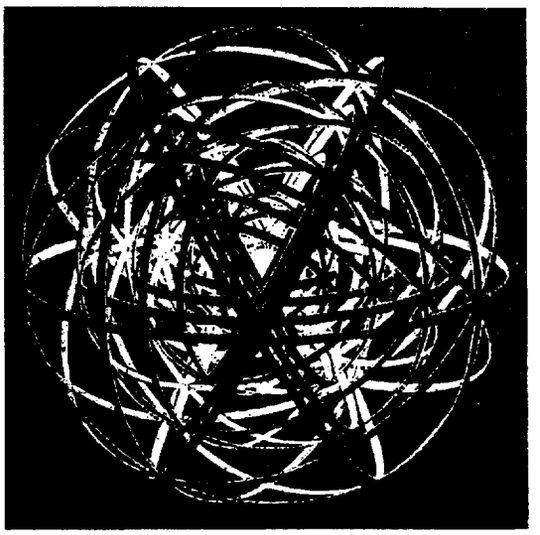
LPNHE 96-07

Laboratoire de Physique Nucléaire et de Hautes Energies

CNRS - IN2P3 - Universités Paris VI et VII

Feasibility study for re-processing H1 data on the IN2P3 computer farm



4, Place Jussieu - Tour 33 - Rez-de-Chaussée
75252 Paris Cedex 05

Tél: 33(1) 44 27 63 13 - FAX: 33(1)44 27 46 38

LPNHE I.N.2 P3 - CNRS UNIVERSITES PARIS VI ET

LPNHE-/96-07
September 13, 1996

Feasibility study for re-processing the H1 data on the IN2P3 computer farm

S. Dagoret-Campagne[1,*],E. Lebreton[1],
G. Farrache[2],Y. Fouilhe[2],W. Wojcik[2],J.Furet[2].

Abstract

This note describes 1) the architecture of the H1 re-processing program based on dice (SGI Unix multiprocessor) multitasking system, 2) the transport of this architecture on "Anastasie" (IN2P3 workstations on a network). A statistical analysis of the network doesn't show any significant drop of the performances.

1) LPNHE, Université Paris VI-VII, IN2P3-CNRS, France.
2) CCIN2P3, IN2P3-CNRS, Villeurbanne, France
*) Contact person

1 Introduction

H1 is the first experiment of a new generation which acquires a few TB of yearly amount of data, 2 order of magnitude more than the previous generation such as LEP experiments. The amount of data expected per year is about 5 TB. This corresponds to an event flow of 10 Hz, with a mean raw event size of 50 KB (after zero-suppression), for about 10^7 seconds of acquisition time, i.e. about 10^8 events [1].

With such an amount of raw data, the aim is to process or re-process them at least at the same rate as the acquisition rate if one wants to analyse them with an acceptable latency after data taking.

The estimated amount of acquired events for this year is about 70 millions of events (maximum 100 millions of events). With a re-processing time of 2 seconds per event [2](this number must be confirmed), if one fix to 3 months the time during which the full re-processing should be done, we end up with a computer which must deliver 18 seconds of CPU time per second of real time (maximum of 26 seconds of CPU time per second of real time). Thus a computer with about 20 processors is needed[1)].

The aim of this note is to see if the Anastasie processor farm can fulfil the requirements for doing such data re-processing.

2 The Anastasie farm

Anastasie is a heterogeneous cluster of UNIX based workstations located at the computer centre (CCIN2P3) of the IN2P3 at Lyon. Those processors are of 2 types, the HP series and RS series. This is an open system in which processors can be added if more computing power is required. This farm is shown on figure 1. At this time, there are 14 HP9000-735(PA RISC 7100 at 125 MHz) and 6 RS6000-390(Power PC 603 at 99 MHz). Those processors are mounted on the network by ATM switches. A performance of 10 Mbytes per second in data transfer has been reached.

The computing power is shared among several physics experiments and users. The CPU time sharing is done by the BQS batch system[3]. The imposed condition to perform the re-processing is the use of this batch system. This condition exclude any solution based on the PVM software. This condition is different for the H1 SGI (dicel) at DESY because the computer dedicated to only one user, the H1 data re-processing, and the use of a batch system is not mandatory.

3 Description of the structure of the re-processing program

The H1 processing or re-processing program has been totally written in C language by Z.Szkutnik in 92-93, in cooperation with R. Gerhards and has only slightly been modified since.

The reconstruction program *H1REC* has been written by H1 physicists in FORTRAN language during many years and is still nowadays updated frequently.

The structure of the SGI program is shown on figure 2. It comprises a master (called *onlrec*) which fork two data servers, the *receiver* and the *sender*. In addition it forks the reconstruction processes *H1REC*.

The *receiver* allows the distribution of the events to be processed in a sequential order to the *H1REC*. It reads events from an input file and fills input buffers in shared

1) This assumes 100% efficiency in CPU-time to real-time conversion and a negligible overhead. But presently the efficiency on Anastasie is only about 50 %

