LHC data files meet mass storage and networks: going after the lost performance

T. Barrass, University of Bristol, UK (tim.barrass@physics.org)
V. Innocente, CERN, Geneva, Switzerland (vincenzo.innocente@cern.ch)
L. A. Tuura, Northeastern University, Boston, MA, USA (lassi.tuura@cern.ch)

Abstract

Experiments frequently produce many small data files for reasons beyond their control, such as output splitting into physics data streams, parallel processing on large farms, database technology incapable of concurrent writes into a single file, and constraints from running farms reliably. Resulting data file size is often far from ideal for network transfer and mass storage performance. Provided that time to analysis does not significantly deteriorate, files arriving from a farm could easily be merged into larger logical chunks, for example by physics stream and file type within a configurable time and size window.

Uncompressed zip archives seem an attractive candidate for such file merging and are currently tested by the CMS experiment. We describe the main components now in use: the merging tools, tools to read and write zip files directly from C++, plug-ins to the database system, mass-storage access optimisation, consistent handling of application and replica metadata, and integration with catalogues and other grid tools. We report on the file size ratio obtained in the CMS 2004 data challenge and observations and analysis on changes to data access as well as estimated impact on network usage.

OVERVIEW

Motivation

Typically high-energy physics applications produce files whose sizes are far from ideal for network transfers, mass storage systems and even read throughput for analysis programs. The median output file size of 570 kB seen in the CMS DC04 data challenge in spring 2004 [1, 2, 3] is, while something that can be improved on, unlikely to grow to the desired 1.5-2 GB.

The reasons for the small file size include splitting the processing of large input samples into many jobs for efficient use of computing farms, limited amount of output produced by a single job, and the fact that each job produces new output files. For reasons beyond direct experiment control this is unlikely to change soon. However the output from different jobs could naturally be merged back to a larger logical "stream" provided the time to analysis does not significantly increase.

A number of merging options are available. In practise merging is cheaper if the file contents or identity does not change. Not touching the contents also avoids violating the file immutability assumptions made by grid tools and allows the merging to be done in any stage of the data management system. A flexible and optional knob for tuning network and mass storage performance without interference to the core frameworks is desirable.

Large files may also cause inconvenience so the benefits should be carefully weighted. Some studies seem to indicate that certain file systems may behave poorly if several large files are simultaneously written to on the same file system. Merging may also complicate use and tracking of meta data.

Candidate Solution

CMS [4] is experimenting with uncompressed zip archives [11] for file merging. We store the original files unmodified *and uncompressed* in a zip archive. Zip archives include an easily accessible table of contents at the end and store the members in a single contiguous byte range. This forms the basis of serving the original files transparently as files within files.

Our software accesses data files through a library that serves storage objects with POSIX-like file semantics [7]. A number of plug-ins extend the base abstract interface: normal files, direct access to a mass storage (Castor, dCache), and the standard network protocols (http, ftp, gsiftp, sfn), and zip members. Any method can recursively be used with zip member access, including zips within zips.

Merged files are registered into catalogues using standard-conforming new URL protocol, co-operating with existing conventions and tools. As far as the rest of the system is concerned, the original files remain distinct and retain their identity and meta data attributes.

The file merging tools are used to either process files directly from a data processing farm before passing them on to the data transfer system, or in a separate step to recluster the data. The merging allows for different queuing policies and strategies for different types of data.

FILE MERGING

PhEDEx is a distributed agent-based data movement system [5]. Each agent performs a single well-specified task. Some of the agents are site-local, preprocessing data or meta-data before it is injected into the transfer system. We call such programs "drop box" agents because they receive task drops from upstream through an incoming "mailbox" directory. For CMS each drop carries information about the output of a single job; the drops can be received in real-time from a farm, or they can be generated for instance from the CMS RefDB production tracking database [6].

The file merging is done by a drop box agent after the job output is cleaned up. It runs a master and number of parallel workers. The master agent examines the files produced by a job and places them into appropriate queues based on the file meta data attributes. When a queue expires, the master collects the files assigned to the queue and passes the information to the least-loaded worker agent for merging. The master can also prefetch files from the mass storage while idle to increase the overall system throughput.

