Scaling CMS data transfer system for LHC start-up

L Tuura¹, B Bockelman², D Bonacorsi³, R Egeland⁴, D Feichtinger⁵, S Metson⁶ and J Rehn⁷

¹ Northeastern University, Boston, MA, USA
² University of Nebraska–Lincoln, Lincoln, NE, USA
³ INFN–CNAF, Bologna, Italy
⁴ University of Minnesota – Twin Cities, Minneapolis, MN, USA
⁵ PSI, Villigen, Switzerland
⁶ University of Bristol, Bristol, UK
⁷ CERN, Geneva, Switzerland

Abstract. The CMS experiment will need to sustain uninterrupted high reliability, high throughput and very diverse data transfer activities as the LHC operations start. PhEDEx, the CMS data transfer system, will be responsible for the full range of the transfer needs of the experiment. Covering the entire spectrum is a demanding task: from the critical high-throughput transfers between CERN and the Tier-1 centres, to high-scale production transfers among the Tier-1 and Tier-2 centres, to managing the 24/7 transfers among all the 170 institutions in CMS and to providing straightforward access to handful of files to individual physicists.

In order to produce the system with confirmed capability to meet the objectives, the PhEDEx data transfer system has undergone rigorous development and numerous demanding scale tests. We have sustained production transfers exceeding 1 PB/month for several months and have demonstrated core system capacity several orders of magnitude above expected LHC levels.

We describe the level of scalability reached, and how we got there, with focus on the main insights into developing a robust, lock-free and scalable distributed database application, the validation stress test methods we have used, and the development and testing tools we found practically useful.

1. Introduction
The Compact Muon Solenoid (CMS) experiment on the Large Hadron Collider at CERN [1, 2, 3] is expected to start running within a year. The global aggregate data transfer rate among the host laboratory and the main 50 or so Tier-1 and Tier-2 centres is expected to reach 200 TB or 100’000 files per day in the first year’s operation. In addition to these continuous transfers there will be some transfers to the 170 institutes involved in CMS and to the 3000 or so personal computers of the physicists.

PhEDEx [9, 10, 11] was developed to manage the CMS data transfers. It provides an interface for CMS and site data managers to manage and monitor data placement decisions, schedules transfer requests for execution, and dispatches file transfers to underlying grid file and storage management services [17]. PhEDEx includes monitoring, visualisation, reporting and consistency checking facilities for data managers, site operators and experiment data operations planning. It has been exercised in progressively increasing complexity and scale during several
Figure 1. CMS global aggregate data transfer volume since 2004, and detailed transfer rate and quality in 2007.

years of use in daily production and computing challenges [7, 8]. In the last 18 months PhEDEx transferred 20 PB data. As can be seen in Figures 1(a) and 1(b), in July 2007 the average global daily transfer rate was 100 TB/day or 1.2 GB/s. In validation tests using a realistic rather large setup but bypassing the actual file transfers, PhEDEx has demonstrated capacity to sustain well over half a million transfers an hour for extended periods of time (Figure 2).

The CMS data management system is made of loosely coupled components [4, 5, 6]. PhEDEx interacts mainly with the local storage, grid file transfer services, the CMS dataset bookkeeping system, the dataset location system and a site-local file catalogue. PhEDEx cross-checks with the dataset bookkeeping system the file-level information for datasets mentioned in transfer requests, and updates the data location system as data transfers complete. The data location system tracks completely transferred data and is used by physicists to locate data available for analysis, whereas PhEDEx only tracks data actually in transfer and is usually not accessed by the physicists.

Technically PhEDEx is an agent based system [16]. The agents store their state and communicate with the other agents via a central “black board” database [14] hosted at CERN. Each agent performs a single well-defined specific task, such as making file routing decisions or managing the file transfers to a specific destination. A number of service agents are operated
centrally at CERN. Each site in general operates only the agents that interact with the storage at the site: an export agent for outbound transfers and a download agent for the inbound ones. At present 150–200 agents run at any one time per PhEDEx instance. A web site offers various management tools and a live view to current and historical transfer conditions.