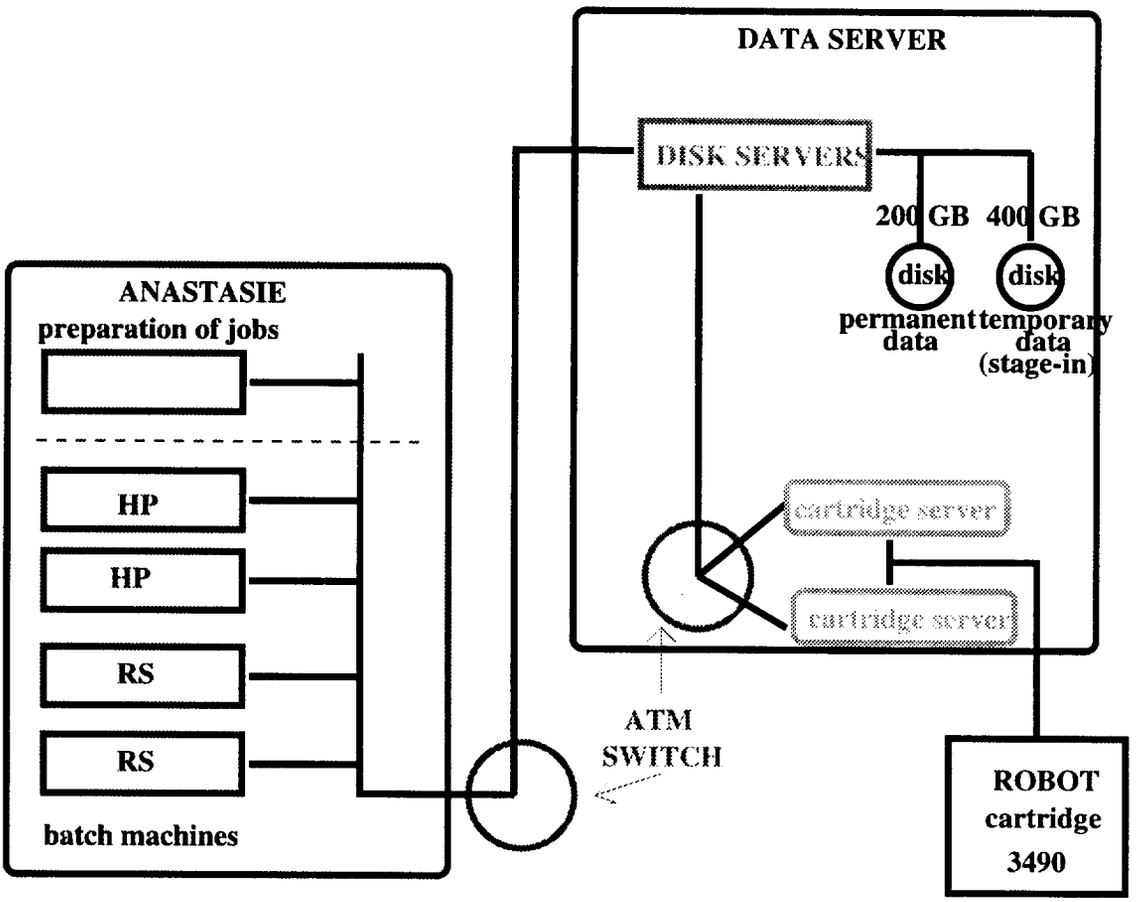


Figure 1: Structure of the CCIN2P3 processor farm Anastasie.

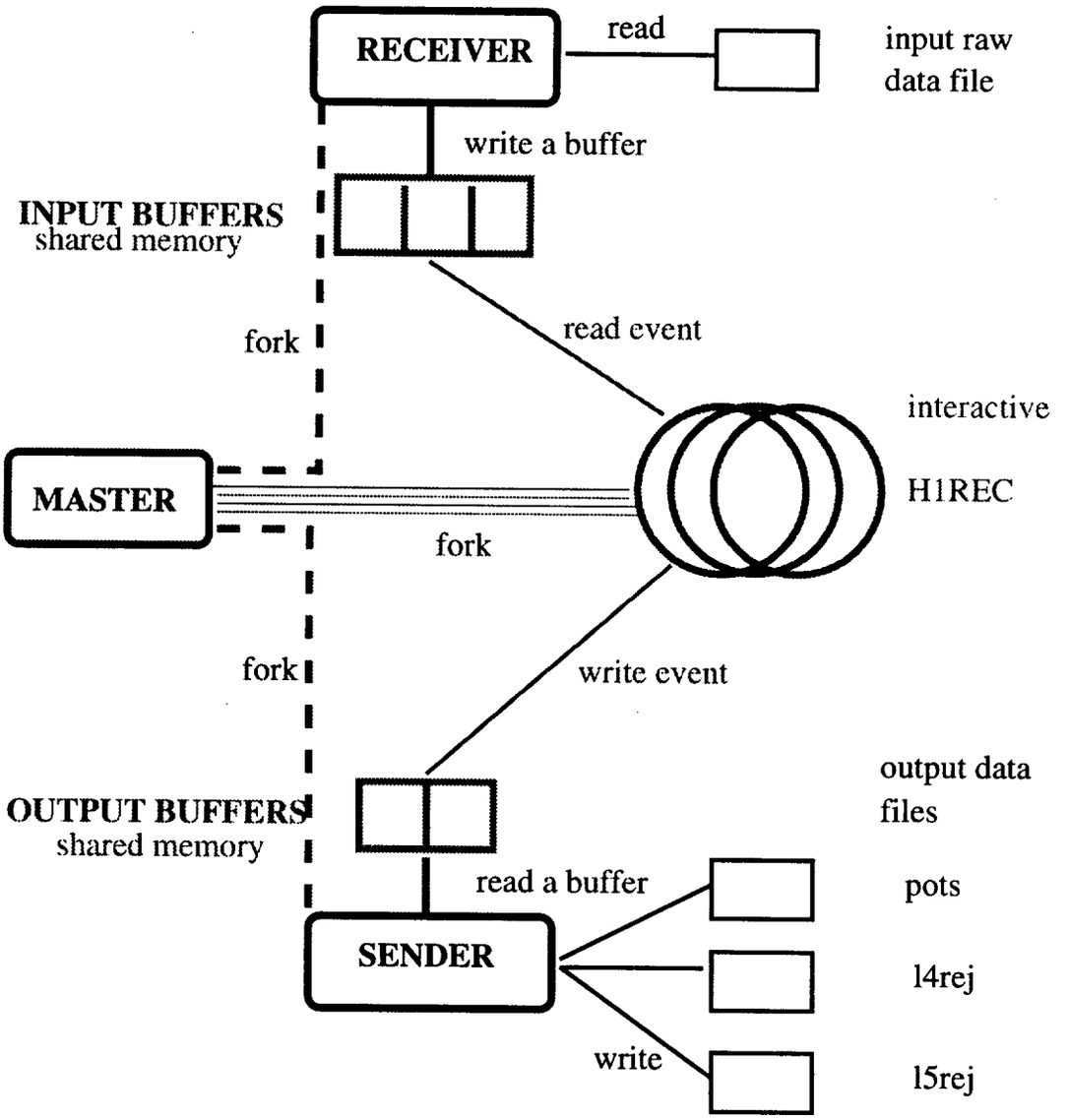


Figure 2: Scheme of the structure of the H1 re-processing program.

memory. The $H1REC$ which want to process a new event gets the first not already processed event from the input buffer.

The *sender* allows the collection of processed events. It gets the events from output buffers in which the $H1REC$ have written in the order of processing completion. Thus it collects events which are out of order due to the different times of reprocessing, which might be considerable depending on the type of event. The *sender* writes events in one of the 3 output files according to their class (pot events, L4 rejected events or L5 rejected events) and starts dumping processes whenever necessary.

Shared memory is also used to share information between the master, the *sender* and the *receiver*.

On SGI, $H1REC$ and *receiver* and *sender* run on the same machine. On the contrary, on Anastasie batch $H1REC$ jobs on the one hand and the *receiver*, the *sender* and the master on the other hand, would run on different machines. On Anastasie, the master, the *receiver* and the *sender* would run interactively on the same workstation. Thus transmission of events between them must be done through the network. The figure 3 shows that it is necessary to add 2 servers, the $INserver$ and the $OUTserver$, in order to distribute and collect events through the network into $H1REC$. But information in common between master, *receiver* and *sender* is still available in the shared memory.

In case of a crash, the master has to be restarted manually after evaluation of the problem by a physicist. On Anastasie "machine de service" could be installed to allow the interruption of the reprocessing procedure and then restart it automatically if machine-handling by CCIN2P3 staff is needed.

4 The event servers

The event servers, the $INserver$ and the $OUTserver$, set up a two-way connection oriented communication[4] between the server and the $H1REC$ client. This communication is based on the TCP/IP protocol and the Unix socket mechanism to transfer events over the network of Anastasie.

5 Access to events in shared memory

Shared memory is a common facility of UNIX-like operating systems. It allows to share some data between independent processes (having of course a different identifier pid but running on the same machine). In the H1 processing system, arbitration of the access is done via semaphores to guarantee that an indivisible instruction of the type "Test and Set" (two operations in a single instruction masking any interruption from the system) can be performed. In our case it prevents from to giving twice the same event to 2 different $H1REC$ processes or to write 2 processed events at the same place in the output buffer (one write erasing the other).

5.1 Access to input buffer

There are 3 input buffers, each having a size of 3 MB. The number of input buffers and their size has been chosen after optimisation studies on SGI.