Queues are defined dynamically by a tuple of file attributes; we used *(dataset, owner, stream, file category, file type)*. Each queue has a configurable age and size limit, allowing merging policy and transfer expediency to be defined. A queue is flushed when either limit expires; we used 1.5 GB and 30 minute limits for all queues and forced related queues to flush simultaneously to ensure that zip archives for related file types always included the output from the same set of jobs.

Each worker agent takes the queue specification sent by the master, downloads from the mass storage all the files not yet prefetched by the master, creates the zip archive and feeds it to next downstream agent. A configurable number of workers run in parallel, and each of them copies files from the mass storage with a configurable number of parallel processes. We used five parallel workers each making five parallel mass storage file copies.

To preserve the identity of the files the worker agents generate an XML catalogue template with entries for the archive and each member. The file transfer agents add a replica for both the zip archive and all the members when detecting a transfer of a zip archive for merged files. The catalogue template is stored in the zip archive along with a simple file list template to simplify the registration: the transfer agents substitute the local path for the archive in one of the templates and register the replicas to the destination catalogue.

FILE ACCESS

Accessing files is illustrated by Figure 1. The file-within-file approach allows us to fool our database libraries (POOL [9], ROOT [10]) into believing that they are reading normal local files when they are in fact accessing members of zip archives directly from a mass storage system. There is no need to make local copies of the archives or to extract the archive members.

We implemented an extensible storage factory using the LCG SEAL project [8] C++ classes to read and write zip archives and the SEAL plug-in manager. The factory provides a simple interface to open and check the existence of URLs, using the URL protocol to select the plug-in that can handle the operations. Opening a file returns a pointer to a SEAL `Storage` object, an abstraction of seekable file-like objects with POSIX-like interface.

Each storage factory plug-in provides protocol-specific means to open and check the existence of files. Opening translates to returning an instance of a suitable `Storage` class, such as `File` or `RFIOFile`. When a remote file cannot be operated on directly, it is downloaded to a local a temporary directory and a reference to the temporary local file is returned. This is the case for FTP for instance.

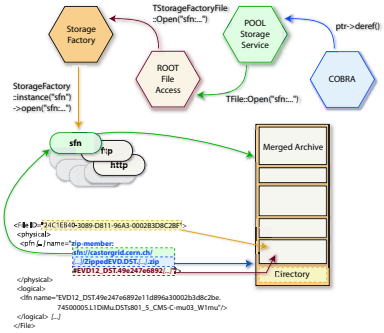


Figure 1: File access.

Some but not all protocols support creation and writing of files. It is possible to write directly to both dCache and Castor, providing a means to write log files and other large output files directly into the mass storage without having to store a copy on the worker node.

The zip-member plug-in splits the URL into archive and member parts, opens the archive using the storage factory, locates the requested member using the archive table of contents, and returns a storage object for the member's byte range. Only uncompressed members are supported to avoid seeking in compressed data—the actual data is normally compressed anyway. Since only reading is possible through the interface, archives are opened once per thread. This reduces the number of file descriptors used and connections made to the mass storage servers.

A ROOT plug-in redirects ROOT's `TFile` operations to the storage factory, rendering changes to POOL or ROOT unnecessary. We optionally hijack ROOT's URL resolution mechanism to disable its internal access methods for non-local files.

The following URL protocols are handled:

- `file`: local files
- `ftp`, `http` and `web`: download files with with `curl`
- `gsiftp` and `sfn`: download files with `globus-url-copy`
- `rfio`: direct RFIO access, no download
- `dcap`: direct dCache access, no download
- `zip-member:archive#member`: direct access to a zip member; the archive part is resolved through the storage factory, allowing for instance `zip-member:rfio:/path/to/a.zip#member`

WORKFLOW

In CMS DC04 the CERN Tier-0 farm processed events at up to 25 Hz, peaking at approximately 250 MB in 19 files every 40 seconds. After copying the output files to Castor, the job created a drop at the transfer chain entry, from

Table 1: DC04 file size statistics

	# of files	Total	Minimum	Maximum	Average	Median
Before merging	560589	6380 GB	21 kB	412 MB	12 MB	570 kB
Merged zips	3683	866 GB	43 kB	1608 MB	241 MB	32 MB
Combined	564272	7246 GB	21 kB	1608 MB	13 MB	582 kB

Table 2: DC04 replay file size statistics: maximum possible merging

	# of files	Total	Minimum	Maximum	Average	Median
Replay input	278483	3411 GB	21 kB	268 MB	13 MB	866 kB
Merged zips	4486	3411 GB	41 kB	1744 MB	779 MB	146 MB
Change	/ 62	–	x 2.0	x 6.5	x 60	x 174

where the files were published for transfer and distributed to the regional centres. For the file merging test the drops were channeled to the merging system on a separate system. The merged archives were sent back to the transfer chain with updated catalogue information. The archives were collected and held until the end of the data challenge, and then released as a transfer performance test.

