CERES Data Acquisition System in 2000

D. Miśkowiec, J. Holeczek, H. Sako, H. Tilsner, J. P. Wessels

January 17, 2003

1 Introduction

The TPC readout system did not work properly in 1999. Moreover, Michael Richter left CERES in late spring 2000. In this situation CERES asked CERN for support. Luciano Musa proposed for the approaching 2000 run a new TPC readout scheme in which analog multiplexed signals were extracted from the front end electronics and digitized in flash ADCs (FEDC) of his design, housed in 9U VME crates. Markus Joos and Geraldine Thomas agreed to provide a software library for accessing the FEDC modules. More information about the new TPC readout can be found in the `note` written by Heinz Tilsner.

The data acquisition system (DAQ) needed to be adapted to the new readout. In the six months between March and September 2000 we designed and implemented a completely new DAQ. It was then successfully used to collect 30 millions central Pb+Au events in the year 2000 heavy ion run. Below we first describe the DAQ scheme and performance, then we list the most important dates, and finally we briefly describe the encountered problems.

2 Scheme

We wanted a simple system, and we wanted to use as many standard hardware pieces and standard software tools as possible. Also, we wanted some modularity so later we could optimize a part of the system without touching the others.

The final hardware and software schemes are shown in Fig. 1 and Fig. 2, respectively. Ten readout PCs (eight for the TPC, one for SDD, RICH1, and RICH2 and one for the VME crates in the grillage) were collecting events during the 5 s long SPS burst. In the 14 s long burst pause they sent the collected data via ethernet to an event builder PC in the CERN Central Data Recording facility (CDR). There the ten data buffers were merged into one and saved on disk. The tape daemon, asynchronously running on this machine, copied the file to tape. The cleaner process was deleting the oldest files such that half of the disk space was always available. Below I go through the various stages in more detail.

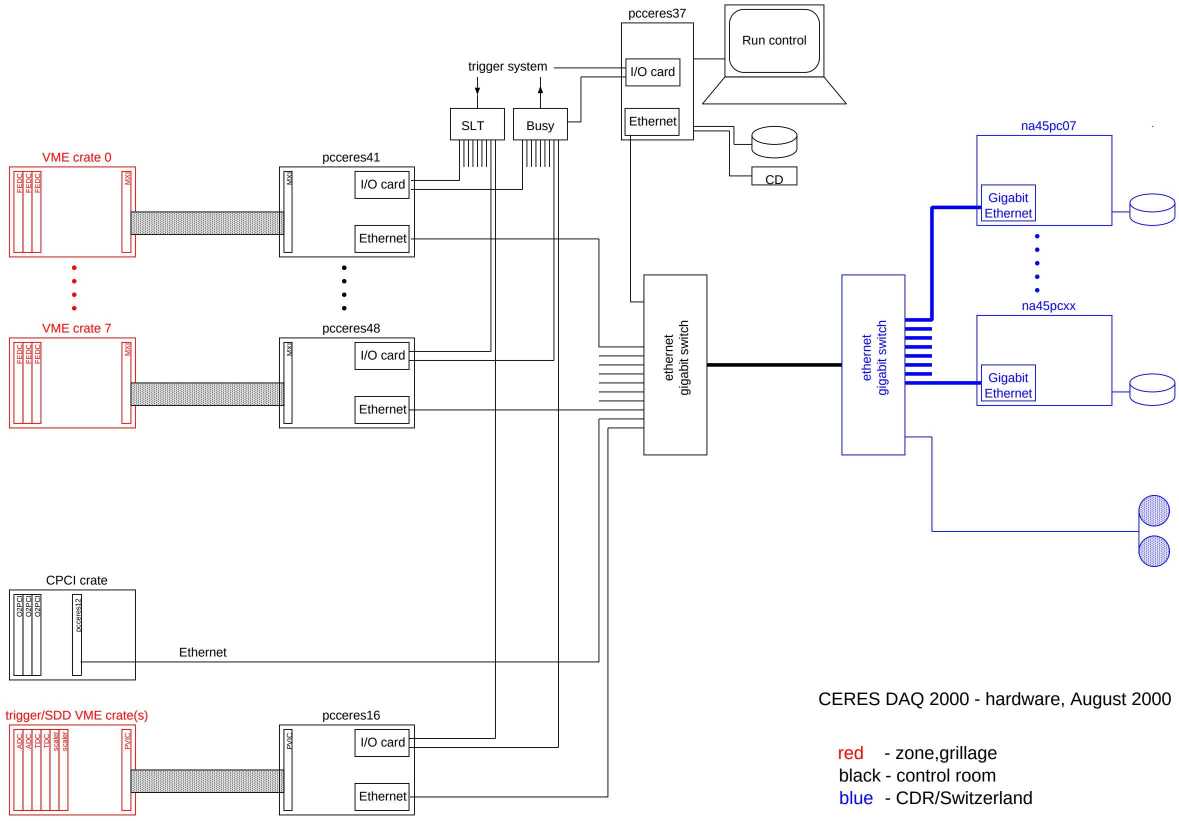

Figure 1: Hardware of the CERES data acquisition in 2000.

