction with local admin so that such failures are quickly accommodated.

To guide development we avoid creating services that aren't necessary, generally by accepting a small increase in complexity on the "client" transfer components. To make message passing simple we utilise a blackboard architecture in which transfer components post state information. The blackboard is currently deployed as a single high availability Oracle database. We also leverage data chunking to improve performance by avoiding somewhat traditional file-to-site mappings, instead mapping files to chunks and chunks to sites; we don't, however, require a chunk to exist anywhere in entirety.

The blackboard space that agents use to exchange messages/state information is envisaged as a database schema. This database schema provides us with a structure for defining a set of message ontologies, and its deployment as a database provides us with a reliable mechanism of communication. The information stored in this schema tracks the list of files currently of interest to the data distribution system, useful metadata concerning the nodes in the system (names, routing information) and high level subscription information (which sites subscribe to which datasets, and which files are currently allocated to which specific node). It also tracks the current state of point-to-point transfers, and maintains an index structure that maps global replica set identifiers to nodes in the distribution network.

The database schema is designed with the goal of insulating information that is considered "local": all tables are logically partitioned by node where possible, and agents typically only touch data concerning their own node, or that of their neighbours.

Agents are defined at a high level only, in terms of their functional behaviour and expected interactions with other agents. They are developed using an agent toolkit, which wraps much of the low level functionality in common to all the agents: for example, handling database connections in a robust manner, or processing job queues. An explicit decision was made at the start of the project to wrap core message passing/ database access in a toolkit rather than to provide services that overlay the databases.

3.2 Overview of transfer operations

The core element of the data management system is a node- a logical entity to which persistent processes named agents are associated. Typical examples might be a regional centre's transfer node, with agents to, for example, manage transfers from the centre's neighbours to the centre's disk buffers; or a mass storage node with an agent to manage the clustering of requests from other nodes for files for transfer.

Possible routes through the network are represented by routings in the central blackboard database. The routes in these routing tables represent a high-level network that overlays the internet (such networks are named overlay networks in peer-to-peer terminology). Each node in the network runs an agent which uses an implementation of the well-known RIP algorithm [17] to maintain links with neighbours and discover or time-out routes to other nodes in the network. Using this algorithm routes can be automatically removed from the network when the node (and its routing agent) go down.

Files are discovered and routed through this network by a complementary set of agents that execute at and act on behalf of each destination. These file routers poll the blackboard to discover whether their node has any data allocated to it. On finding allocated data, the file routers search for replicas of that data; they determine which replicas are closest (fewest "hops") to their node and trigger a transfer for each. Note that the routers do not actually handle transfers- they just post a state in the blackboard describing a transfer that should happen for a particular file between two particular nodes.

Transfer workflows are divided into functionally coherent units— file transfer, for example, or migration to tape. Within those units workflow stages are defined as internal state transitions. The handover of responsibility for a replication between units are seen as messages passed between agents, and are currently represented as state transitions on the blackboard. This message passing represents the collaboration of agents to achieve their goals.

For example, a basic transfer workflow proceeds as shown in Fig. 3. 1: Allocator agent recognises that new files are available at site B for transfer. Using subscription information it allocates those files to a set of final destinations—in this case site A. 2: FileRouter agent working at A finds the new file allocated to it, finds a replica at B and posts a B–A transfer request. 3: Download agent at A selects a group of files for transfer and marks them as “wanted” 4: Export agent at B clusters the wanted files from all its neighbours and requests their stage in a manner that is efficient for local resources. When they are staged it marks the files as “available”. 5: Download agent uses some transfer tool to replicate available files. It then validates the transfer, publishes the file to a local catalogue and marks the transfer as complete.

Buffers are then cleaned by cleaner agents. These agents act as garbage collectors: they check with the replicas on a given node are required. Typically, if the file has reached all currently known destinations then the replicas created and cached at intermediate buffers are no longer required, and can be deleted.

