Boosting the data logging rates in run 4 of the PHENIX experiment

Chris Pinkenburg, BNL, Upton, NY, USA
for the PHENIX collaboration

Abstract

With the improvements in CPU and disk speed over the past years, we were able to exceed the original design data logging rate of 40MB/s by a factor of 3 already for the Run 3 in 2002 of the PHENIX experiment. For the Run 4 in 2003, we increased the raw disk logging capacity further to about 400MB/s.

A major improvement was the implementation of compressed data logging. The PHENIX raw data, after application of the standard data reduction techniques, were found to be further compressible by utilities like gzip by almost a factor of 2, and we defined a PHENIX standard of a compressed raw data format. The buffers that make up a raw data file consist of buffers that would get compressed and the resulting smaller data volume written out to disk. For a long time, this proved to be much too slow to be usable in the DAQ, until we could shift the compression to the event builder machines and so distributed the load over many fast CPU's. We also selected a different compression algorithm, LZO, which is about a factor of 4 faster than the "compress2" algorithm used internally in gzip. With the compression, the raw data volume shrinks to about 60% of the original size, boosting the original data rate before compression to more than 700MB/s.

INTRODUCTION

The PHENIX experiment [1] at the Relativistic Heavy Ion Collider (RHIC) consists of 4 large spectrometer arms, two central arms and two forward Muon arms (Fig. 1). There is a total number of about 400,000 readout channels, giving a typical event size of about 230KB/event before compression. The typical sustained readout rate is about 2.2KHz.

The network in the data acquisition system was upgraded to Gigabit. We doubled the local disk buffer to 8TB, which allows us to ride out short service interruptions of the HPSS-based tape robot, and gives the detector groups a window of several hours to access the data and perform monitoring and calibration tasks.

DATA FLOW

Fig. 2 shows a schematic view of the data flow in PHENIX. The detector signals are digitized in Front-End Modules (FEM) with a number of channels ranging between 12 and several hundred, depending on the detector. The digitized data are sent via optical fiber to Data Collection Modules (DCM), which receive the data, package

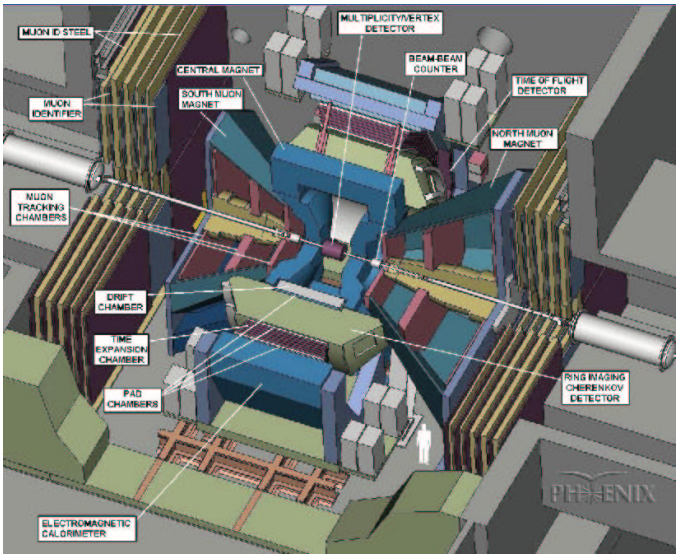

Figure 1: Overview of the PHENIX experiment.

them, and send the data on to the Event Builder, which assembles the event fragments into whole events (fig. 3).

The SubEvent Buffers (SEB) receive the data from the DCMs and send the parts from one event through a crossbar switch to an Assembly and Trigger Processor (ATP). While the number of SEBs is fixed by the way the front-end connections are structured, the number of ATPs is not fixed and more machines can be added to increase the computing power. The ATPs assemble the event fragments into a whole event and, because this is the first time that the data of a whole event are available in one place, runs a Level-2 trigger on the data.

The ATPs receive events on a first come, first serve basis, and send the accepted events on to "Buffer Boxes", which receive the events and store them on disk for later transfer to an HPSS-based tape robot system. Their primary role is to buffer the data until they can be shipped to the tape robot, and so decouple the data taking from possible downtimes of the tape robot.

DATA COMPRESSION

When the PHENIX Raw Data Format (PRDF) was specified, we found that utilities such as *gzip* could further compress the already zero-suppressed data by more than a factor of two. Using gzip however, one would need to restore the whole data file to its uncompressed length before reading it, which is undesirable. We therefore specified a com-

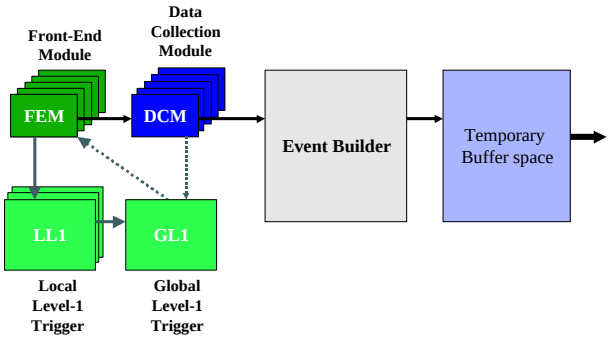

Figure 2: Schematic view of the data flow in PHENIX.