2.1 Collecting

Charge induced on a single TPC pad was sampled 250 times and the corresponding 250 analog charges were stored in the switch capacitor array (SCA). Subsequently, the signals of 16 pads, which belong to the same front-end board, were multiplexed and sent via a single LEMO cable to an input channel of a FEDC module. The multiplexing sequence was following: 0-th time bin of pad 0, 0-th time bin of pad 1, ..., 0-th time bin of pad 15, 1-th time bin of pad 0, ..., 249-th time bin of pad 15. Since one TPC chamber has 60 front-end boards, and since we were using 40 of 48 input channels of each FEDC, the signals of two TPC chambers could be handled by three FEDC modules. These three FEDC modules were sitting in one 9U VME crate. This crate was connected via a MXI interface (12 MB/s in the block transfer mode) to a readout PC. Eight readout PCs were thus needed to read the total 16 TPC chambers. More information about the TPC readout can be found in the `note` written by Heinz Tilsner.

SDD, RICH1, and RICH2 sent their data to the receivers which sit in the respective three crates. From the receivers, the data were transfered into the memory modules in the same crates. The memory module data were then sent via an optical link and an O2PCI module sitting in a CPCI crate directly to the memory of a PC connected to this CPCI crate. In 1999 this connection was realized by a PVIC interface, and the data transfer was triggered by a process running on an additional embedded PC sitting in the CPCI crate. Since in 2000 the TPC data was going via a different path,

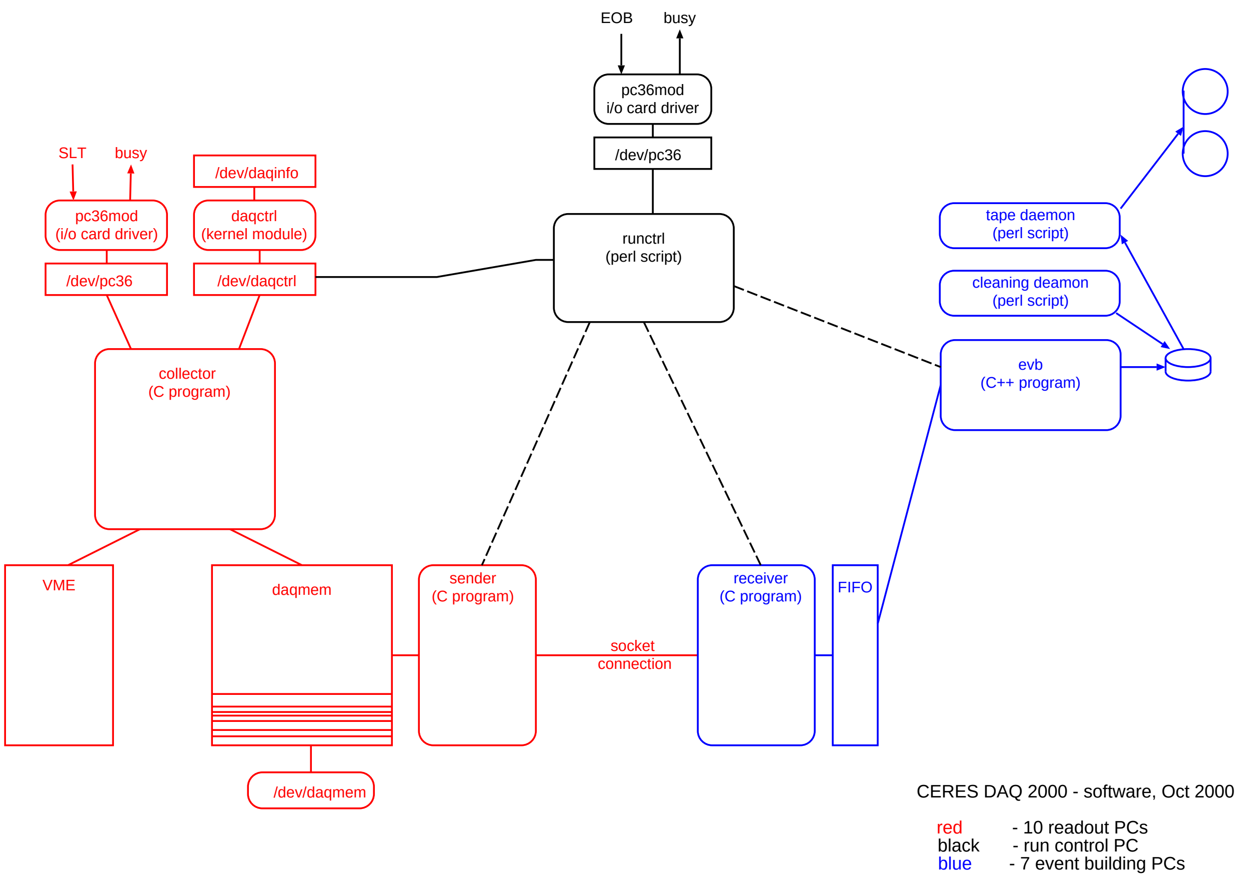

Figure 2: Software of the CERES data acquisition in 2000 – snapshot at the end of the run.

and only SDD, RICH1, and RICH2 needed to be readout via CPCI, the 1999 solution seemed to be an overkill. In addition, the existing software was difficult to handle. In this situation we decided to simplify the system by eliminating PVIC and storing the data in the memory of the embedded PC, physically sitting in the CPCI crate.

