Experience with highly-parallel software for the storage system of the ATLAS Experiment at CERN

T Colombo\textsuperscript{1,2}, W Vandelli\textsuperscript{2}

\textsuperscript{1} Università di Pavia, Pavia, Italy
\textsuperscript{2} CERN, Meyrin, Switzerland
E-mail: wainer.vandelli@cern.ch

Abstract.

The ATLAS experiment records proton-proton collisions delivered by the LHC accelerator. The ATLAS Trigger and Data Acquisition (TDAQ) system selects interesting events on-line in a three-level trigger system in order to store them at a budgeted rate of several hundred Hz. This paper focuses on the TDAQ data-logging system and in particular on the implementation and performance of a novel parallel software design. The main challenge presented by a parallel data-logging implementation is the conflict between the largely parallel nature of the event processing, especially the recently introduced event compression, and the constraint of sequential file writing and hash-sum evaluation. This is further complicated by the necessity of operating in a fully data-driven mode, to cope with continuously evolving trigger and detector configurations.

In this paper we report on the design of the new ATLAS on-line storage software. In particular we will discuss our development experience using recent concurrency-oriented libraries. Finally we will show the new system performance with respect to the old, single-threaded software design.

1. Introduction

ATLAS [1] is one of the four experiments installed at the LHC, CERN, Geneva, Switzerland. The ATLAS Trigger and Data Acquisition (TDAQ) system [2] is responsible for the selection and the conveyance of physics data, reducing the initial LHC collision frequency to a rate of stored events of several hundred Hz. Given the typical ATLAS event size of 1.5 MB, this corresponds to a throughput as high as \( \sim 1 \) GB/s. The TDAQ system includes O(20k) applications running on roughly 2000 nodes interconnected by a multi-stage Gigabit Ethernet network.

The ATLAS TDAQ is organised in a three-level selection scheme, including a hardware-based first-level trigger and software-based second and third level triggers. In particular, the second-level trigger operates over limited regions of the detector, the so-called Region-of-Interest (RoI). The last selection step, the Event Filter, deals instead with complete events. The TDAQ system is based on custom-designed multi-threaded software, mostly written in C++ and Java and running on the Linux operating system.

2. Data-Logging System

The ATLAS data-logging computer farm is the last stage of the data-acquisition system and is currently composed of 5 nodes. Events are written into binary data files that are then
asynchronously moved to the offline storage facility. Each node executes a so-called SFO (Sub-Farm Output) application which receives a subset of the events from the Event Filter farm.

The trigger classifies events into *streams* on the basis of either their physics content or offline use (e.g. detector calibration). At the data-logging level, events are then written into one or multiple files on the local disk volumes, respecting the stream associations. In addition, the ATLAS online system uses the concept of *lumiblock*: a lumiblock is a time interval during which beam, detector and data-taking conditions are considered stable. Files belong to physics analysis streams are indexed and closed at the lumiblock boundaries, simplifying the offline data processing.

The ATLAS binary file format is strictly sequential: this allows on-the-fly file hash-sum calculations and greatly simplifies the file writing task. However, it also makes it impossible to concurrently write multiple events to the same file. It is clearly still possible to concurrently write to different files. In addition to hash-sum calculation the SFO application also performs other CPU-intensive tasks. Event stripping is the process of reducing the content of an event to minimise the amount of data stored for calibration streams. More importantly, the aim of the discussed redesign is the introduction of online event compression. Differently from hash-sum calculation however, both stripping and compression can be performed in a parallel fashion without additional constraints. Besides data-handling tasks, the SFO publishes operational and event data monitoring information using the TDAQ services, stores meta-data about produced files, like location, size and hash sum, in a dedicated handshake database used to seed the offline processing and implements policies for a balanced usage of the local disk volumes.

The current ATLAS data-logging hardware has the following configuration:

- 2 E5520 2.4 GHz quad-core CPUs (16 SMT threads)
- 24 GB DDR3-1333 memory
- 3 Adaptec 5805 RAID controllers
- 24 1 TB hard-disks
- 2 Gbit Ethernet data links

The three RAID controllers implement one RAID5 volume each. The volumes are operated in a round-robin fashion avoiding concurrent reading and writing operations for the same volume [5].

