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

Tim Barrass,[†] Vincenzo Innocente,[‡] Lassi A. Tuura[§] (on behalf of CMS collaboration)

[†]Bristol University, UK [‡]CERN, Switzerland [§]Northeastern University, Boston, USA

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.

Problem statement

Typically high-energy physics data processing applications produce lots of files whose sizes are far from ideal when it comes to network transfers, mass storage systems and even read throughput for analysis programs. For reasons beyond the experiments control it is not reasonable to assume this imbalance will change soon. The median file size of 560 kB seen in the CMS DC04 data challenge in spring 2004 will remain more typical than the desired 1.5-2 GB. The processing of large input data samples need to be split into many jobs that run on grid farms. It would make sense to merge their output files to natural larger logical chunks provided that time to analysis does not significantly increase.

A number of merging options are available. In practise the cost of merging is lower if the files are merged without touching their contents or changing their catalogue identity. Not touching the contents also avoids violating the file immutablity assumptions made by grid tools and allows the merging to be done in any stage of the data management system. In short, a flexible and optional knob for tuning network and mass storage performance without interference to the core frameworks is desirable.

Strategy

CMS is experimenting with uncompressed zip archives for file merging. We store the original data files unmodified and uncompressed in a zip archive, effectively providing random-access byte-range inside the larger container archive. Data files are accessed through an abstraction library as seekable stroage objects with POSIX-like file semantics. Normal local files are an obvious storage default, but we also provide plug-ins for accessing files directly from the mass storage systems (Castor, dCache) and with standard network protocols (http, ftp, gsiftp, sfn). Any of these may be combined recursively with direct access to the members of zip archives, effectively serving byte ranges of larger files as virtual smaller files. This allows us to fool our database libraries (POOL, ROOT) into believing that they are reading normal files when they are in fact getting members of zip archives directly from a mass storage system there is no need to extract the members from the archive before accessing them.

Files are registered into catalogues using special URL syntax, co-operating with existing conventions and tools; as far as the rest of the system is concerned, the original files have not changed identity, they merely happens to have unusual physical file name URLs. An added advantage is that the number of connections to the mass storage system can be reduced substantially by opening the archive only once and sharing the connection for all the members.

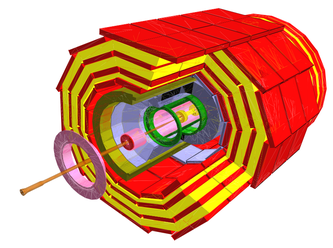
Our solution also includes the file merging tools: they either process files directly from a grid farm before passing them on to the data transfer system, or in a separate step to recluster the data. The merging allows for different queuing policies and strategies for different types of data.

Workflow and deployment

Our C++ utilities to read and write zip archives have existed for a while. The adaptors to expose members of zip archives as database files were introduced in late 2003 and early 2004. For the DC04 data challenge we introduced tools to evaluate the effectiveness of the merging to help us decide whether the technology is worth pursuing further and to estimate the advantages that we might gain. Our results are presented below.

References

CMS Software	http://cmsdoc.cern.ch/cmsoo/cmsoo.html
PhEDEx	http://cern.ch/cms-project-phedex/
LCG SEAL project	http://seal.cern.ch/
LCG POOL project	http://pool.cern.ch/
ROOT	http://root.cern.ch/
Zip file format	http://www.pkware.com/company/standards/
Info-Zip tools	http://www.info-zip.org/



merging

PhEDEx file transfer injection chain

The **CMS PhEDEx** project supplies an agent-based data movement system. Each agent performs a single well-specified task. A fraction of the agents are designed to be site-local, performing pre-processing tasks on 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. Each drop box agent does its transformation work on the drops before passing them to the inbox of the next agent. The last agent of the chain destroys the drops. For CMS each drop holds the information about the output of a single simulation or reconstruction job; the drops may be produced genuinely in real-time by a farm, or they can be generated from other sources such as the CMS RefDB production tracking database. More details about the agent workflow are given in a separate topic below.

File merging agent

The file merging agent is a drop box agent that receives the job output from upstream clean-up agents. It runs a master and number of parallel workers. The master manages the file queues and sends expired queues for the least-loaded worker process. The worker downloads the files from the mass storage system, creates the zip archive and generates new catalogue stub, and passes the result to downstream agents, which publish the merged file for transfer.