The discriminators, coincidences, and downscalers used for the trigger logic, ADCs and TDCs of the beam related photomultiplier detectors, and counters of various beam and trigger signals were sitting in three VME crates located in the grillage. These crates were daisy-chained via VME extenders, and connected via a PVIC interface to another readout PC.

In the case of the TPC and the grillage, the readout was triggered by an external signal applied to an input channel of an I/O card (PC36C by Eagle Technology, purchased from Meilhouse Electronics) plugged in the ISA bus of the corresponding readout PC. The collector software was polling on the card. Once a trigger has been seen, the PC would set a busy signal on an output line of the I/O card. A logic OR of all busy signals went to the trigger system and inhibited new triggers. The busy signal was removed only after the complete data had been in the memory of the PC (unless in the pipeline mode, see below). The reaction time (defined as the delay between the trigger and the busy) was initially tested to be about 3 microseconds. Under realistic data taking conditions it grew to 50 microseconds. A device driver `pc36mod.c` was used to access the I/O card. A process could open `/dev/pc36_0` and read a byte containing the states of the 8 input lines, or write a byte to set the output lines.

The tenth readout PC, the embedded PC in the CPCI crate, did not use an I/O card. Instead, the collector process running on this machine was polling on three memory locations, corresponding to the three detectors to be readout, which were overwritten by the O2PCI interfaces each time a new event arrived into the memory module.

Since the TPC readout hardware (crates and FEDC) was supposed to be available only very short before the start of the run, we developed a fake collector which was generating data rather than getting them from FEDCs. This allowed us to test the other parts of DAQ before the FEDCs were built.

2.2 Storing

On all ten readout PCs the data were collected in the upper most 64 MB of the physical memory. This memory was disabled for linux by putting

```
append = "mem=32M"
```

in /etc/lilo.conf on pcceres12 (which had 96 MB of physical RAM) and

```
append = "mem=64M"
```

on all other readout PCs (128 MB physical RAM). A memory device driver (similar to mem.c) `daqmem.c` was used to access this area. The collector processes would open and then mmap `/dev/daqmem` to get a virtual pointer. From the system level the user could handle `/dev/daqmem` like an ordinary disk file, including dumping, editing, copying, etc.

The collecting was controlled by the following structure, residing in the kernel memory:

```
struct daqctrl_type {
  int branch;                          /* 0-7 TPC; 9 grill; 10 SD+RICH */
  int run;                             /* current run number   */
  int burst;                           /* current burst number */
  int nevents_collected;               /* number of events in daqmem */
  int nbytes_collected;                /* number of bytes in daqmem */
  int collecting_status;               /* 0-waiting for trigger;
                                          1-trigger received;
                                          2-writing event into daqmem;
                                          9-daqmem full */
};
```

Before storing a new event in `daqmem`, the collector process read the run, burst, and event number (equal to `nevents_collected`) from this structure and generated the event header. The variable `nbytes_collected` would tell the collector where to store the event. After the event was stored, the collector incremented `nevents_collected` and `nbytes_collected`. Later, when the burst was finished and the data needed to be sent to CDR, the sending routine used `nbytes_collected` to figure out how many bytes to send. After the data have been sent, it set `nevents_collected` and `nbytes_collected` to zero and incremented `burst` by one. The access to this structure was provided via the device driver `daqctrl.c`. A process could open `/dev/daqctrl` and read a line which contains all the 6 values, or write to `/dev/daqctrl` following a predefined syntax. From the system level the user could cat `/dev/daqctrl` or echo to it. The following examples show how to set these variables from the system level using echo:

```
echo "run 1123" > /dev/daqctrl
echo "burst 12" > /dev/daqctrl
echo "collecting_status 9" > /dev/daqctrl
echo "incr_nevents 1" > /dev/daqctrl
echo "zero_nbytes 0" > /dev/daqctrl
```

The control variables were, in addition, accessible via `/proc/daqctrl`.

2.3 Sending

In the burst pause the ten readout PCs sent the first `nbytes_collected` bytes from `/dev/daqmem` to an event builder PC in CDR. The sending speed of each readout PC was limited to about 10 MB/s (fast ethernet). Once all the buffers arrived at CDR, the event builder would start merging them into one file. This was taking longer than one burst and thus the data of the next burst had to be sent to a different CDR machine. In total seven event builder machines were available. Various topologies and transfer methods have been tested.

Topology

The most attractive topology is that of 10 readout PCs sending to 1 event builder PC (10 to 1). The ethernet interface of the latter machine is the bottle neck. The event builder reads and writes files locally and thus it is fast.

Sending to two PCs (two times 5 to 1) is nearly two times faster but the event builder has to read half of its input files via nfs.

Sending to five PCs (five times 2 to 1) is very fast. The event building, however, cannot be done at all and the buffers have to go to tape separately.

Data transfer method

The disks (or ramdisks, see below) of the event builder machines can be nfs-mounted on the readout PCs. Then the data can be transferred by a simple `cp`. The data rate, however, is only about 5 MB/s.