5.1.1 On SGI:

The access to input buffers on SGI is shown on figure 4. The *receiver* writes a full buffer and then $H1RECs$ unstack events one after the other. The *receiver* writes buffers in sequential order. It stops the buffer filling when it encounters a buffer which has not

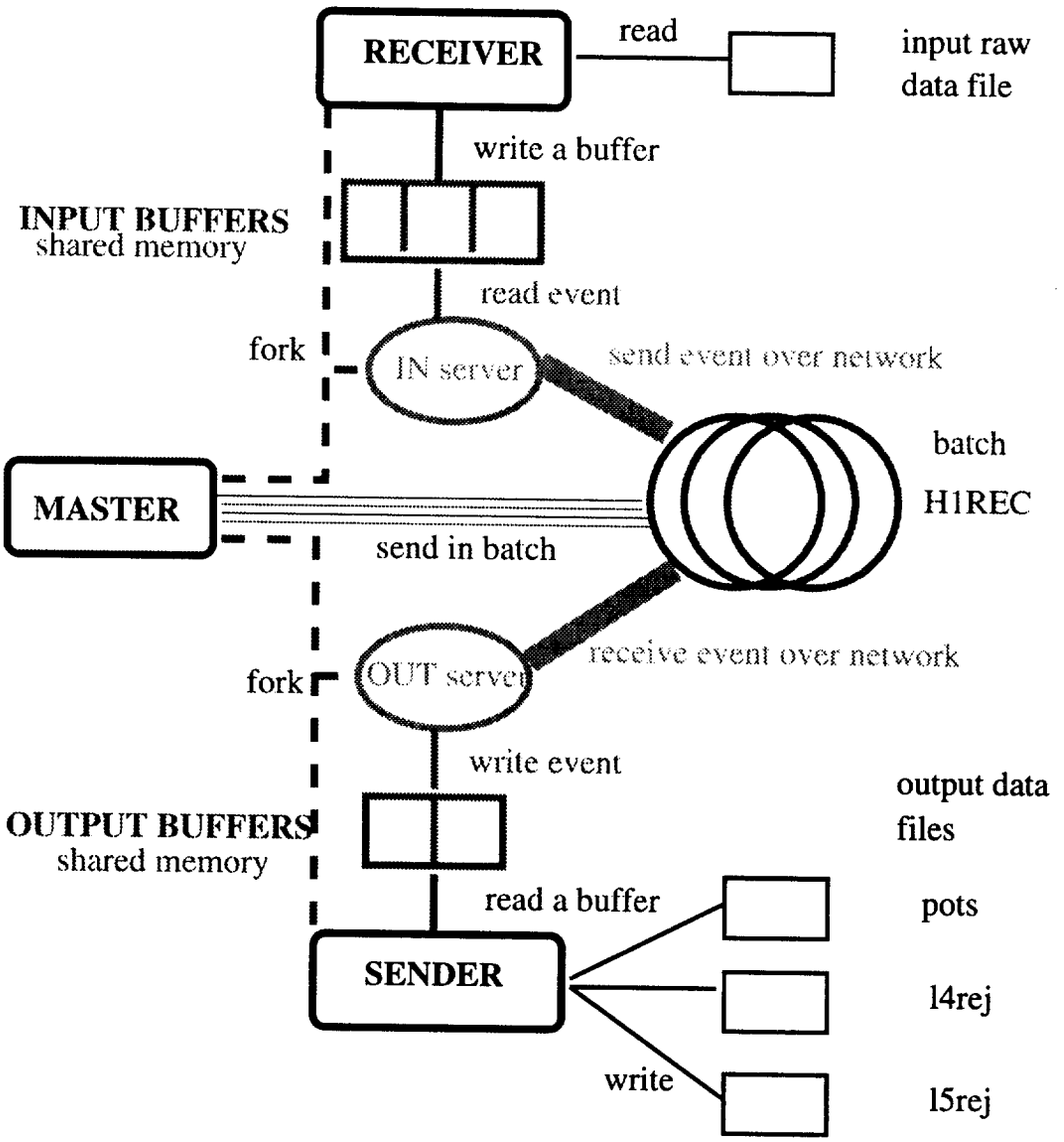


Figure 3: Adaptation of the H1 re-processing program on Anastasie farm.

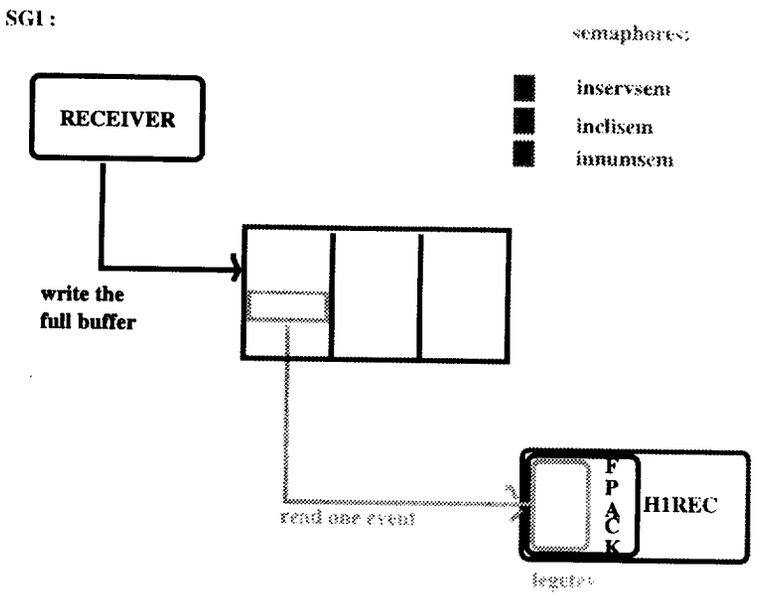


Figure 4: Access to input buffers on SGI.

been totally read by $H1REC$s. The $H1REC$ reads an event in the input buffer through specialized FPACK input routines. The lowest level routine *lgetev* reads physically the event from the input buffer.

Arbitration of the access is done via 3 semaphores :

- the *inservsem* semaphore which is scrutinised by the *receiver* in order to access to one input buffer (the value of *inservsem* is incremented by $H1REC$ whenever an input buffer has been emptied),
- the *inclisem* semaphore which is scrutinised by $H1REC$s in order to authorise globally the event reading in this buffer (the value of *inclisem* is incremented by the *receiver* after filling a buffer with raw data),
- the *innumsem* semaphore which arbitrate the access between the different $H1RECs$.

5.1.2 On Anastasie:

The access to the input buffers on Anastasie is shown on figure 5.

The reading of an event in the input buffer is now done by the $INserver$ in the same way as $H1REC$ did previously, by the routine *lgetev*. The server transmits the event on the network as a packet of a certain number of bytes. The reception of the event by $H1REC$ is done by a new low level routine *read_network_event* and is transmitted to FPACK routine as *lgetev* did previously.

The number of semaphores is reduced to 2, the *inservsem* and *inclisem*, as there is only one reader (the $INserver$) now.

5.2 Access to output buffer

There are 2 output buffers, each having a size of 3 MB. The number of output buffers and their size has been chosen after optimisation studies on SGI.

5.2.1 On SGI:

The access to output buffers is shown on the figure 6.