Master: queue management

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 is also able to prefetch files from the mass storage while idle to increase the overall system throughput.

There are no predefined queues. Instead queues are defined dynamically by the file meta data attributes. We used the attribute tuples **(dataset, owner, stream, file category, file type)** to define queues. Each queue has a configurable age and size limit. If either one expires, the queue is flushed. We used 1.5 GB and 30 minute limits for all queues. By customising the limits an experiment can apply a merging policy and transfer expediency class by the type of the contents of the files, for instance by physics streams. Queues may be linked such that if one of them expires, all the linked queues expire simultaneously. This is helpful to ensure that zip archives for related file types always include the output from exactly the same set of jobs.

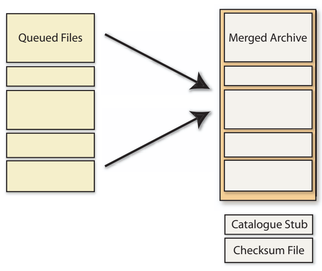
Because the queues are dynamically defined, the merging efficiency depends on the data mixture received by the agent. When a farm runs jobs for a mixture of datasets and the queue age limits are low compared to the output rates, files for different datasets are likely to get intermingled, leading to limited merging efficiency. On the other hand, if data is injected offline for entire datasets at once, thousands of files may get merged together.

Workers: zip creation

A 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. Each one does copies from the mass storage with a configurable number of parallel processes. In DC04 we used five parallel workers, each issueing up to five parallel file copies.

Meta-data preservation and distribution

We chose to **fully preserve the identity of the files.** The zip archives are invisible to much of the system. Archive members are registered individually into the file catalogues with complete meta data details. The only thing special about them are the physical file name URLs (see right). The worker agents generate a catalogue template that contains both an entry for the archive itself and copies of the entries for all the members. As the archives are transferred by the data movement system, the **file transfer agents add replica information** for both the zip archive and all the members; the archives are specially flagged so the agents know to perform the additional processing. The template catalogue is stored in the zip archive along with a simple template file list to simplify the registration process; the transfer agents substitute the local archive path in one of the two templates and register the replicas to the destination catalogue.



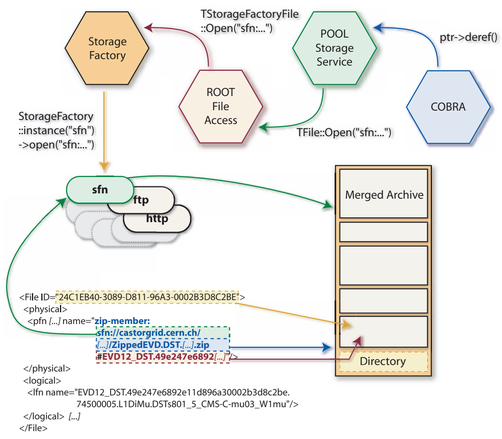
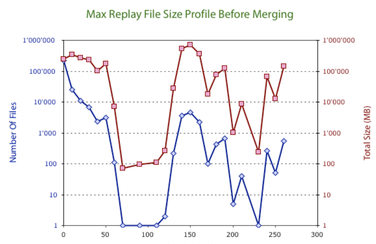
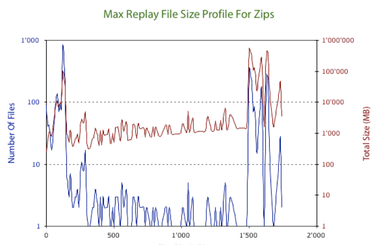


Table 1: DC04 replay file size statistics: maximum possible merging

	# of files	Total size	Minimum file size	Maximum file size	Average file size	Median file size
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*