RESULTS

We have not been able to measure any overhead for the layer accessing files through the storage factory plug-ins. Opening each archive only once in each thread however is an advantage if small amounts of data are read from many archive members, especially with dCache.

File merging is resource intensive. Only moderate amounts of CPU capacity are required to create zips. The main requirement is correct scheduling of simultaneous large streams of data to and from network and disks. We rejected three Linux configurations before settling on a Sun Solaris system capable enough for the job. The first Linux system lacked sufficient disk capacity and CPU and I/O capacity to keep up with the incoming data averaging at most 5-6 MB/s. The second system was identical except it was dedicated to the task and had more disk capacity. While the hardware (2 x 1 GHz Pentium III, 512 MB memory, 20 GB/7200 rpm EIDE drive, ext3 file system) seemed adequate, the operating system (Linux 2.4.x) appeared unable to properly schedule concurrent reads and writes: the merging was read starved and the disks trashed. We tested another Linux configuration with a RAID drive to check this hypothesis, and the I/O performance improved, but not sufficiently. We finally settled on a 2-CPU Sun with SAN (RAID) disks which proved to handle both the CPU, network and I/O load well enough to keep up with the incoming data rate and to even recover from processing backlogs. It is evident that sufficiently capable hardware and operating system must be used for file merging.

Merging efficiency depends on the data mixture received by the agent. When a farm runs jobs for a mixture of datasets and the queue age limits are low compared to the output rates, files for different datasets are likely to get mingled, leading to less effective merging. If data is injected offline for entire datasets at once, thousands of files may get merged together.

We compared the rates achieved at DC04 to an ideal replay, and the results appear encouraging: DC04 sample on a subset of the total data achieved significant merging rate, approximately *factor sixty increase* in median file size and *factor twenty increase* in average file size. Table 1 and Figure 2 present the results for DC04. For comparison Table 2 and Figure 3 present similar results for a simulated partial replay of the data challenge, with infinite queue time limits to discover maximum possible merging rates.[1]

The transfer rate also increased as expected. The test at the end of DC04 sustained approximately 70 MB/s from CERN to two Tier-1s, a significant increase over typical rates seen: as shown in Figure 4 (courtesy of José Hernández, CIEMAT), during the transfer test the transfer rate from CERN to PIC increased by *factor of four to five* to a sustained rate of about 30 MB/s.

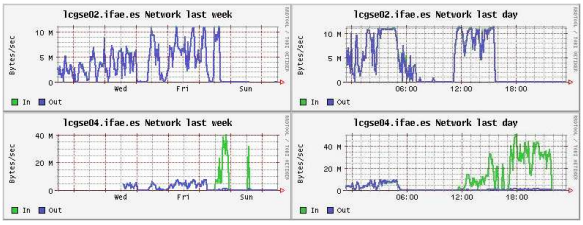
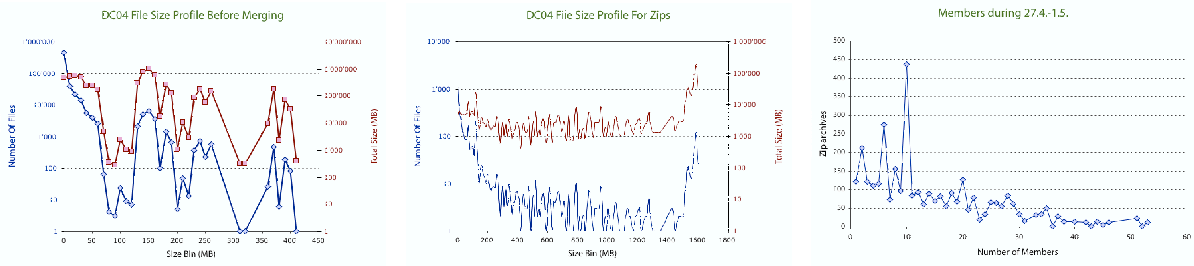


Figure 4: DC04 transfer test: PIC bandwidth usage.