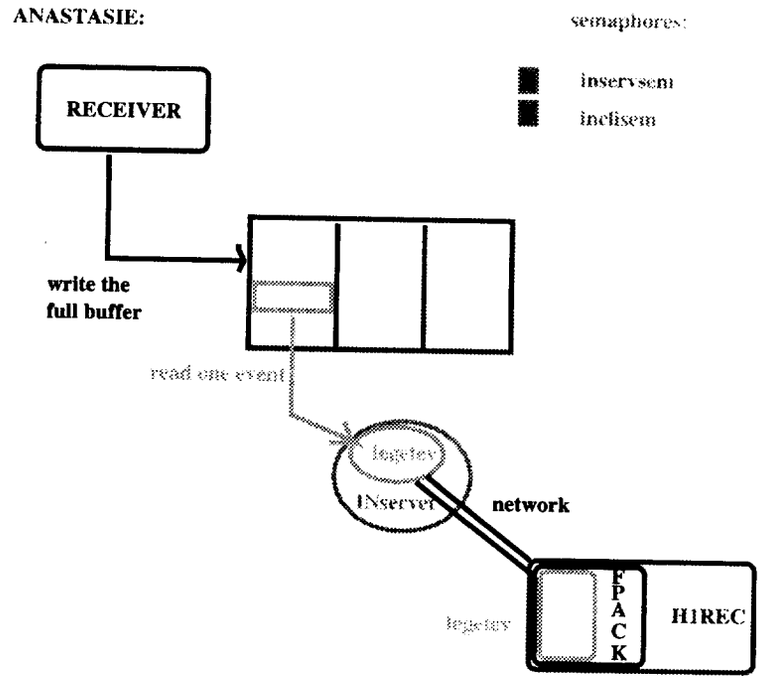


Figure 5: Access to input buffers on Anastasie.

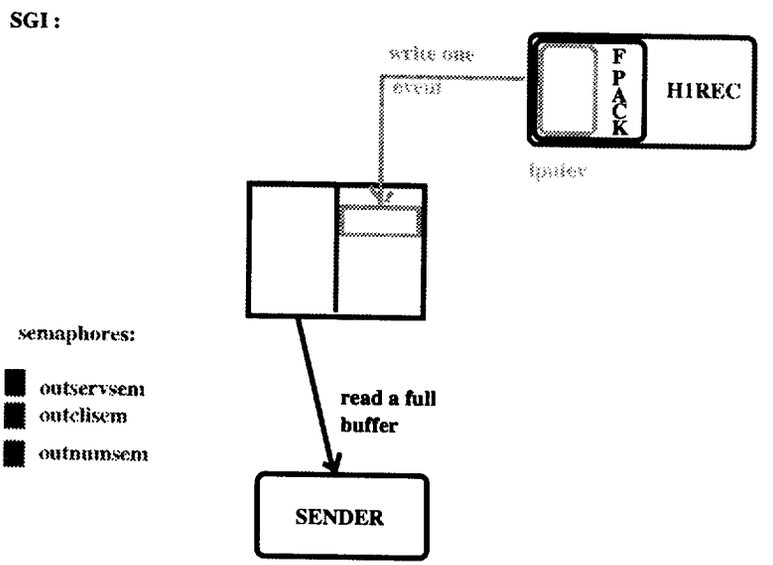


Figure 6: Access to output buffers on SGI.

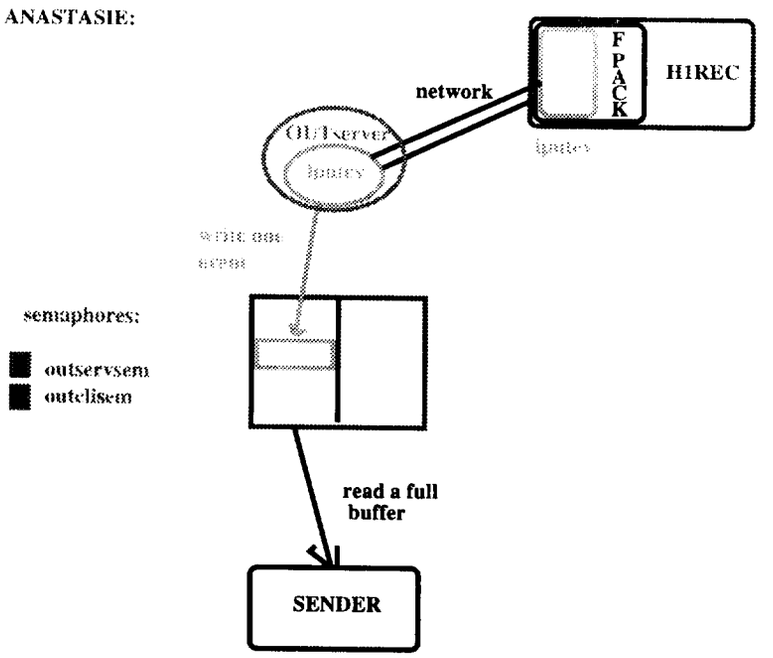


Figure 7: Access to output buffers on Anastasie.

*H1REC*s write into the output buffers, one after the other through the FPACK routine layers, using the lowest level routine *lputev*. When a buffer has been totally filled, it is then read by the *sender*.

Again, arbitration of the access to buffers is done via 3 semaphores :

- the *outservsem* semaphore which is scrutinised by the *sender* in order to access to one output buffer (the value of *outservsem* is incremented by *H1REC* upon writing the last event to the output buffer),
- the *outclisem* semaphore which is scrutinised by *H1REC*s in order to authorise globally the event writing in this buffer (the value of *outclisem* is incremented by the *sender* after having read the whole buffer),
- the *outnumsem* semaphore which arbitrate the access between the different *H1REC*s.

5.2.2 On Anastasie:

The access to output buffers is shown on the figure 7.

The *sender* read a full buffer as it did on SGI. The change intervenes when *H1REC* has a processed event to send back. It sends it through the network to the *OUTserver* as a packet of the corresponding number of bytes by the new routine *write_network_event*. When the *OUTserver* receives the event, it writes it into the output buffer in the same manner as *H1REC* did it previously, using the routine *lputev*.

The number of semaphores is reduced to 2, the *outservsem* and *outclisem*, as there is only one writer (the *OUTserver*) now.

6 Forking the processes

The master, named *onlrec*, handles the whole processing and forks all the necessary processes. The master has 2 phases, an initialisation phase and an infinite control loop.