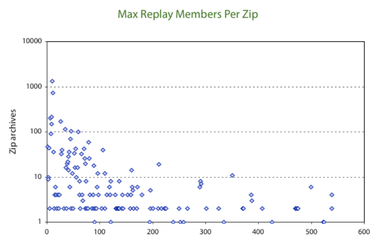
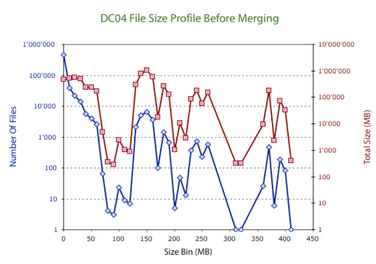
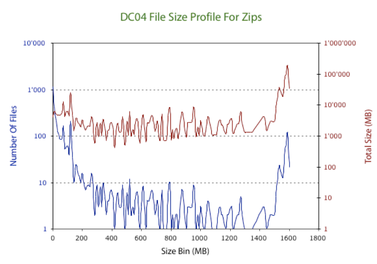
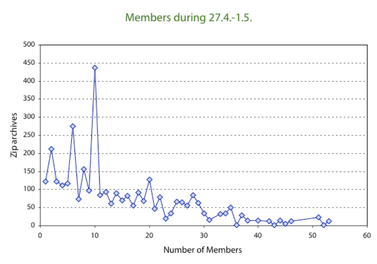


Table 2: DC04 file size statistics

	# of files	Total size	Minimum file size	Maximum file size	Average file size	Median file size
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







file access

The file access process is illustrated by the diagram on the left.

Storage factory

CMS has contributed to the LCG SEAL project C++ classes to read and write zip archives. In the end of 2003 and early 2004 we used them and the SEAL plug-in manager to implement a modular, plug-in based storage factory. The factory provides a simple interface to open and check the existence of files by URL. The factory uses the URL protocol to determine which plug-in is able to access the file and forwards the operations to the plug-in. Opening a file returns a pointer to a SEAL **Storage** object, an abstraction of seekable file-like objects.

Storage object plug-ins

Each storage factory plug-in is responsible for handling file operations for the protocols it is registered for. In several cases this translates to returning an instance of a suitable Storage class, such as File or RFIOFile. If there is no way to access the file directly remotely, as is the case with FTP for instance, the remote file is first downloaded to a local a temporary directory, and the plug-in returns a reference to the temporary file.

Some but not all protocols also support writing and creation of files. It is possible to write directly to both dCache and Castor, which provides an attractive 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 first.

Zip member plug-in

The zip-member storage plug-in is slightly special in that it first opens the archive by recursing back to the storage factory, then locates the requested member by name (a zip archive contains an index directory at the end), and then returns a sub-storage object for the byte range occupied by the member. Only stored, uncompressed members are supported as seeking in compressed data would be complicated, and in any case normally the data stored in the files is already compressed.

Since the zip member protocol supports only reading, each archive is opened only once in each thread. This produces a substantial savings in the number of file descriptors used and in the number of connections made to the mass storage servers.

ROOT adaptor

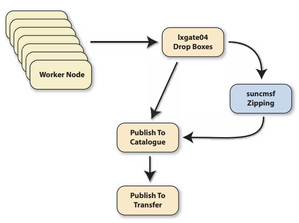
We also provide a plug-in to ROOT that redirects operations on ROOT s TFile to our storage factory objects. This allowed us to deploy our changes immediately without touching any of the database code in POOL or ROOT. We can also optionally hijack the URL resolution mechanism in ROOT to stop it from using it s internal plug-ins for accessing different storage systems, e.g. RFIO or dCache.

Catalogue URL syntax

The files are registered to a catalogue using the standard physical file name URL syntax. The protocol prefix is used to select the appropriate storage factory plug-in. Any of the following supported protocols can be used for file access:

- **file:** for direct local file access (operating system interface)
- **ftp:** for downloading files with FTP (using curl)
- **http**: or **web:** for downloading files with HTTP (using curl)
- **gsiftp:** or **sfn:** for downloading files with GridFTP (w/globus-url-copy)
- **rfio:** for direct file access with RFIO (aka Castor, no download)
- **dcap**: for direct file access with dCache (no download)
- **zip-member:archive#member** for direct access to a zip member; the archive part is resolved again through the storage factory mechanism, so it is possible to use for instance **zip-member:rfio:/path/to/a.zip#member**

workflow



The figure above illustrates the file **merging workflow** in the CMS DC04 data challenge. The CERN Tier-0 farm processed events at roughly 25 Hz, translating to approximately 250 MB in 19 files every 40 seconds. After writing the files to the Castor mass storage system, the job copied the necessary information to the transfer chain drop boxes, from where the files were published for transfer and distributed to the regional centres. The default file transfer mode was to simply publish the information in the file catalogues and make it available for transfer.