We examine in this paper four areas with notable contributions to PhEDEx performance and scalability: the data management system design, engineering choices, testing and validation process, and distributed database design. Each has helped CMS build a data transfer system whose performance is limited by the storage systems, networks and the file-level transfer tools, not by the transfer system itself. Obviously these other factors profoundly influence the performance of CMS data transfers; it is not merely a function of the performance of our own systems. While this paper focuses on just PhEDEx, the reader should not ignore the importance of the extensive commissioning and debugging work CMS has carried out to ensure the entire stack performs well.

This paper expands on and extends the techniques we have described earlier [12, 13].

2. System design choices

In this section we explore the system design choices with most direct impact on scalability and performance. While we feel these decisions are not particularly sophisticated, it is worth noting they nevertheless pursue a direction significantly different from that explored and proposed by the early grid projects.

2.1. Restricted problem scale

CMS datasets are divided into file blocks of some terabytes, or from hundreds to some thousands of files each. Data placement decisions are made only in terms of these blocks. The file block size is determined by whatever is a practical unit for storage management (5–10 TB), and greatly reduces the cost of data placement decisions. By design the data transfer system only needs to consider file blocks actually in transfer, simplifying the handling of “infinite” data sets: open-ended collections growing over years to millions of files.

2.2. Optimisation for success

CMS data transfers must succeed almost always in order for us to succeed as an experiment; we simply cannot afford a very high transfer failure rate. Therefore we accept heavy penalties for failure handling if that earns us a significantly optimised fast path for successful operations. We utilise fairly deep queue pipelines on the assumption that disruptive behaviour requiring a queue reshuffle is infrequent and preceded by much greater damage done elsewhere.

2.3. Optimisation for stable operation

Stable state performance is more important to CMS than edge behaviour. We optimise to fill the available capacity of an actively transferring site at the expense of response time to sites transferring files only from time to time. CMS data transfer requests indicate a priority which the system uses to prioritise transfers; this relieves humans of mindless juggling tasks. We prefer to schedule transfers at a guaranteed fair pace rather than consider the entire work queue for an optimum order. Sudden request surges are handled piece-meal and sites with unreasonable backlogs can only inconvenience themselves. The self-pacing is implemented by automatically limiting the amount of “active” pending work, and keeping only approximate track of the complete pending work queue. Sites with backlogs are regularly probed to give them a fair

1 For example, we have estimated that at start-up 1% permanent transfer failure rate would translate to two hours of work per day, per site.
chance to resume operations, but otherwise backlogs are hibernated to prevent general system slowdown.

2.4. Local information remains local
CMS does not employ the database-based global or site-local file catalogues proposed by the early grid projects. In CMS logical file names are determined before the jobs are submitted, and at each site a trivial file catalogue, a text file with a handful of mapping rules maintained by the site operators, is used to determine actual file locations. This eliminates a slow and fragile component from the interactions of the transfer system. Furthermore, CMS data transfers are managed by agent programs running near to the destination storage. The site operators are responsible for the transfers to the site and have much freedom to configure the agents for best performance, and can diagnose and monitor errors in the local agent logs. On the other hand, the PhEDEx database records considerable agent health and transfer error detail, allowing remote problem analysis and the highlighting and summarising of issues on the central web pages. These are two examples of curbing the information a site has to maintain up-to-date outside its perimeter, but which still encourages effective co-operation.

2.5. Ease of operation
One of the primary goals of the project has always been to ease the life of the site and data managers. Operating data transfers at the required scale requires robust, resilient tools which automate the mundane, run autonomously for weeks with little or no attention, and yet provide a clear overview of the situation in a glance. In this day and age one should be able to monitor and administer transfers via a web site to further reduce the burden on the operators.

2.6. Extensive monitoring
Effective data transfer operation requires access to extensive monitoring and reporting facilities on the relevant performance data, both current and historical conditions, for a variety of user groups. A sizeable portion of PhEDEx is dedicated to this task. The monitoring and reporting contribute significantly to the effectiveness of the people as well as the automated decision making in the data transfer system itself.