The forked processes on SGI are shown on figure 8. During the initialisation phases, it forks the *sender* and the *receiver* using the UNIX-standard *fork* routine. During the control loop phase, it forks or stops the *H1REC* processes, adjusting the number of running

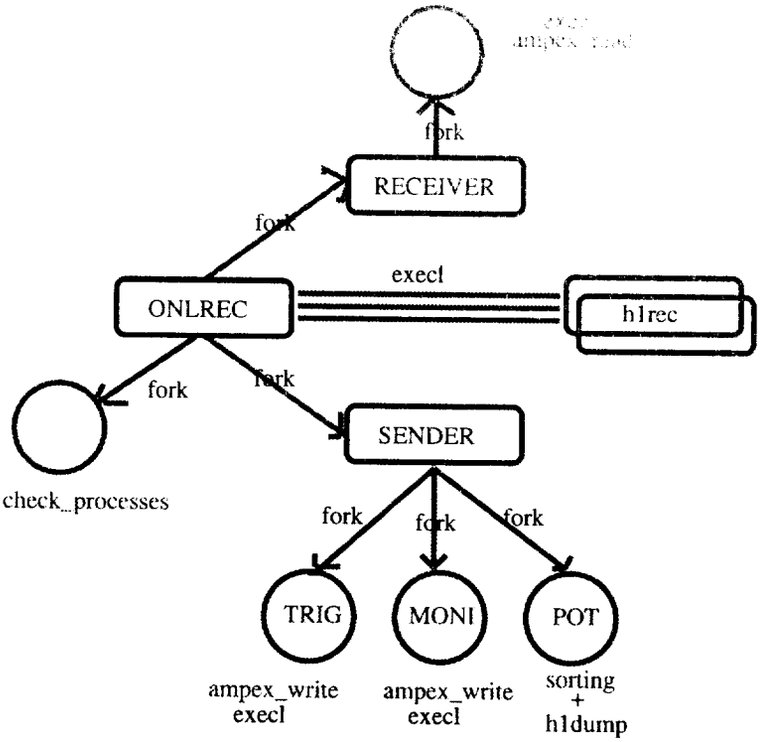


Figure 8: Forked processes on SGI.

H1RECs to the desired value, using the UNIX-standard *execl* routine. In addition it forks regularly a *check_process* routine which controls the status of the *H1REC*, using the *fork* routine. In their turn, the *receiver* and the *sender* fork processes using the *fork* command, for stage-in input file or dumping output file onto media.

There is only a small difference on Anastasie with respect to SGI shown on figure 8.

The *H1REC* are no more forked by *execl* routine but rather sent in batch by the BQS batch command *qsub*. The *INserver* and the *OUTserver* are forked in addition by the *fork* routine.

7 Signal handling

During the execution of the program on SGI, the different processes exchange several UNIX signals (these are software interruptions). Proper signal handlers are set for those processes. The exchanged UNIX signals are sketched on the figure 10.

These signals are often associated with a termination procedure.

When the master sends the signal *SIGTERM* to the *sender* or *receiver*, those processes clean their respective semaphores. When the master sends *SIGUSR1* to a *H1REC*, the later sets a stop flag. *H1REC* exits properly at the next attempt for reading next event in the input buffer. The *sender* can receive the signal *SIGUSR2* from the *receiver*. It corresponds to an order to dump immediately all the processed events (handler *force_dump*). (This order is set in the steering file which is regularly read by *onlrec*, which transmits the flag *receiver_block* to the *receiver* via shared memory.) *H1REC* can also send to the master a signal *SIGUSR2*, asking for a database update.

Almost all these signals can be used on Anastasie (see figure 11) because the master, the *receiver* and the *sender* are running on the same processor. The signal *SIGUSR2* from the master to *H1REC* can be used by the [illegible]

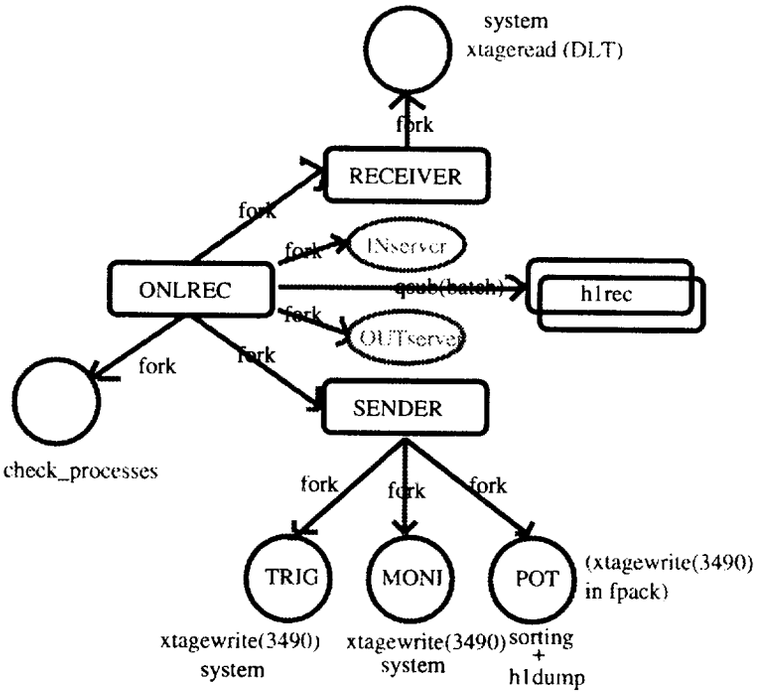


Figure 9: Forking processes on Anastasie.

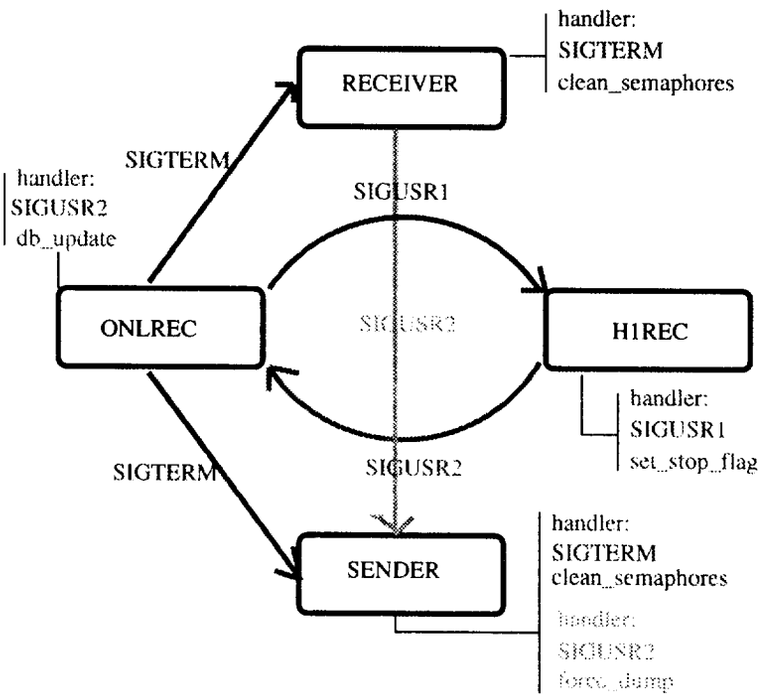


Figure 10: Signals handling on SGI.