FUTURE WORK

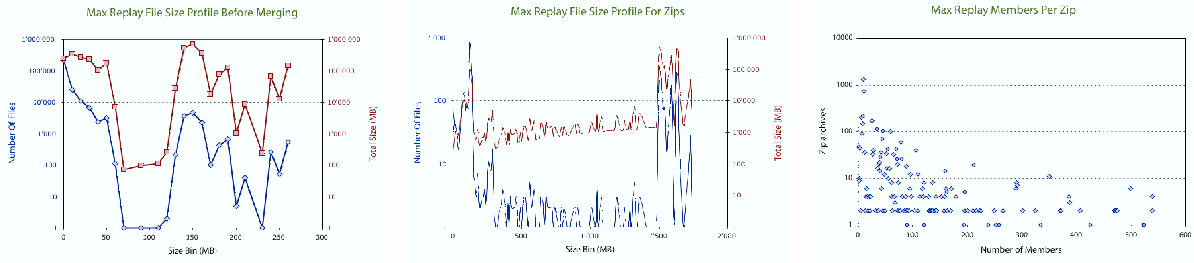
Several parties have expressed interest in this techology and CMS is in the process of making it more accessible. The C++ classes to read and write zip archives have been available through the LCG SEAL project for some time already. The storage factory plug-in interface and the modules to access different storage technologies through a

[1] Produced using PhEDEx replay infrastructure for what-if analysis, testing different algorithms, and making performance tests. The replay pushed about half of the DC04 data through a fake merging process.



(a) Size profile before merging (b) Size profile for zips (c) Number of members per zip

Figure 2: DC04 file size statistics



(a) Size profile before merging (b) Size profile for zips (c) Number of members per zip

Figure 3: DC04 replay file size statistics.

simple file-like abstraction are also being made available through the SEAL project. The ROOT plug-ins will become available through the LCG POOL project.

The file merging tools are available via PhEDEx [5]. Any zip-standard compliant tool can be used for merging, including the popular Info-Zip [12] command line tools `zip` and `unzip` available on many systems. The PhEDEx tools will work on most unix-like systems; they have been tested on Linux, Solaris and Mac OS X. The analysis and replay tools are also part of PhEDEx.

The ROOT developers considered the zip read/write technology sufficiently useful that they copied most of the plug-ins and zip file access code whole sale into their project and released it in a recent ROOT 4 version. Their implementation does not use the SEAL or CMS libraries, it has a separate copy of the code.

CMS is evaluating this technology further. We are improving the performance of the merging tools and making further simulations on the impact of the parameters on the file size, network transfer, mass storage and analysis performance. We are writing a guide on hardware configurations and deployment. We will also study the feasibility of using the same technology in CMS online farms.

ACKNOWLEDGEMENTS

We thank the CMS management and the DC04 crew.

REFERENCES

[1] D. Stickland, “Computing Models and Data Challenges of the LHC experiments”, these proceedings.

[2] A. Fanfani et al, “Distributed Computing Grid Experiences in CMS DC04”, these proceedings.

[3] D. Bonacorsi et al, “Role of Tier-0, Tier-1 and Tier-2 Regional Centers during CMS DC04”, these proceedings.

[4] The CMS Technical Proposal, http://cmsinfo.cern.ch/TP/TP.html

[5] T. A. Barrass et al, “Software agents in data and workflow management”, these proceedings. http://cern.ch/cms-project-phedex

[6] V. Lefébure, J. Andreeva, “RefDB: The Reference Database for CMS Monte Carlo Production”, Proceedings of CHEP’03, La Jolla, USA, March 2003.

[7] http://www.opengroup.org/onlinepubs/007908799/

[8] P. Mato et al., “SEAL: Common core libraries and services for LHC applications”, Proceedings of CHEP’03, La Jolla, USA, March 2003. http://seal.cern.ch/

[9] D. Düllmann et al, “POOL Development Status and Plans”, these proceedings. http://pool.cern.ch/

[10] http://root.cern.ch/

[11] http://www.pkware.com/company/standards/

[12] http://www.info-zip.org/