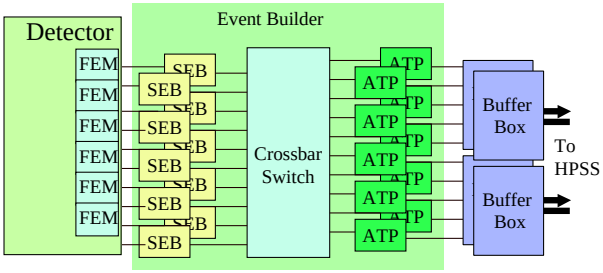

Figure 3: Overview of the data flow in the Event Builder. The data from the front-end is received by the Subevent Buffers (SEB). All parts of one event are sent to an Assembly and Trigger Processor (ATP), where the full event is assembled and the Level-2 trigger algorithms are run. The events are then sent to the Buffer Boxes, where they are stored on disk and later sent to a tape robot.

pressed raw data format where the data could be read and uncompressed on the fly. The raw data stream consists of buffers of about 8MB but variable length, and the length is always a multiple of 8KB. A typical buffer holds between 50 and 100 events. Originally this was put in place to increase the data integrity (if a corrupt buffer is encountered, one can salvage the subsequent buffers by skipping 8KB records until either the next buffer header or the end-of-file is found). The compression works on this buffer level. A complete buffer, including its header, is compressed by some algorithm. The result, typically much smaller than the original buffer, gets a new buffer header and is written out to the file instead (fig. 4). The buffer header makes this a legitimate buffer as far as the transport layers of the system are concerned. On readback, the buffer is recognized as one which holds another compressed one as its payload, which gets uncompressed. The original buffer is thus restored, and passed on to the next software layer, just as if it had been read from the file. In this way, the compression is handled at a relatively low layer in the I/O system, and completely transparent to the rest of the system.

Traditionally we used the same algorithm internally used by the gzip utility, “compress2”, which is readily available on most systems. However, while this algorithm achieves a good compression ratio (the compressed buffer shrinks

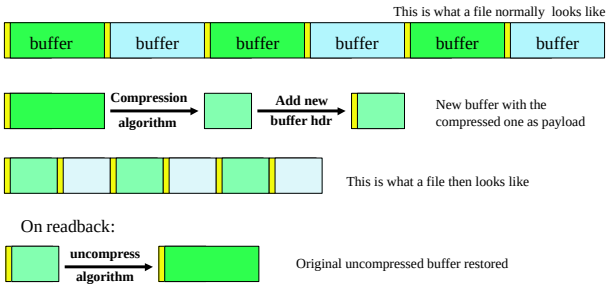

Figure 4: The compression works on buffers. A complete buffer, including its header (denoted in yellow) is compressed. It receives a new buffer header, and is written out this way. On readback, the original buffer is restored.

to about 45% of its original size), it is also rather CPU-intensive, which makes it too slow to be usable in the online system. We searched for a different algorithm which has properties comparable to *compress2*, but is much faster. We wanted an open-source algorithm which is robust, adheres to a published standard, and is available on most systems. We identified the family of the *LZO* algorithms [2] as a good candidate, and settled on the *lzo_1x* algorithm. It achieves a compression of about 50% on a typical buffer (slightly worse than *compress2*) but is at least 4 times faster (to compress one minute worth of data, *lzo_1x* takes one minute, while *compress2* takes 4 minutes).

This is still a considerable amount of CPU cycles, but the work is distributed over all ATP’s in the system, which compress each buffer before sending it to the logger.

We achieve a factor of two reduction of the data volume in this way, which fits twice the number of events into our bandwidth budget. In addition to increasing the number of events, this allowed to postpone the turn-on of the Level-2 trigger. Until the RHIC luminosity becomes so high that the bandwidth budget is exceeded, the level-2 trigger can be run to just tag, but not actually reject events. In this way we are able to better evaluate its performance and efficiency.

CONCLUSION

For the Run 4 of the RHIC accelerator at the Brookhaven National Laboratory, the DAQ logging capacity of the PHENIX experiment has been quadrupled by adding more disk buffer capacity and more file servers, and by using a revamped compression technology which squeezes twice the number of events into a given bandwidth budget.

With corresponding improvements in the detector front-end readout, we achieved a data rate of 2.2KHz without turning on the Level-2 trigger in rejection mode. Overall, due to the increased logging rates and the compression, we recorded about 1.5 Billion events in the Run 4 of the PHENIX experiment.

REFERENCES

[1] K. Adcox et al, *PHENIX detector overview*, Nucl. Instr. Meth A 499, 2003, pp 469-479.

[2] M. Oberhumer, *The Lempel-Ziv-Oberhumer data compression library*, http://www.oberhumer.com/opensource/lzo