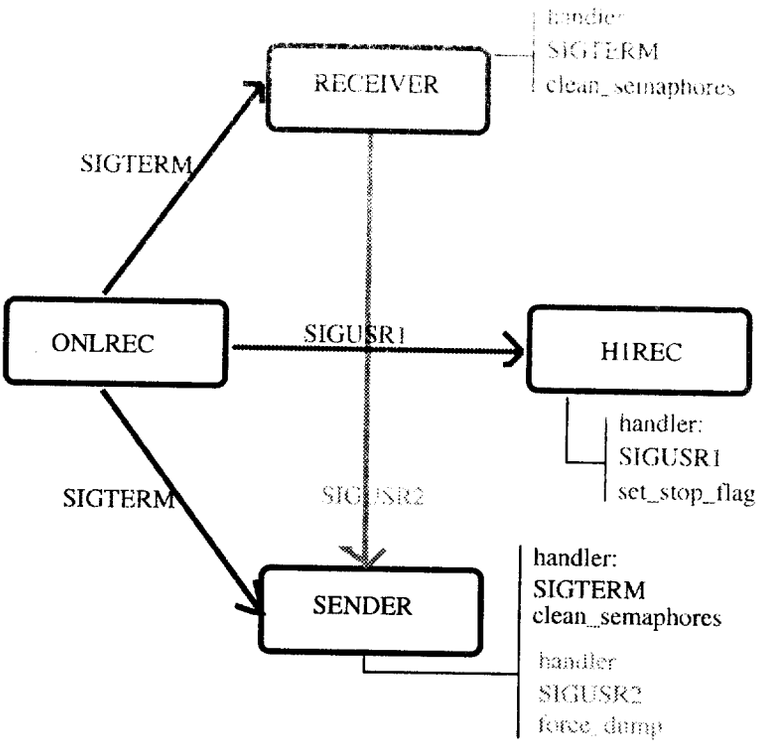


Figure 11: Signal handling on Anastasie.

SIGUSR2 issued by *H1REC* to the master, associated to the database update request cannot be used anymore. This is not critical, however, for re-processing, because the database should anyway contain proper constants in that case.

8 Getting the input file

The input file is staged-in by the receiver as shown on figure 12 for the SGI. The name of the next file is found by the routine *get_fname* which calls the routine *finfil*. In the routine *finfil*, there is a mechanism to set the current RUN-EVENT number to start in, either at the beginning of the whole reprocessing, or at the restart point after a crash. Then the correspondence between the RUN-EVENT number and the filename is found by reading a mapping file *H01NCR.STPOT93.RUN*. The filename is a H1-standard file name and its form must be kept on Anastasie in order to avoid much changes in the *receiver* code. As soon as the filename is known, the staging of the media is done in the routine *get_file*. Note that at DESY, the knowledge of the filename is sufficient to have a link to the correct number cartridge. At Lyon, this facility doesn't exist and an additional mapping file must give the correspondence between the filename and the DLT-VID number. This mapping file must be provided by H1-DESY when they will send the DLT to the CCIN2P3.

9 Dumping the output files

After having read one buffer, the *sender* stores the events according to their FPACK bits set during the reconstruction in 3 output files of 200 MB each (the MONFILE for L4 rejected events, the TRIGFILE for L5 rejected events and the POTFILE for good

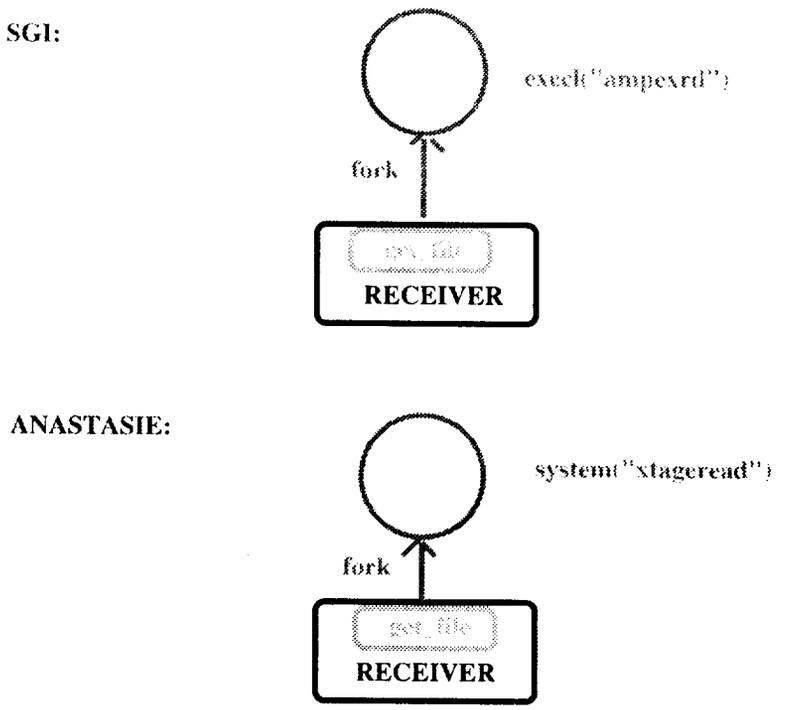


Figure 12: Getting the input file on SGI and on Anastasie.

physical events also called the pot data). The figure 13 shows that both implementations on SGI and Anastasie are very similar. Concerning the POTFILE, the events must be sorted (using an intermediate 65 MB SORTFILE) before being dumped on a cartridge. This is done via a set of routines *fsort h1dump* and *rdump*. After a successful output dump, a mapping-file is written to map the output pot filename with the RUN-EVENT reconstructed event number range : the *potsum* routine write the $HERA05.H1POT95$ mapping-file. An additional mapping-file is needed on Anastasie to map the pot filename with the output cartridge VID and file sequence number. Note that this last mapping file is essential when the reprocessed data are to be stored at DESY.

10 Information saved on files

10.1 The control file

During the execution, some information is saved in the file "H01NCR.CONTROL.NCR" in the form of a table of 50 integers "*lstat*". This information is regularly updated in the main control loop of the master. The most important parameters are the following :

lstat(3): current number of $H1REC$ processes,

lstat(5),lstat(6): current run number and event number in the *sender*,

lstat(9): 1: start from scratch 0:restart after a crash, used in the routine *finfil* (this works only together with lsteer(3)),

lstat(38),lstat(39): last event dumped on tapes, saved by *rdump*, used as the restart point after a crash.

10.1.1 The steering file

The user can write some steering choice during the execution in the file "H01NCR.STEERING" using an interface program with name "ctrread". The user can

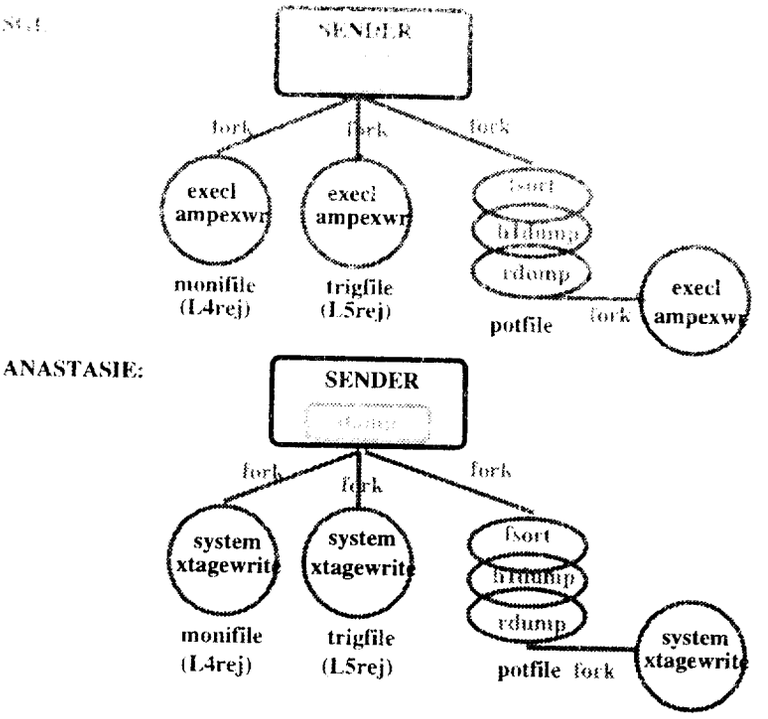


Figure 13: Dump of the output files on SGI and Anastasie.

change interactively the number of "*H1REC*" processes and ask for stopping the reprocessing. The steering information is under the form of a table of 5 integers "*lsteer*". The most important parameters are the following :
lsteer(2): the requested number of *H1REC* processes by the operator,
lsteer(3): run number to stop at,
lsteer(4): run number to start at,
lsteer(1): stop the main control loop of the master,
lsteer(5): force an urgent dump when requested (a signal is send from receiver to sender).