For the file merging test path the same files were also fed to the merging system on a different computer. The merging system maintained the queues for incoming files, merged files in the expired queues, put the resulting zip archives back to the mass storage and then fed the archives back to the data transfer chain with updated catalogue information. The archives were collected and held until the end of the data challenge, at which point they were injected into the transfer chain in one go as a transfer performance test. This resulted in sustained transfer rate from CERN to two Tier-1s of approximately 70 MB/s.

The final part of the workflow, not illustrated on this diagram, is the use of the data files in further analysis jobs. See the more detailed explanation on file access above right.

requirements

Data access overhead through the plug-ins seems to be negligible. We have not been able to measure any cost for the intermediate storage abstraction layers. On the other hand, our plug-ins caches connections by opening each archive only once per thread, leading to much fewer file opens to the mass storage systems. When relatively small amounts of data are read from many archive members, as summary data processing normally does, the performance advantage seems to be significant.

File merging is resource intensive even when the data itself is not touched and the files are simply merged into an uncompressed archive. A moderate level of CPU capacity is required to CRC-32 checksum the zip archive contents, however the main requirement is for **substantial I/O bandwidth.** Simultaneously balancing large network and disk reads and writes is particularly important. We tested four different systems until the capacity was sufficient for CMS DC04: three Linux configurations were rejected before we finally settled on a Sun Solaris system.

The first Linux system we tested had neither sufficient disk capacity, nor the I/O performance, nor the CPU performance to keep up with the incoming data (5-6 MB/s on average). The second system was otherwise identically equipped except for sufficient disk capacity. While the hardware (2 x 1 GHz Pentium III, 512 MB memory, 20 GB/7200 rpm EIDE drive) seemed adequate, the kernel appeared unable to feed enough data at the same time to and from the local disks. It appeared that the merging processes were read starved and the disks were simply trashing. We tested a third Linux configuration with RAID drives to verify this hypothesis, and indeed the I/O performance improved, but not enough. In appears that Linux 2.4 kernels do not seem to handle well several processes concurrently streaming O(1.5 GB) of data into and out of the disks. We finally settled on a 2-CPU Sun with SAN (RAID) disks. The machine had sufficient CPU, network and I/O bandwidth to keep up with the incoming data rate and to even recover from processing backlogs.

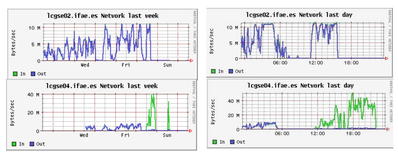
It is evident from our experience that it is important to deploy sufficiently capable hardware and operating system for the merging system.

results

Statistics

Table 2 above presents the file merging statistics from the CMS DC04. Merging ran for a few days at the end of the challenge, not all the data challenge data was merged. Still, this sample presents a rather significant increase in the file size: the average file size increased by a factor of 20, and more importantly, the median file size increased by nearly a factor of 60. More detailed plots of the file size profile before and for the zip archives are shown below the tables.

At the end of the of the data challenge all the zip archives were released into transfer to see the impact on transfer performance. As can be seen from the plots below (courtesy of Jos Hern ndez, CIEMAT), the transfer rates from CERN to PIC increased substantially for several hours, up to four-fold sustained rate to about 30 MB/s.



Data challenge replayed

PhEDEx also includes a replay infrastructure which allows us to perform what-if analysis, to test different algorithms, and make performance stress tests. We ran about half of the DC04 data through fake merging process to see what the maximum possible merging ratio would be. Results and statistics plots are shown in Table 1 above the DC04 results.

...and more

Code availability

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 simple file-like abstraction are also being made available through the SEAL project. The ROOT plug-ins will most likely be made more generally available through the LCG POOL project.

The file merging tools are available via CMS PhEDEx, a distributed data movement service project. There is little special about the tools, anyone can merge files into zip archives with any zip-standard compliant tool, including the popular Info-Zip command line tools zip and unzip available on many systems. The PhEDEx tools will too work on just about any unix-like system.

The ROOT developers considered the zip read/write techology 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 LCG SEAL libraries, it has a separate copy of the code.

Further studies

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, deploying the merging service in grid environments and data movement in general. We will also investigate in more detail the feasibility of using the same technology directly in CMS online farms.