2.7. Working systems isolated from problems
Some but not all data transfers are critical to the success of CMS. It is highly desirable the transfers not to be affected by instabilities in unrelated parts of the transfer network. Of course in a system this large some part will always be in trouble at any one time. Combined with our experience we were lead to design a system hardened to operate robustly and near-autonomously even in the most debilitating conditions.

We have concluded the best strategy is to defensively expect failure everywhere and to contain the problems without attempt to understand what is wrong. We simply capture as much context and detail as possible for later manual analysis and diagnosis offline. We maintain our own independent data transfer performance and quality records, and regard irrefutable forward progress—independsently verified arrival of files into destination storage—as the only meaningful measure of the systems’ capability to perform. We prevent the transfer system from being clogged by errors by gracefully backing off on several levels, by pacing backlog handling, and by probing sites with questionable recent record before attempting large transfer volumes.

A rather grim experience from years of remarkable manpower investment to compensate for major deficiencies in the technologies we use, not least of which are unreliability, almost complete lack of useful error detail and failure to report successes accurately.
We have chosen these strategies to protect our interests. Obviously they will not magically fix broken underlying systems. We emphasise CMS can only execute its computing model successfully when the vast majority of transfers succeed on the first try.

3. Technical implementation choices
In this section we explore technical choices and preferences which we feel have affected the performance and scalability of the CMS data transfer system in an important manner. It is entirely reasonable for others to build a sound design from different choices. None of this is particularly novel.

3.1. High availability high performance central database
We chose to build a distributed database application with a central well-tuned highly available database at the experiment host laboratory CERN, with clients connecting to it from everywhere in the world. CMS has been well pleased with this arrangement. We do not think it would be wise at this time to deploy PhEDEx-like databases at dozens of locations.

3.2. Level-triggered asynchronous state manipulation
We considered it too costly to implement, at a level of quality we would expect, a distributed edge-triggered system reacting to an event such as “transfer completed” with a synchronous response. Instead, PhEDEx operations are asynchronous for error-resilience: work is queued in the database and updated or removed once successfully completed; there is no direct communication between the requestor and the agent performing work. Furthermore, work to be done is defined implicitly as the difference of the current and the desired state, not as explicit messages.

This avoids losing track of edge transitions and lends itself to state manipulations which correct and repair the state over time, giving the system a self-healing character resistant to minor bugs. In general this implies work to be done is unordered, or rather ordered by the processes scheduling new desired state; where further ordering is required, we use “rank” hints which the agents may disrespect when efficiency so requires. From the level-triggered state design it follows operations do not produce responses: the new, desired state is the response.

It must be noted this design is not without challenges. The transfer system requires a high turn-around rate, and too heavy a state or an operation would chew the database servers to death. Later sections explore some of the lessons we have learnt in this area.

3.3. Defensive error handling
PhEDEx provides a reliable service on an unreliable foundation. Most grid services appear to assume most of the time everything works correctly and errors are just a rare inconvenience, and throw in some haphazard error handling accordingly. Our experience has been almost entirely opposite, we find all kinds of errors and unexpected behaviour happen all the time, and have worked hard to make the system resilient in these circumstances. Programming defensively and coding patterns for error recovery have been essential to building a fault handling strategy able to contribute towards overall system performance. The techniques we use include squelching redundant noise, swallow-and-retry-later on uninteresting transient errors, backing off and letting error-prone systems “cool off”, and generating operator alarms in distress.

3.4. Hundreds of concurrently operating stateless agents
The PhEDEx database is small, and only grows slowly with the volume of data to transfer. It is however highly transactional and concurrent, the shared “level” state is constantly being queried and modified by hundreds of active database processes around the world as described above. All
this requires careful attention to locking and cache coherence issues in the clustered database environment we use. While certainly not trivial to implement efficiently, this is the bread-and-butter business for database servers and much optimisation expertise is readily available.