4 Scalability

By design it is easy to deploy a self-contained PhEDEx testbed in which all operations (if necessary) can be faked: for example, fake transfers (with no file movement) can be set to succeed immediately every time, or to succeed or fail with a finite time governed by a statistical analysis of real operational data. On a test network of 5 nodes- a similar number to the number of regional centres in CMS- it is possible to show that PhEDEx can replicate around 30,000 files an hour (which translates to approximately 200,000 routing decisions an hour).

In addition, PhEDEx can handle of order 10^6 files in transfer at any given time. This performance is currently being enhanced by considering replicas of file sets rather than files, thus

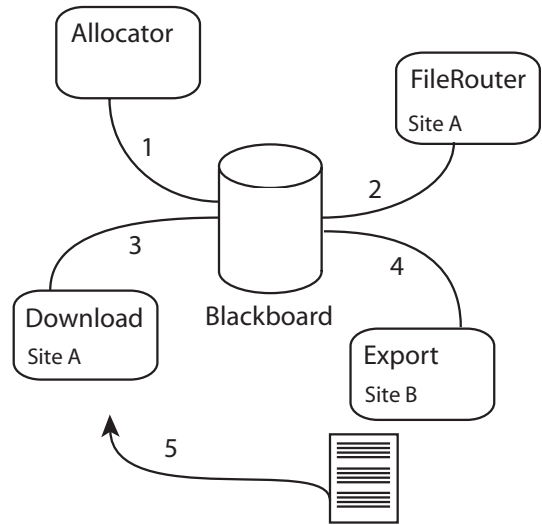


Figure 3: A sample exchange between agents to manage the replication of a file. Please refer to text for description

limiting the number of files considered at any given time.

5 Operations

PhEDEx is currently deployed at 8 large CMS regional centres, as well as a number of smaller sites. It manages transfers between a variety of storage resource types: SRM, gsftpp servers, dCache [18] disk pool managers, and a variety of underlying tape storage technologies (Castor [19], Enstore [20]). WAN transfers are managed using globus-url-copy [14] and srmcp [12].

All sites are capable of downloading data; some sites are capable of uploading data. By far the largest problems encountered have been in managing problems with the underlying storage and transfer technologies. Although SRM is in some sense a standard, there is sufficient variation between implementations (and versions of implementations) to mean that installation and upgrade is not trivial; differences in implementations have led to a number of problems resulting in failed transfers.

Bandwidth is very rarely an issue, as the experiment doesn’t yet have enough data to saturate current networks and hardware for extended periods of time. The stability of the underlying systems is frequently problematic, meaning that the system generally is not completely “up” for more than a day at a time. On average about a third of the network is down, although obviously the system is able to maintain transfers in

the face of these problems thanks to its design. Identifying problems is typically a very lengthy procedure, as we deal with complicated stacks of software deployed at widely distributed sites with varying levels of support.

The majority of the agents have been developed as perl scripts that call other transfer tools. Communication with other agents through the blackboard database is via wrapped client-side SQL.

As of June 2005 PhEDEx managed over 200TB of data. In the preceding 6 months it managed the transfer 45TB of data, with some sustained transfers of nearly 20TB a month. PhEDEx proved able to manage aggregate transfer rates in excess of 500 Mbps, including PhEDEx overhead for catalogue publishing and validation of transfers. Our current experience with the system is that efficient tape staging is the key bottleneck in the system: high rates can be achieved disk-to-disk, but stage pools are rapidly depleted of files to transfer. Generally the highest real stage rates achievable for tape stages are 300Mbps; efficient tape stages represent the most keenly-sought improvement in our performance.

6 Future developments

6.1 File Routing by Contract Tender

The current PhEDEx routing mechanism is the heaviest load on the central blackboard. An alternative mechanism is for a filerouter to send out a tender for transfer contracts: agents at each of the nodes where a replica for a given file exist would determine the cost of supplying that file, and hand their response over to the next hop in the transfer chain. A similar agent at the next hop would determine its cost for supplying that files, and hand over to the next hop, and so on, until a total cost for each chain of possible transfers from replicas to destination had been established. The destination file router could then choose the lowest-cost transfer chain for each file and trigger transfers as necessary.