11 The master

The master program named "*onlrec*" consist of a phase of initialisation and a phase of an infinite control loop.

11.1 The initialisation phase

During the initialisation it performs the following tasks :
1. creation and attachment of the shared memory segment,
2. reading of an input steering file and a control file,
3. forking the *receiver* and the *sender*.

This phase can be re-copied from the standard H1 code without major changes. On Anastasie, the *INserver* and the *OUTserver* are forked in addition.

11.2 The infinite control loop

The command in the main control loop of the second phase are shown on figure 14. This loop has a temporisation of 1 minute. It loops over the following tasks :
1. writing the control file,
2. reading the input steering file.

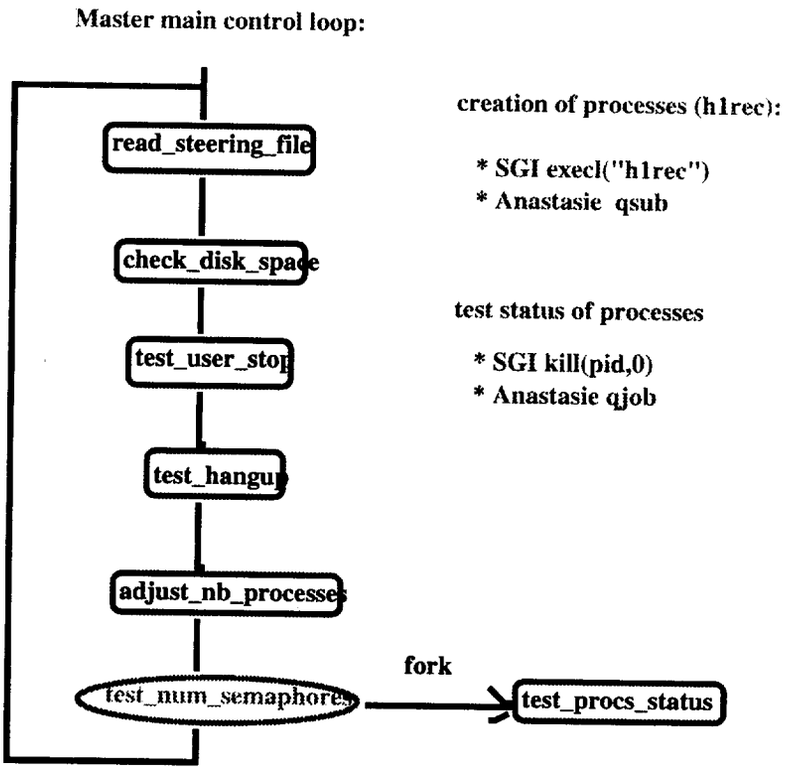


Figure 14: Task performed in the main control loop.

3. checking the remaining disk space,
4. checking for user-stop,
5. **adjustment of the number of batch $H1REC$ processes,**
6. **checking the status of batch jobs (and of servers),**
7. **handling of the merging of control histograms.**

A delicate point for the adaptation of the master on Anastasie are the adjustment of the process number and the checking the status of the process because batch commands must be used.

On SGI, the information concerning $H1REC$ processes is kept in tables of C-structures. One table contains the pids of the processes $H1REC$s, the other table contains the status of those processes. These tables are located in the shared memory used for storing information. A simple adaptation for Anastasie would consist in adding a parallel table of C-structures. The C-structure would contain all the information about the batch jobs :

- the machine name which runs the process,
- the (batch) job name given by the batch supervisor,
- the pid on the batch worker (machine),
- the date and time at which the job started,
- the status of the job : queued, running, ended, killed...
- the CPU time used.

In addition some primitive routines must be defined in order to replace them in the standard-H1 code.

These primitive routines would do :

- sending job in batch (*new_processes*),

- testing batch job status,
- killing batch jobs.

Another important point is the histogram production for a data quality control. The histograms are generated by the reconstruction processes. Theses files have to be merged afterward by a dedicated process (*H1PRINT*) which is forked by the master from the control loop. This procedure occurs when the master detect that all processes *H1REC* have ended the processing of a run. Conditions of the implementation of histogram production on Anastasie are described in section 12.

11.2.1 Adjustment of the number of *H1REC* tasks and the checking of their status

In the 2 implementations, SGI and Anastasie, the logic of creation/deletion and checking-status of the *H1REC* processes or batch jobs can be kept. Obviously the technical tools, in order to execute an *H1REC* process on SGI (*execl* C routine) or to submit an *H1REC* job *qsub* BQS command in the Anastasie differ. In Anastasie implementation, a C-interface to BQS has been written for this reprocessing project. On SGI, the deletion of *H1REC* is done by sending Unix signals (SIGUSR1 or SIGTERM). This can also be used on Anastasie by sending signals with the *qsig* BQS command. In order to check the status of the *H1REC*, a checking-process is forked from the main control loop. While the state (RUNNING or DEAD) of a process is obtained by sending the NULL signal on SGI, the batch-status of a batch job is obtained by issuing the *qjob* BQS command on Anastasie. The *H1REC* batch job is considered to be RUNNING if its batch-status is either QUEUED or RUNNING and it is considered to be DEAD if its batch-status is ENDED, KILLED or DELETED.

11.3 Automatic restart from a crash

This is done automatically when the master is restarted (by a "machine de service"). **This "machine de service" must be delivered by CCIN2P3.** Before restart, the machine must check that all previous processes are dead or kill them. The restart point is determined from the information saved in the control file.

12 The reconstruction program H1REC

In SGI, the standard *H1REC* (FORTRAN written) reconstruction program is encapsulated in a C-written program *ch1rec*. *ch1rec* set signal handlers for all possible errors which could occur during the execution of the standard *H1REC*. Then shared memory is attached and semaphores are opened before the execution of the standard *H1REC*. Shared memory is accessed in the following conditions :

1. reading a raw event from the input buffers,
2. writing a reconstructed event to output buffers,
3. database update at the beginning of each run,
4. to announce printed histograms to the master at the end of each run,
5. to erase itself from the table of processes when the master has send a SIGUSR1 signal.

In Anastasie, the access to shared memory and semaphore is not possible. Conditions 1 and 2 are solved by the use of data servers. Condition 3 doesn't occur because the full database is available on disk.