3.5. Hierarchical monitoring

While the monitoring offered by PhEDEx must represent events accurately and in a timely manner, there is no need for truly real time database state capture. Similarly, it is desirable to have very quick access to common monitoring data, but it is acceptable for user to have to traverse to the gory details in a number of steps, each of which exposes additional detail but is increasingly costly (and usually narrows down on the target) [15].

We have designed a hierarchical monitoring system within PhEDEx exploiting these properties. Our scheme guarantees minor predictable lag normally unnoticeable to the users, and yields feedback quicker than queries to the true database state could return. We do this by dividing PhEDEx data into monitoring zones, and statistics collected from each zone are propagated up the monitoring hierarchy to the consumers. The statistics are also recorded into variable-resolution time series summaries optimised for frequent and speedy overall statistics access, even for extended historical queries. Each monitoring zone operates autonomously: a designated “well-informed” agent takes monitoring state snapshots when it knows it to be economical. This allows us to “leak” the state data from the hot database zones at suitable frequency but with little additional load.  

3.6. Straightforward database access

We maintain our database schema in simple SQL text files, one per each major component in our system. Our agent code embeds verbatim SQL statements for database access; we use no abstraction packages such as object-relational mapping toolkits. We believe these practises gained us database technology familiarity and design flexibility that were pivotal to reaching the levels of scalability we will describe below.

4. Validation and test processes

Outside the uses in the daily production, various computing challenges and the CMS computing integration activities, the developers carry out an extensive validation of PhEDEx on average twice a year. The validation is performed on a separate database cluster dedicated to the tests so that findings are unambiguous. The validation is done always before releasing major schema changes and from time to time to verify that cumulative patches have not introduced undesirable side effects. The CERN database administrators are involved and decide whether the version may be operated on production database servers. The findings are written up as technical reports. Figure 2 shows some of the metrics we measure and report.

PhEDEx allows nearly any part to be short-circuited for various kinds of tests, and to simulate realistic behaviour using statistical models from profiles of real transfers [13]. These capabilities are used during the validation to study performance in conditions reflecting real life, and to create the stress tests. The stress tests use problem sizes that exceed the expected peak load by a factor of about 100. We require the system to handle the “extra” load gracefully in order to consider the test successfully passed.  

In the most costly and heavily hit configuration we have been able to create for our tests, the overhead of the monitoring system was measured to account for less than 1% of the database server load and approximately 10% of the database I/O.

This is not quite as excessive as it may sound. The expected peak loads are expressed as daily averages. If a day’s worth of transfer work is injected into the database in 15 minutes, the factor 100 safety margin has been used up. It is highly desirable the system has capacity to handle such cases gracefully, even if the file transfers themselves are more limited.
Figure 2. Examples of metrics reported from the PhEDEx 2.5 validation.

the system is actually ready for release.

Creating sufficiently large and realistic test cases of course requires an investment from the developers. We have developed a suite of scripts for controlled initialisation of different test conditions. We also use snapshots of the production instances and portions of the CMS Tier-0 infrastructure to generate additional starting points. In order to execute tests involving large numbers of concurrent agents, we launch groups of agents as jobs to the CERN batch system and then control them remotely via the database and the web site.

In order to monitor, measure and understand the system under test, we have made extensive use of Oracle’s performance analysis features [18]. We use the real-time and historical performance analysis features of the Enterprise Manager, and the Automatic Workload Repository reports for detailed summary data. We recommend both highly. The Enterprise Manager in particular helped us understand the application performance at depth such that we could identify the real causes and make corrections very quickly.

5. Scalable distributed database design

We now discuss some database technology factors we have learnt to be important for the performance and scalability of this particular type of application when using Oracle version 10g cluster servers (RAC or Reliable Application Cluster). The reader is advised the comments
may translate poorly to other database products and types of applications.

5.1. Basics
We always use bind variables. We use row arrays wherever possible, both for fetches and for uploads. Our default array sizes are thousands of rows. We are very selective about when and where we commit in order not to ruin performance.

At the validation stage we remove or disable all indices except those required for constraints, and in case of Oracle, indices covering the foreign keys. We then re-enable and keep only those indices the performance analysis indicates are truly beneficial.