Alternatively, the buffers can be sent via rfcp (CERN remote copy routine, based on rfio) or ncftpput (non-interactive ftp), both these methods yielding up to 20 MB/s.

Finally, the data can be transferred via sockets. The sender process on the readout PC is trying to connect to a socket on the event builder machine. Once the connection is established, it sends the run number, the burst number, and the data length in bytes, followed by the data buffer. The receiver process on the event builder machine opens a socket and listens on it. Once it receives the data size, it allocates an array of the appropriate size and starts receiving data in blocks of 8 kB. The data are stored in the allocated array. Once the number of received bytes is equal to the expected data size, the receiver process closes the socket connection, opens a FIFO (named pipe) with a name containing the run and burst numbers, and dumps the data into this FIFO. Once the whole data has been dumped, the receiver exits.

Storing buffers on the event builder PC

The simplest way is to save the buffers on disk. We also tried to distribute the ten buffers over several local disks in order to minimize the disk head activity.

Since the disk access limits the data rate to something like 10 MB/s while the pure memory access speed is about 200 MB/s, we considered storing the buffers on a ramdisk. The gain should be twofold, during writing and during reading.

In the tests, however, we did not see much difference between writing on disk and writing on ramdisk. This is because of caching. For the same reason it nearly makes no difference whether one or more disks are used.

Storing the data in the memory of the receiving processes is an obvious choice when the data were sent via sockets. The data can then be extracted via FIFOs. The event builder can read FIFOs faster than it would read disk or ramdisk files.

The data could be piped through gzip. However, the gain is negligible because the zipping reduces the data volume by only 20%.

Various combination of the described methods have been tested. The most important tests are listed below. The individual buffers had about 40 MB each, and the total data size was 350 MB.

Our final choice was sending 10 to 1 via sockets. The sender and the receiver processes were started new for each burst. Initially this was done by remote shell commands (`rsh`), later via telnet (see section 2.8). Starting these processes via telnet decreased the overhead. 200-250 MB of burst data could then be sent within the 14 s burst pause. This was the bottle neck of the DAQ.

One probably could improve the performance by implementing a perl-shell (public domain source code exists) on the CDR side of the telnet connection and implementing all important actions as perl functions (rather than calling bash shell scripts which internally again use perl).

Table 1: Sending methods.

topology	transfer	storage	speed	comment
local	memcpy	memory	180 MB/s	array to array copy within process
local	cp	ramdisk	40 MB/s	ramdisk to ramdisk copy
local	cp	disk	10 MB/s	disk to disk copy with sync
1 to 1	ncftpput	disk	10 MB/s	fast ethernet limit
10 to 1	ncftpput	disk	17 MB/s	the simplest solution
10 to 1	ncftpput	2 disks	19 MB/s	
10 to 1	ncftpput	4 disks	19 MB/s	
10 to 1	ncftpput	ramdisk	17 MB/s	
10 to 1	ncftpput	/dev/null	28 MB/s	
10 to 2	ncftpput	ramdisk	38 MB/s	event building slow
10 to 2	ncftpput	disk	38 MB/s	event building very slow
10 to 5	ncftpput	disk	62 MB/s	event building impossible
1 to 1	sockets	FIFOs	30 MB/s	this was finally used

2.4 Event building

The event builder (evb) process was called by the run control script on one of the event builder machines (na45pc07, 08, 09, 10, 11, 12, and 14) which has received one-burst data, just after the data transfer has been done. Since each evb system was completely symmetric with respect to each evb machine, run control could call it in any order of machines after each data transfer. In case one of the machines had a problem (crashed, or disk space full), the machine could be simply removed from the list of evb machines (done in run control).

The evb executable is based on C++. In each event, evb was checking checks consistency of event counters in the event headers (event header counters) with the evb's internal event counter among all 10 branch data, merging them, and creating the new event header with the recalculated total event size. If one of the branches had a wrong event header counter, this branch was not written to the output data. The executable was wrapped by a Perl script (`doEvb.pl`), which checked existence of input files, selected the output hard disk according to free disk space, and created status files for run control and link files for the taping script.

There were 3 versions of evb during the run according to the I/O scheme;

1. Local I/O version
2. rfio version
3. FIFO input version (final version)

Version 1 used standard C++ I/O (iostream class) and this could be used only with local input / output data. The process speed of evb was typically ~ 10 MB/s. Version 2 has been designed for the data sending scheme to more than one evb machines. In this case, evb ran on one of the machines which had received data, and read local data as well as remote data through rfio protocol. The process speed with reading and writing local data was as fast as in Version 1. With reading remote data and writing local data, the process speed was $\sim$ 4 MB /s in average. With both reading and writing remote data, it was only $\sim$ 2 MB/s in average. In Version 3, the whole input data were first read from FIFO's (named pipes) to memory buffers, and output data was written directly on the disk. The process speed was much higher, 15 $\sim$ 17 MB/s, for the total input data size of 150 $\sim$ 190 MB.

Since synchronousness of the module event counters (event counters of each detector) was not checked in evb, it was monitored from time to time with the evb log during the run. However, it was possible only when no after-protection is included in the trigger, since SDD and RICH count first-level triggers while TPC counts second-level triggers. With after-protection, the only way to check it was to compare module event counters of RICH and SDD, and TPC with scaler values of first level and second level triggers, respectively, in Grillage data. This method was once checked to work during the run.