### 3. Data Compression

The ATLAS binary data format uses standard frames, identified by specific headers and possibly trailers. These frames encapsulate the detector-specific data, which are created and formatted in the readout-electronics into a common lightweight structure. Even if compression and zero-suppression techniques are implemented in the front-end and readout electronics, the final event data is highly redundant in the information representation. This is due to the generally limited computing power available at the front-end, the high detector granularity and the requirement to operate at high speed with low latency. As a consequence, it was found that loss-less compression algorithms are extremely efficient in reducing the binary size of ATLAS events.

Considering the longevity, portability and stability requirements for the ATLAS data, the choice of a compression algorithm among the large number of commonly available ones became evident. The DEFLATE algorithm [3], and its most common implementation, the zlib library [4], is utilised in several key technologies with world-wide distribution: for example the HTTP protocol, the PNG and PDF file formats.

The zlib performances have been evaluated on samples of physics data, in particular to study the data compressibility and the effects of different compression levels. As shown in figure 1, the compression ratio is about 50% and is essentially independent from the compression level. The
Figure 1. Performance benchmark of the zlib compression library for ATLAS binary data.

compression time, on the other hand, rapidly scales up with the level. Defining a performance index as the inverse of the compression ratio and time product and normalising it to the value obtained for level 1, clearly shows why the lowest compression level is the best choice. Even in this best configuration however, compressing a single ATLAS event with zlib on a modern CPU takes roughly 70 ms.

4. Parallel SFO Design
The original SFO design [5], whose development started in mid-2000, was oriented toward single CPU or dual-CPU servers and did not expect any relevant CPU-intensive task for the application. It is therefore based on a very simple multi-threading design where a thread pool is in charge of the data reception over the network and a single writing thread takes care of data decoding and writing.

The introduction of CPU-intensive operations, like file hash-sum calculation and stripping, exposed the scalability limits of the old design. In particular the prospect of introducing online event compression made clear the need for a new design, oriented toward modern multi-core architectures. The conflict between the parallel event processing and the sequential file writing is the main challenge for a parallel implementation of the data-logging application. In fact, for each received event, the event header has to be decoded to find out the needed information about the event. Subsequently, stripping has to be executed for each stream requiring it, and
Figure 2. Architecture of proposed parallel SFO design.

finally data compression has to take place. Different events can clearly be processed in parallel, however the final writing operation, due to the sequential nature of the file format, must be serialised for events belonging to the same file.

 Several architectures have been considered and prototypes developed to find the best design meeting the above criteria. The result of these studies [6] is the design presented in figure 2. A thread pool takes care of the parallel, asynchronous processing of the incoming events. A unique global object, the Raw File Manager, acts as a proxy and serialises the access to the data files. The Raw File Manager in addition is responsible for other global tasks, like handling the local data volume balancing.

5. Task-based SFO implementation
The current implementation of the parallel SFO architecture introduced in section 4, the so-called SFOng, is heavily based upon and inspired by the Intel Thread Building Block (TBB) library [7]. The application workload is split into logically well-defined activities called tasks. In the SFOng two different type of event-handling tasks exist:

- Processing Task (PT): takes care of the processing of an event apart from the actual data writing. Each processing task will spawn a number of writing Tasks, one for each stream. Any number of processing tasks can be executed in parallel, since they all operate on different events.
- Writing Task (WT): each writing task is in charge of writing one event into one data file. As part of this operation the file hash sum is updated.

As shown in figure 3, both processing tasks and writing tasks are scheduled for execution in a common thread pool. The Raw File Manager however acts as a scheduling barrier, making
sure only one writing task per stream is active at any given time. In this way, the serialisation constraint is met, while allowing full parallelism for different stream and overall for the event processing, the latter being the most CPU intensive task. In addition, the use of a single thread pool allows for an implicit, automatic resource sharing between the processing and writing operations.