In general we find it is better to send queries for execution at the database server rather than keep the logic in the client and ship data to the client, or worse yet, back and forth between the client and the server. Long network latencies can make this evident in a spectacular manner. There are a few notable exceptions discussed further below.

We find most of our tables need storage options such as partitioning and data organisation directives. Our schema benefits tremendously from cost-based query optimisation. The cost data needs to be well maintained; ours is refreshed daily.

5.2. Merging large state efficiently
In one part of our application we recompute a new state and merge it with the old one. The old state is large, hundreds of thousands of rows, the delta new state usually some thousand rows at most. We initially uploaded the new state to a scratch pad table and asked the database server to merge with the old one. This turned out to be much less efficient than doing the merge on the client side. We were slightly surprised with the discovery that it is efficient to read hundreds of thousands of rows each minute or so.

5.3. Increasing parallelism
We found a number of useful methods to increase scalability when numerous clients need to simultaneously modify the database contents, and specifically when they need to communicate with each other through the database state changes. Parallelism was in our case mostly constrained by row lock contention and excess cache coherence traffic. Hence each of these is effectively focused at reducing either or both.

Firstly, most of our tables and indices are partitioned such that agents will in general access only one partition, thus effectively giving each site their own private copy of the table with just their data. This concentrates the data of a particular site to the database server serving the agents for that site, and thus eliminates a great deal of cache coherence traffic.

Secondly, we defined a data ownership model with a clear, single owner for every row in every table at any one moment in time. Only the owner of the row is allowed to change the data, and ownership must pass clearly through the system. In our case this usually corresponds very nicely with the application state machines. This approach eliminated practically all known lock contention from PhEDEx. It completely eliminated the “system log jams” we used to have every once in a while when some agent failed to commit for a long time.6

Thirdly, we observed that Oracle is efficient with inserts and tolerable on deletes, but row updates are undesirable.6 In certain places we augmented the above data ownership policy with

5 It is not particularly unusual for the agents to experience an unstable network connection to the database from time to time. We usually recover from the condition automatically after a little while, e.g. half an hour. The problem was that the row locks prevented other agents from making progress until the agent recovered or Oracle rolled back the transaction and released the locks.

6 Note for example that rows inserted by one transaction will not be seen by another transaction until a commit. With suitably constructed queries and transactions, it is possible to completely avoid phantom reads and any particular need for locking.
Passing the baton through hot path

Figure 3. Splitting a hot state table into a main table and auxiliary index-organised tables. The main table is read-only after insert up to the last agent in the chain. The auxiliary tables are read-only after insert. Inserting a row and committing is a sign for the next agent to start processing; the previous agent may no longer even access that row. The last agent in the chain is out of the hot path and deletes the rows at an opportune time.

an immutability rule illustrated in Figure 3: the rows become immutable on creation and remain so until the ownership is passed to the agent performing final clean-up. The model asserts that once the row ownership is passed to the next agent in the chain, the previous agent may no longer even access the row. This means the new owner, and in particular the last agent in the chain, is free to do whatever it pleases with the rows, whenever it thinks is the optimal time to mop up. As illustrated in the figure, each owner adds its data to a separate auxiliary index-organised table; these are very cheap to use. The scheme is free of all lock contention and guarantees perfect read consistency.

5.4. Scaling the number of database connections

Let us consider again the number of clients connecting to the central servers, at present 150–200 per PhEDEx instance but as much as twice that in the past. Each database server can efficiently serve a certain number of client connections (500 per our present server), and on the other hand the number of PhEDEx clients connecting can only be expected to grow as more institutes become involved. For quite a while PhEDEx was served by a single database server, using another as a fallback, but it became evident we needed to address the matter. We considered three alternatives: abandoning the direct client connection model and introducing application gateway servers; reducing the number of connections by sharing database connections between agents; and pruning unnecessary agents at the sites and optimising the application for a clustered database, i.e. making sure cache coherence traffic remains limited. We implemented the last option using methods mentioned above. The resulting design appears sufficient for the foreseeable future.