The synchronousness was good except for the following problems during the run. When SDD busy happened, module event counters of SDD and RICH were sometimes off by one. A few times during the run it happened that either in SDD or TPC (SDD much more frequent than TPC), a branch data was corrupted (evb could not find the next event header). The reason was unknown.

2.5 Taping

A tape control script (`taper.pl`) was running on na45pc13. It sorted link files created by evb into tape job (tape-daemon) queues, and submitted them on each evb machine. Typically the total number of jobs was $1-3$ over 7 evb machines. The maximum number of tape-daemons on each evb machine was set to 3, so that the taping system had enough speed tolerance ($3 \times 7 \times 5$ MB/s $= 105$ MB/s) to deal with high maximum speed required when disk space was becoming full or after taping had been failed for a long time. The maximum possible constant data sending rate from DAQ was $\sim$ 20 MB/s.

Writing data to tape in tape-daemon was done with tpwrite (CERN SHIFT software package) command. The average speed of tpwrite was $\sim$ 5 MB/s. Each tape (Redwood) had 50 GB, 45 GB out of which was used to write data.

A typical tape failure rate (tape stuck in the drive during tpwrite or dump command) was 1-2 tapes / day during the run. Typical repair time by CERN tape support was very long (2-5 days). When the failure happens, the rest of queues for the tape was moved to the next tape in the script.

2.6 Cleaning

A data cleaning script (`cleaner.pl`) was running on all the 7 evb machines, to keep the total disk space less than 50 % on each machine. Of the data files which had been written on the tapes, one with the smallest run number and the smallest burst number was deleted first.

2.7 Monitoring CDR

Monitoring of evb, taper, cleaner, and evb machines status was done with Perl/Tk-based code `cdrmon.pl` running on pcceres37.

2.8 Run control

The run control script `runctrl-tk.pl`, written in Perl-Tk, was used to start and stop runs and to monitor the DAQ behavior. It was detecting end-of-burst and initiating sending of the collected burst data to CDR. At startup the script was opening telnet connections to all readout PCs and event builder PCs. The communication with these machines proceeded then via the telnet sessions.

The run control communicated with the readout PCs via their `/dev/daqctrl` interface, and with the event builder machines via status files in the shared directory `/cdrShare01/cerescdr/cdr2k/evb/sta/`. In addition, the run control was running remote `ps` to estimate load on the event builder machines and to send the next burst to the idlest of them.

2.9 Machine configuration

Ten readout PCs and one server PC were used on to collect data. On the CDR side, up to seven machines were used to receive the bursts, do event building, and copy the data to tape. These machines were called event builder PCs. One more PC synchronized the taping. The machine parameters are given below. All machines were running RedHat Linux 6.1/i386, except pcceres12 which was running SuSE Linux/i386.

Some special, very mysterious and extremely dangerous settings were necessary to make the system run. These were performed under cover of night and in disguise by our special agent JH. The secrets of these setting will be revealed, for the first time, in our distinguished paper. Persons with heart diseases are discouraged from further reading. We take no responsibility whatsoever for any accidents resulting from not having followed this advice.

This concerns interrupts. In all machines using the ISA based I/O card PC36C they needed to be manually setup in BIOS. PC36C is not a PnP card. It uses up to three interrupt lines selected by on board jumpers. Thus, one needed to reserve these interrupts (set them as "legacy ISA") in BIOS. As the total number of free interrupts is very small (it is a PC hardware problem in general), additionally one needs to switch off assigning IRQ line for VGA card, block USB and COM2, not used in these PCs. Afterwards one should inspect the `/proc/interrupts` and check that all devices are assigned separate interrupts. If this is not the case, for example the ethernet card shares

Table 2: Machines used in the CERES data acquisition in 2000.

name	task	processor	mem	ethernet	linux
pcceres41	TPC readout	PIII 666 MHz	128	fast	2.2.16
pcceres42	TPC readout	PIII 666→600 MHz	128	fast	2.2.16
pcceres43	TPC readout	PIII 666→PII 400 MHz	128	fast	2.2.16
pcceres44	TPC readout	PIII 666→PII 450 MHz	128	fast	2.2.16
pcceres45	TPC readout	PIII 666 MHz	128	fast	2.2.16
pcceres46	TPC readout	PIII 666→600 MHz	128	fast	2.2.16
pcceres47	TPC readout	PIII 666 MHz	128	fast	2.2.16
pcceres48	TPC readout	PIII 666 MHz	128	fast	2.2.16
pcceres16	grill readout	PII 266 MHz	64	fast	2.2.16
pcceres12	SDD/RICH r/o	P MMX 200 MHz	96	fast	2.2.13→16
pcceres37	server, runctrl	two PIII 500 MHz	512	fast	2.2.16
na45pc07	event builder	two PIII 450 MHz	512	gigabit	2.2.16
na45pc08	event builder	two PIII 450 MHz	512	gigabit	2.2.16
na45pc09	event builder	two PIII 450 MHz	512	gigabit	2.2.16
na45pc10	event builder	two PIII 450 MHz	512	gigabit	2.2.16
na45pc11	event builder	two PIII 450 MHz	512	gigabit	2.2.16
na45pc12	event builder	two PIII 450 MHz	512	gigabit	2.2.16
na45pc14	event builder	two PIII 450 MHz	512	gigabit	2.2.16
na45pc13	taper	two PIII 450 MHz	256	fast	2.2.12-20

an interrupt with the MXI card, one needs to move one of these cards into another PCI slot.