6.2 Decentralisation

At present the use of a single point-of-failure database is only a theoretical problem; indeed, the central blackboard has been the most stable component of the system. Peer-to-peer filesharing applications like BitTorrent also have similar single-points-of-failure and are also demonstrably robust. There are obviously, however,

arguments for decentralizing sections of the information in the central blackboard. The most likely candidates for this decentralisation are the routing tables and the replica location information. Decentralised analogues of the former are found in everyday routing networks; analogues of the latter might be the separation of storage and lookup in peer-to-peer systems like Chord or Pastry [21,22].

6.3 Priority

One of the key issues in large-scale HEP data distribution is the management of transfer priority. Push transfers are typically higher priority than pull transfers. The balance between these two use cases might be easily met with a simple priority system. Within these use cases however there are grades of importance- some raw data might be important for calibration so that the rest might be processed, for example. Some pulled analysis data might be for a particularly interesting new analysis of interest to many people. Orthogonally to these, there must also be a resolution of global and local priority- a given site, for example, may give higher priority to data that it requests than to data that it is just caching for other sites. Within PhEDEx the discussion of this subject is currently getting underway.

7 Conclusion

The LCG HEP environment places great demands on data transfer infrastructures. The LCG CMS experiment has developed a data transfer layer named PhEDEx that demonstrably scales to 10^4 replications an hour. The early production system has been shown to be able to enable aggregate rates of order 500Mbps. Further work promises to make this system able to handle the prioritisation of transfers, and to make the system more robust in the face of possible network outages.

References

- [1] CERN, "The LHC Computing Grid," <http://lcg.cern.ch>.
- [2] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*. San Francisco, USA: Morgan Kaufmann, 1999.
- [3] CERN, "The MONARC Project," <http://monarc.web.cern.ch/MONARC/>.

- [4] CMS, “The CMS Computing Technical Design Report,” 2005.
- [5] —, “PhEDEx (Physics Experiment Data Export),” <http://cms-project-phedex.web.cern.ch/cms-project-phedex>.
- [6] FIPA, “The Foundation for Intelligent Physical Agents,” <http://www.fipa.org>.
- [7] TILAB and Motorola, “JADE (The Java Agent DEvelopment framework),” <http://jade.tilab.com>.
- [8] M. A. Shafi, M. Riaz, S. Kiani, A. Shehzad, U. Farooq, A. Ali, I. C. Legrand, and H. B. Newman, “DIAMOnDS- Distributed agents for mobile and dynamic services,” in *Computing in High Energy Physics 2003 (CHEP03)*, La Jolla, CA, USA, 2003.
- [9] D. D. Corkill, “Collaborating Software: Blackboard and Multi-Agent Systems and the Future,” in *Proceedings of the International Lisp Conference*, New York, New York, 2003.
- [10] “The European DataGrid,” <http://eu-datagrid.web.cern.ch/eu-datagrid/>.
- [11] “EGEE GLite,” <http://glite.web.cern.ch/glite/>.
- [12] “The Storage Resource Management Working Group,” <http://sdm.lbl.gov/srm-wg/>.
- [13] “The GridFTP protocol,” <http://www-fp.globus.org/datagrid/gridftp.html>.
- [14] “The Globus Project,” <http://www.globus.org>.
- [15] J. Bryson, “Where should the complexity go? Cooperation in complex agents with minimal communication,” in *Lecture Notes in Computer Science*, vol. 2564, 2003, pp. 303–319.
- [16] K. Birman, *Reliable Distributed Systems*. Springer-Verlag, 2004.
- [17] R. 2453, “The Routing Internet Protocol Version 2,” <http://www.faqs.org/rfcs/rfc2453.html>.
- [18] D. E. SYnchrotron and F. N. A. Lab, “dCache,” <http://www.desy.org>.
- [19] CERN, “Cern advanced storage (castor),” <http://castor.web.cern.ch/castor/>.
- [20] FNAL, “Enstore,” <http://hppc.fnal.gov/enstore/>.
- [21] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica, “Wide Area Cooperative Storage with CFS,” in *ACM SOSR*, Banff, 2001.
- [22] A. Rowstron and P. Druschel, “Pastry: Scalable Distributed Object Location and Routing for Large-scale Peer-to-peer Systems,” in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, 2001.