Due to the chosen task-based approach, an aspect that required additional care is the operational monitoring of the application state. In fact, since the definition of an activity and its actual execution are independent, the understanding of the software operational parameters and bottlenecks requires to know the task distribution within the different elements. A dedicated lightweight internal monitoring infrastructure collects and organises the operational information produced by the various subsystems as well as statistics on the processed events.

6. SFOng Performance
The final SFOng performance evaluation required the use of the production ATLAS data-logging hardware, the latter having the necessary qualities in term of CPU, network and disk capabilities to allow the full software characterisation. The DAQ system was configured to operate a single data-logging node at saturation. A sample of physics events was chosen to present a very non-uniform stream distribution in order to stress the file access serialisation and therefore the conflict with the parallel processing. The SFOng number of threads was manually changed to gauge the software scaling behaviour.

As reported in figure 4, the throughput scales linearly up to 8 threads, corresponding to the

Figure 3. SFOng task-based implementation. A detailed discussion is provided in the text.
Figure 4. SFOng scaling behaviour. The application, executed on the final hardware, was operated under a realistic workload and with data compression enabled.

number of physical cores in the machine. From 9 to 16 threads, one still observes a significant growth, exploiting the SMT (simultaneous multi-threading) capabilities of the CPUs. At the peak performance the application receives 180 MB/s of event data from the network and writes 120 MB/s of compressed data. The maximum network and disk throughputs for this hardware configuration, established with independent measurements, are about 240 MB/s and 350 MB/s, respectively. This indicates that the peak performance of the SFOng application is defined by limit of the CPU capabilities. This results in a 900% improvement with respect to the old SFO implementation, whose performance corresponds to the leftmost point in figure 4.

Another characteristic to be highlighted is the limited overhead introduced by the whole data-logging workload with respect to the pure data compression. At the peak throughput, the SFOng performance is only about 15% smaller than the parallel zlib compression rate of the same data set.

Despite the multi-threaded architecture the memory footprint of the SFOng application is very limited. The worker threads locally store only monitoring information, while the tasks carry references to the event data. As a result, the memory usage essentially scales just like the number of events handled in parallel. For maximum performance this number has to be of the same order of magnitude of the number of threads. As shown in figure 4, using more threads than SMT cores does not increase the throughput. Therefore we can expect the maximum number of events handled in parallel on current and future architectures to be $O(10)$.

7. Conclusions
A parallel implementation of the ATLAS data-logging software, the SFOng, was presented. It was shown that the new application is outperforming the old design. In particular, the SFOng is able to effectively exploit recent multi-core platforms, taking advantage of the SMT feature of recent processors. It is interesting to notice how the task-based approach, and more in general the TBB library, even if generally oriented to CPU intensive workloads, perform very successfully in the ATLAS data-logging mixed operation.
References
[1] ATLAS Collaboration 2008 “The ATLAS experiment at the CERN large hadron collider” JINST 3 http://iopscience.iop.org/1748-0221/3/08/S08003
[2] ATLAS Collaboration 2003 “ATLAS High-Level Trigger, Data Acquisition and Controls: Technical Design Report”, CERN/LHCC 22
[3] Deutsch P 1996 “DEFLATE compressed data format specification version 1.3”, Request for Comments 1951, Internet Engineering Task Force, http://www.ietf.org/rfc/rfc1951.txt
[4] Gailly J–L and Adler M 2012 “zlib” http://zlib.net/
[5] Battaglia A, Beck H P, Dobson M, Gadomski S, Kordas K and Vandelli W 2008 “The Data-Logging system of the trigger and data acquisition for the ATLAS experiment at CERN” IEEE Trans. on Nucl. Sc. 55 2607–12
[6] Colombo T and Vandelli W 2011 “Multi-threaded evolution of the data-logging system of the ATLAS experiment at CERN” IEEE NSS/MIC Conference Record 2039–43
[7] Intel Corporation 2012 “Threading building blocks” http://threadingbuildingblocks.org/