At the end we actually did not use the PC36C interrupts. The software was simply polling on an input channel of this card. Nevertheless, the interrupts were configured and ready to be used if necessary.

The second topic is the network hardware. Most of our DAQ PCs were equipped with Intel EtherExpress PRO/100 cards (the driver module is called eepro100.o). These are very cheap cards, nevertheless they behave quite well. We were able to saturate the 100 Mbs lines without problems. The only real problem that we have met was found after we (mea culpa, mea culpa, mea maxima culpa) upgraded the kernel from 2.2.12-20 to the latest RedHat 2.2.16-3. Almost every second reboot failed then to properly start the network. Nothing could be done except a new reboot (preceded with a pray that this time it will boot properly). This problem was mostly (but not completely) cured after we installed the newest network drivers taken from `http://www.scyld.com/network.` (As far as I can judge it, the problem was/is related to the kernel version AND to the VIA chipset based motherboards.) At a certain point one of DAQ machines was equipped with a DEC Tulip (dc21x4x) card, but we have found that the performance was slightly lower as compared to eepro100. I think that the problem lies in the network driver for this card. Actually there are at least two drivers (modules) that exist for these cards (I don't remember which module was used). Each of them can be "tuned", of course, so most probably one can get better performance, but we had no time (during the last experimental run) to play with it (if I remember well, we were getting about 7.5 MB/s transfers with these DEC Tulip based cards, while with eepro100 we could actually almost saturate 100 Mbs).

All these DAQ PCs (staying in Prevessin North Area) were then connected to a 3Com 100 Mbs switch (also in Prevessin) which, on the "other side" had a fiber link to CDR (CERN Central Data Recording facility in Meyrin). Data "collected" by DAQ PCs was then sent to event builder PC machines staying in CDR (in Meyrin). Event builder PCs were equipped with Gigabit ethernet cards of the type "Netgear GA620". We were not involved in choosing/maintaining this setup. For deeper informations you need to contact people in the CERN Computing Department: Catherine Charbonnier, Bernd Panzer-Steindel, and Andreas Pfeiffer who were/are directly responsible for these machines. These event builder PCs were connected (via their Gigabit ethernets) to a Gigabit switch in CDR (in Meyrin). A special note should be added here for 3Com "fast" switches. In some tests (performed not by CERES, however) it was found that under heavy load they can go "crazy". The solution was/is to upgrade their internal bios to the latest software version. You should take care that ALL switches between your DAQ PCs and event builder PCs are brought to the latest state (in our case there were 6 switches involved on the route from our DAQ PCs to our event builder PCs). The following people from IT/CS who did the job for us: Marc Collignon and Eric Sallaz. Some words of warning should be issued. According to experience, today's PCs are able to transfer data via network with a maximum speed of about 35 MB/s. That means that if you send or receive data using a 100 Mbs card, you will easily get 10 MB/s in both directions simultaneously (modern cards are full duplex). If you, however, try to do the same using a Gigabit card you will NOT get 10 times more (i.e. 100 MB/s), but you will be limited to about 35 MB/s. This is not only valid if you try to send data between two PCs equipped with Gigabit cards, but also if multiple PCs send their data to a single PC equipped with a Gigabit card. For example, in our setup, that means that as soon as more then 3 DAQ PCs (using 100 Mbs cards) send their data to a single event builder PC (equipped with a Gigabit card), they will not be able to send their data with full 10 MB/s speed. This is clearly a limit in the PC that receives the data and NOT a limit of the ethernet "medium" (i.e. fiber link, for example). You can saturate the Gigabit ethernet transferring data between many different machines. For example, if you take 3 pairs of PCs with Gigabit ethernet, each pair will be able to transfer data with a maximum speed of 35 MB/s, but in total you will get 3*35 MB/s = 100 MB/s in the Gigabit switch (don't forget to upgrade the bios of this switch :-). Note also that it takes quite a lot of CPU power to "move" data over network. Expect even 100 percent CPU usage under heavy load. Last, but not least, for security reasons, none of our DAQ PCs nor event builder PCs were accessible from outside of CERN (we did not use any special firewalls, we fully relied on the standard CERN policy in this matter). And that would be it.

3 Performance

The typical event size for a central Pb+Au collisions at 160 GeV/nucleon was 500 kB. The CPCI branch (SDD and both RICHs) had 20% more data than a single TPC branch. The data volume from the grillage branch was negligible. The average busy duration, i.e. the average time needed to get the event into the memory of a readout PC, was 1.7 ms for grillage VME crates, 3.8 ms for SDD, 2.3 ms for RICH, and 5.7 ms for TPC. The largest fraction of the TPC time, about 5 ms, was spent in the data transfer from FEDC to the memory of a readout PC. This transfer was going via MXI interface with the rate of 11-12 MB/s. The collecting time, thus, limits the data which can be taken during a 5 s burst to about 500 MB, corresponding to 1000 events, when running with infinite trigger candidate rate. With the reasonable beam intensity of 10^6 per burst, with the beam before- and after-protections of 1 microsecond each, and with the centrality trigger of 8%, the rate of offered and accepted triggers per burst was 1000 and 300, respectively.