6. Conclusions

CMS has designed and implemented a data placement and reliable transfer system PhEDEx. We have pushed our own part as hard as we could to ensure our service scales well beyond the capacities needed at LHC start-up. We have gained extensive experience with grid transfer technologies by using them quickly after each release, and in large scale validation tests, including all the LCG service challenges. We are just now reaching data transfer volumes, but not yet
the full complexity, expected in a year’s time, and have shown we can operate the system uninterruptedly for extended periods of time.

The LHC experiments arrived together at the conclusion that currently there is no demand for a grid middleware product for data placement: this function is tightly coupled with the experiment computing models, policies and dataset bookkeeping systems. On the other hand, portions of EGEEs FTS were modelled after PhEDEx, a significant departure from the designs in previous grid projects. It was expected that FTS would become the implementation of the lowest data transfer layers of PhEDEx. After a strained initial relationship, that is now largely the case at the EGEE sites. In short, the grid data transfers are now more aligned with the experiment plans than ever before.

ATLAS and CMS, apparently independently, have arrived at similar conclusions on high-level data management concepts. Both place data at sites as a deliberate policy action, not reactively in response to jobs. The data “subscription” processes are similar. Both have data units larger than files but smaller than datasets and an agent based transfer management design. We conclude this validates many of our choices.

Acknowledgments
The authors wish to thank especially the CERN database support group and the CMS data and site administrators. Without these people the results described here would not have been possible, nor meaningful.

References
[1] CMS Collaboration, “Technical proposal”, CERN/LHCC 94-38, 1994.
[2] CMS Collaboration, “The Compact Muon Solenoid Computing Technical Proposal”, CERN/LHCC 1996-045, 1996
[3] CMS Collaboration, “The CMS Computing Project Technical Design Report”, CERN/LHCC 2005-023, 2005
[4] CMS Collaboration, “The CMS Computing Model”, CERN/LHCC 2004-035, 2004
[5] Hernandez, “The CMS Computing Model”, Proc. CHEP06, Computing in High Energy Physics, Mumbai, India, 2006
[6] Fanfani, “Distributed Data Management in CMS”, Proc. CHEP06, Computing in High Energy Physics, Mumbai, India, 2006
[7] D. Stickland, “Computing Models and Data Challenges of the LHC experiments”, Proc. CHEP04, Computing in High Energy Physics, Interlaken, 2004
[8] Bonacorsi, “CMS experience in LCG SC3”, Proc. CHEP06, Computing in High Energy Physics, Mumbai, India, 2006
[9] Barrass et al, “Software agents in data and workflow management”, Proc. CHEP04, Computing in High Energy Physics, Interlaken, 2004
[10] Barrass, Newbold and Tuura, “The CMS PhEDEx System: a Novel Approach to Robust Grid Data Distribution”, UK e-science Programme All Hands Meeting, Nottingham, UK, 2005
[11] Rehn, Barrass, Bonacorsi, Hernandez, Semeniouk, Tuura and Wu, “PhEDEx high-throughput data transfer management system”, Proc. CHEP06, Computing in High Energy Physics, Mumbai, India, 2006
[12] Barrass, Bonacorsi, Hernandez, Rehn, Tuura and Wu, “Techniques in high-throughput, reliable transfer systems: break-down of PhEDEx design”, Proc. CHEP06, Computing in High Energy Physics, Mumbai, India, 2006
[13] Barrass, Innocente and Tuura, “Replaying a data challenge to optimise resource usage”, Proc. CHEP04, Computing in High Energy Physics, Interlaken, 2004
[14] Corkill, “Collaborating Software: Blackboard and Multi-Agent Systems and the Future”, Proc. International Lisp Conference, New York, 2003
[15] Navas and Wynblatt, “The network is the database: data management for highly distributed systems”, Proc. ACM SIGMOD, 2001
[16] The Foundation for Intelligent Physical Agents, http://www.fipa.org
[17] Foster and Kesselman, “The Grid”, Morgan Kaufmann, 1999
[18] Kyte, “Effective Oracle by Design”, McGraw-Hill Osborne Media, 2003