During the reprocessing, at the end of each run, the reconstruction processes write an histogram files with the name *LOOKNCRun* (*nn* is the number of the *H1REC* process

attributed by the master). Those histogram files must be accessed by the master which has to perform the merging task. On Anastasie, each batch job should write the histogram files at a common place on a disk accessible via AFS. Condition 4 and 5 can be solved by doing remote shell execution (Unix command *remsh*) : the execution of a small program is asked to be run on the machine of the master. This small program attaches the shared memory and write some integers (given through script arguments) at the right place in shared memory.

13 Results and present status at August 27, 1996

A version of the reprocessing close to the final version has been made running with a master forking the processes *receiver*, *sender*, *INserver*, *OUTserver*. Jobs with a pseudo-*H1REC* (skeleton but without the reconstruction code) were send in batch and controlled by the master. Automatic stage-in and stage-out was not yet implemented in the *receiver* and the *sender*.

The most advanced structure tested is the following : a version of a simplified *receiver* read a file containing 95 raw data. This *receiver* fills the input buffers looping infinitely on this file. The *INserver* unstacks the raw events one after the other from the input buffers when receiving a client request of a job. The event is send over the network to the pseudo-*H1REC*. The pseudo-*H1REC* send the events to the *OUTserver*. Then, the *OUTserver* fills the output buffers which are read by the *sender*. The *sender* writes events in the pot output file.

For this structure, all the processes were running on the same HP processor cc-shp001 except the pseudo-*H1REC* which were running on batch machine.

The performance of the *INserver* are shown on figure 15. The time of transmission is roughly correlated to the event size by the formula :

$$\delta t(ms) = 0.16 size(kB) + 11$$

but a tail at large δt raises the mean time of transfer to 30 ms (with a RMS of 30 ms). These results are satisfactory for the reprocessing purpose.

14 User interface

A user interface is available in order to allow the user to choose a run range and to start "by hand" the re-processing or to ask for urgent stopping of the re-processing. A user friendly interface with multi-windows giving the status of the re-processing exist on dice1 and could be installed on Anastasie.

15 Conclusion

This study proved that all the logic and the H1 re-processing program running on the SGI, can be implemented on Anastasie without major difficulties. Some development was necessary because Anastasie is a processor farm and not a multi processor computer. This was done by writing 2 servers which have to distribute raw data events to batch reconstruction jobs or have to collect the reconstructed events from these batch jobs. Preliminary results on the *INserver* have shown that the transmission time of events on network is small compared to the reconstruction time.

If H1 takes the decision to re-process data on Anastasie, the development of service routines in order to manage the continuous flow of operation will be needed.

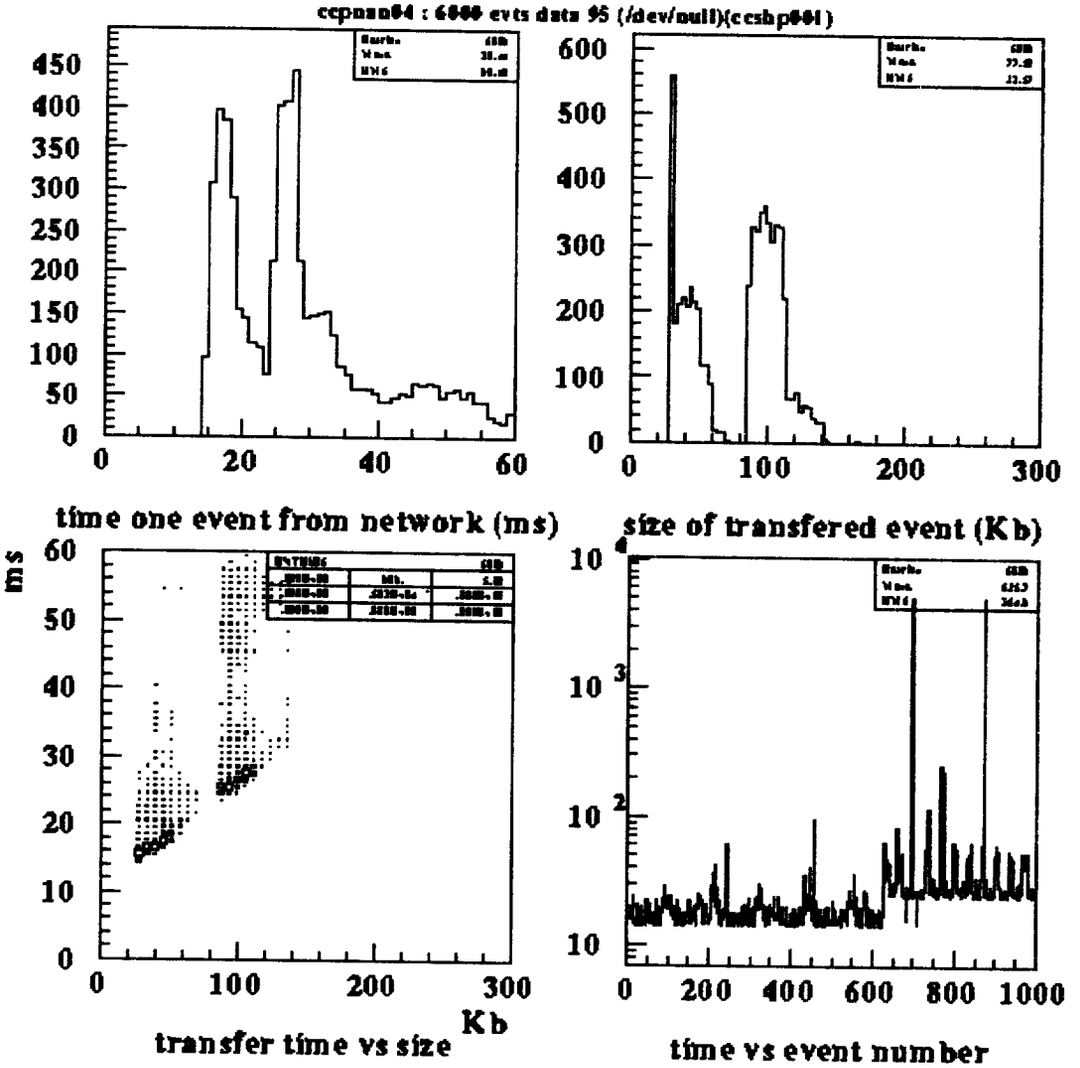


Figure 15: The distribution of the time of transfer (upper left), the event size distribution in kB (upper right), the time of transfer versus the size of event (lower left), time of transfer versus the event number for the first 1000 event (lower right)(the step in the time of transfer is due the the change in the event size).

Acknowledgements

The H1-IN2P3 group expresses its gratitude to J. Ganouna, director of the CCIN2P3 who strongly supported this re-processing project as one of the main developments of his computing centre and E. Auger, who defined our critical framework and gave us courage to start.

REFERENCES

[1] Data logging and online reconstruction in H1
P. Fuhrmann, R. Gerhards, U. Kruner-Marquis, J-E. Olsson, Z. Szkutnik

[2] Reprocessing des données à Lyon
U.Berthon,V.Boudry,M.Jaffre,E.Lebreton,JP.Pharabod
13 juillet 95

[3] Utilisation des Ressources Unix à l'IN2P3
CCIN2P3
6 octobre 95
Under development

[4] Practical Unix Programming
K.A. Robbins, S. Robbins
Prentice Hall PTR 1996