In the middle of the lead run, on October 10, a pipeline readout scheme was implemented for the TPC. The 5.7 ms were split in two parts: $\sim$1 ms, needed for the data to get in the ALTRO chips of the FEDC, and $\sim$5 ms for the transfer to the readout PC. The number of accepted triggers went up to 400 per burst. A similar pipeline for SDD would further increase the event rate to about 480. For this one would have to reprogram the chips on the SDD control module. Since we had no spare control modules, and since it was not quite certain whether we would be able to restore the old settings in case of failure, the collaboration decided not to take the risk.

The resulting 200 MB per burst could, in most cases, be sent to CDR without problems. This, however, was close to the limit. Decreasing the centrality threshold to 12%, which would allow to take additional less central events practically without loosing the most central ones, would increase the data volume to 300 MB/burst. Sending this amount of data was not possible in the 10-to-1-via-sockets scheme.

4 History

Feb 23	Luciano proposes new readout scheme
Mar 1-8	we collect offers for embedded processors
Mar mid	we decide to use MXI, available from CERN electronics pool
Mar 22	speed tests of I/O cards
Apr end	ordering readout PCs with ISA slot
May 9	Heinz wants to test his front end boards
May 9	first readout PC from ELONEX delivered
May 15	I/O cards delivered
May 17	readout PCs delivered
May 17	racks in the zone installed
	one VME with 2 ALICE FEDC and MXI connected to pcceres40
	Heinz uses Labview to read the data
May 26	daqmem driver corrected by Michael
Jun beg	TPC faker ready
Jun 20	I/O card interfaces ready
Jul beg	collector software for ALICE FEDCs ready
Jul beg	ramdisks in CDR; ncftpput
Aug 14 or bef	first 2 CERES FEDCs in the zone
Aug 16	3 VME crates and 9 CERES FEDCs in the zone
Aug 17 or bef	collector software for CERES FEDCs ready
Aug 30	start of proton beam
Sep 2	7 TPC crates, SDD, and RICH readout; CPCI data bad
Sep 6	rearrangement of crates in grillage
Sep 8	stripped collector used to debug the TPC length problem
Sep 10	end of proton beam
Sep 11	all-day test of data taking, sending to ramdisk CDR, taping
	sending to 1 - 15 MB/s, evb 10 MB/s, burst up to 160 MB
	sending to 2 - 25 MB/s, evb 5 MB/s
Sep 13	pcceres46 hanging-while-sending
Sep 14	CPCI readout by embedded PC (no PVIC) - collector ready
Sep 18	ramdisk has no advantage over disk+cache (when writing)
Sep 20?	start of 80 GeV lead beam
Sep 20	run 1063 complete setup readout
	pcceres12 hanging
Sep 21	sending: rfcp replaced by ncftpput
	sending to 5 eb machines
Sep 22 or bef	sending via sockets
Sep 24	Sunday 80 GeV run; problem: hanging-while-sending
Sep 24	pcceres41 replaced by pcceres49
Sep 25	Monday morning, end of 80 GeV lead beam
Sep 26	start of 160 GeV lead beam
Sep 26	pcceres42, 46 hanging-while-sending

Sep 29	runctrl logged numbers were underestimated; corrected now
Oct 1 or bef	some readout PCs (42,...) replaced by desktop PCs, slower
Oct 2	1.5LT implemented, beam after-protection activated
Oct 3	CPU replaced in 42,43,44,46 → hanging-while-sending fixed
Oct 4	autofix SDD busy
Oct 7	autofix TPC busy
Oct 9	TPC pipeline test for one branch
Oct 10	TPC pipeline
Oct 14-16	Tetris in Rich
Oct 16	pcceres12 kernel upgrade from 2.2.13 to 2.2.16
	→ pcceres12 network hangup problem fixed
Oct 17	discovered that SDD busy was caused by the FLT-abort-SLT sequence
Oct 18	trigger logic changed to eliminate the FLT-abort-SLT case
Oct 25	runctrl rsh replaced by telnet
Oct 26	stopping rfiod processes
Oct 30	SDD busy identified as collector crashing after counter mismatch
	→ collector changed such to ignore this mismatch
Nov 2	end of 160 GeV lead beam

5 Solved problems

hanging-while-sending

The process sending data to CDR via socket sometimes would get stuck. It would usually suffice to kill the sending process, no need to reboot the PC. The frequency of this depended on data volume and on the sending machine: frequent for pcceres41, 42, 44, 46, less frequent for pcceres43, never happened for pcceres45, 47, 48. (Note that these machines were bought in one batch and were supposed to be identical.) During normal running it was happening every five minutes or so.

Solved by replacing pcceres41 with pcceres49, and replacing processors of pcceres42,43,44,46 with slower ones.

All these PCs (pcceres41-49) had VIA-chipset based motherboards suited for SLOT1 CPUs with 133 MHz bus (new P-III CPUs). Originally they were all equipped with P-III FCPGA/133 based CPUs with special Socket370-SLOT1 adapter cards. As the origin of the problem was unclear, we took some experiments and finally we found that using old P-II SLOT1 CPUs with 100 MHz bus cured the problem (the loss of CPU power is negligible and unimportant). We did not have a single SLOT1 CPU with 133 MHz bus, so we were not able to check whether the problem was related to the bus speed or to the CPU type (P-II versus P-III and/or SLOT1 versus FCPGA+adapter).

pcceres12 (embedded processor in CPCI) network hanging

Several times per day pcceres12 the network daemon (?) of pcceres12 was hanging so one had to restart it. This was the only machine in the setup still running linux 2.2.13 and not 2.2.16. The problem was solved by kernel upgrade. We did not, though, convert it from SUSE to RedHat.

wrong TPC data size

The non-zero suppressed data of one FEDC should have 0xa000 longwords. From time to time, for some FEDCs more often than for others, 0xa001 bytes would be reported instead. Other deviations from the expected data size were also possible but less frequent. It turned out that some of the FEDC channels were not initialized properly. After fixing this, the problem disappeared.

FICs do not want to boot

Sometimes, especially after it has been moved to another location, a FIC did not want to boot. To our knowledge there may be two reasons for this. First, many CERN network switches/routers are intelligent enough to remember where a particular machine is located - they need some time (say, 15 minutes) to forget it (i.e. clean their caches). The same problem may happen after a power cut in some crates – if the OS/9 NFS server still tries to keep the connection to the dead FIC, many switches/routers get crazy - recovering time may be very long in this case. Second, it may happen that a particular FIC won't work in the new location due to an old FIC bios version in eprom. In this case you need to get a new eprom from Catherine Moine.

SDD busy

In the trigger system each FLT should be followed either by an Abort or by a SLT. Because of an unreliable implementation of the trigger logic, the forbidden sequence FLT-Abort-SLT was generated at the 10^{-4} level, i.e. once every 5 minutes. In this case the collector was waiting for data but the event had been aborted so no data would come. There was no timeout handling on the collector level. Instead, at the end-of-burst the run control would detect a persisting SDD busy and would reset the readout. The time needed for this was such that part of the next burst would be lost.

Once we recognized the trigger as the source of the problem, the trigger implementation was changed such that the FLT-Abort-SLT sequence never happened again, and the problem disappeared.

several subsequent bursts sent to the same event builder PC

When sending was taking long, no time was left to check the status of the event builder PCs and pick the one to which the next burst should go. Then the next burst simply went once again to the same PC. This was provisorically fixed by requiring that two subsequent burst are never sent to same event builder PC – the burst is rather not sent at all. It should be solved in a more elegant way.

offline analysis

The offline analysis processes, running at the time of the experiment, accessed the data stored locally on the event builder PCs. The resulting rfio activity was introducing fluctuations in the data transfer speed. Once it was recognized, the offline analysis was moved to other machines.

tetris in rich

Partly missing RICH data, caused by changing FLT-SLT delay from 250 to 100 microseconds. According to Michael the SLT timing should make no difference. So the mechanism is not understood.

4 a.m. misery At 4 a.m. a cron job (HEPiX or something) was running on the CDR machines. The network access to the event builder PCs was slow because of this. The data transfer was taking so long that the run control would start missing bursts (see below), resulting in total confusion lasting till the network became fast again, which usually would take about one hour. The problem was recognized and understood only at the end of the run.

6 Unsolved problems

TPC unpack errors during pedestal taking

The unpack used in the pedestal taking was less fault tolerant than the unpack used in the data analysis. So the suspicion was that the problems exist already before and only become visible during pedestal taking. It is not clear what the problems are. Power cycling VME crates helps. If the origin of the problem is not understood, maybe the safest thing would be to power cycle the crates once a day or so.

run control misses end-of-burst

If the data transfer takes longer than the burst pause, and if the telnet connection to the remote machines is slow, the run control may miss the end-of-burst. Then it does not send the burst to CDR, and it does not reset the buffers. The next burst is collected on top of the burst already present in the daqmems. The buffers have double size, and the next data transfer takes even longer. If the data transfer takes longer than... etc. The end-of-burst detection logic should be improved.

network connection to pcceres7 is hanging under heavy load

Appeared in 1998, disappeared in 1999, and reappeared in 2000. Not really a DAQ problem.

I/O card interfaces draw too much current

The interfaces get power from the 12 V line of the NIM crate. The load is so high that the crate cannot be turned on when all 8 TPC interfaces are plugged in. So one has to pull two or three out, turn the crate on, and carefully push them back in. The interfaces need to be modified so they use the 6 V power line. The modification scheme and the needed elements are available.

data of one branch in one burst is corrupted

Happened several times during the run. The reason is unknown.

wrong time after MXI crash

Sometimes the MXI got stuck so thoroughly that a power cycle of the VME crate and a reboot of the readout PC were necessary. In some cases the readout PCs were coming back with a wrong clock setting. The difference was too large to be handled by xntp and thus manual setting of the clock was needed.