MONARC-99/4
25 May 1999

Preliminary Objectivity tests for MONARC project on a local federated database

Matteo Boschini (CILEA and INFN - Milano)

Laura Perini (Milano University and INFN - Milano)

Francesco Prelz (INFN - Milano)

Silvia Resconi (INFN - Milano and CILEA)

1 Introduction

This note documents some preliminary local tests with Objectivity for MONARC testbed. The work is still in progress and futher developements and refinements of the work are foreseen. All the details of the performed tests are explained in chapters 2.1 and 2.2 and ideas for further tests are summarized in chapter 3.
To perform these tests the Atlfast++ program in LHC++ enviroment has been used. This package has been developed with the aim of porting Atlfast++ algorithms [1] into the LHC++ enviroment and to provide the MONARC project with a useful way to populate the Objectivity database with fast simulated events. At the moment the program allows:

- to generate and reconstruct events, produce a hbook file (with Histomanager, hbook wrapper by A. Dell'Acqua) and populate the database following the Tag/Event data model proposed by the LHC++ project.
- to read data from the database with a simple program, and to fill some histograms with HistoManager or with the HTL package.

The Atlfast++ kit and all the instructions to install it [2] on Sun platform (see the README file) can be found at the following address:

$/afs/cern.ch/user/r/resconi/public/pythia\_monarc$

The Atlfast++ database architecture follows the Tag/Event data model, at the moment it is very simple: one single container for the Event with no Objectivity associations. Here is a list of the LHC++ tools needed to run Atlfast++ on SUN:

- Objectivity 5.1
- HepODBMS
- Objectspace 2.0
- CLHEP
- HTL

Once Atlfast++ and all the libraries are installed on Sun, two test jobs can be run: one to populate Objectivity/DB and the other one to read Objectivity/DB and fill some histograms.

2 Preliminary tests on a local federated database

The first tests have been started on a local federation with the following configuration:

- System = SUN Ultra 5 (333 MHz)
- Memory size = 128 Megabytes
- Disk Controller = UltraSCSI-3
- OS = Solaris 2.7
- Compiler = Sun C++ 4.2
- Objectivity page size = 8192 Bytes

The local federated database has been filled with 100.000 events generated with Atlfast++ corresponding to a total occupancy of $\sim$ 4 Gigabytes.
With the above configuration the time to generate, reconstruct and fill the database is $\sim$ 0.11 seconds/event and the size of one event $\sim$ 40 Kilobytes (one event corresponds to $\sim$ 5 Objectivity pages).
Due to the fact that the maximum size for one database is 2 Gigabytes (for a 32 bit system), Atlfast++ opens a new Events.DB every 40000 events; the size of an Event database is $\sim$ 1.7 Gigabytes for 40000 events and the size of the Tag database for all the statistics is $\sim$ 8.4 Megabytes.
The following data retrieval tests have been started on the local federation:

- Stress tests
- Timing tests

2.1 Stress tests on the local federation

The procedure followed to perform these tests consists in submitting an increasing number of concurrent jobs and then monitoring the loss of performance in terms of execution time (Real Elapsed Time).
These tests have been performed in a single user configuration on the Sun machine, and the load of the CPU has been monitored during the tests.
The concurrent jobs read the same number of events from Objectivity/DB, accessing the same containers, and perform very few calculations.
The same test has been repeated for different samples of jobs:

- jobs reading only Tag attributes of all 100.000 events, corresponding to $\sim$ 10 MegaBytes.
- jobs reading Tag+Event attributes for the following numbers of events: 1209, 2967, 6645, corresponding to $\sim$ 50, 120, 270 MegaBytes.

The distribution of the execution time versus the number of concurrent jobs for the 4 cases is shown in fig. 1. It follows a straight line untill a divergent behaviour starts for a number of concurrent jobs reading Tag+Event attributes around 30, and moreover the not linear behaviour is reached slightly earlier by jobs reading more events.
The more than linear dependence is explained with the exhaustion of available physical memory and the need to access swap space. The acces time to swap space is proportional to the number of jobs that are concurrently active. The function used to fit the curves in fig. 1 is the following:

$$y = ax + \frac{b(x-c)^2}{1+e^{(x-c)}}$$

where:
c = number of concurrent jobs causing saturation phase
$x < c \rightarrow y \sim ax$, straight line
$x > c \rightarrow y \sim bx^2$, parabola (saturation phase)

The same test has been repeated decreasing the size of the Objectivity cache from the default one (250 pages) to 100 pages. Fig. 2 shows the superimposed distributions of execution time versus the number of concurrent jobs for the two cases. It is interesting to notice that for an Objectivity cache size of 100 pages the divergences disappear and the limitation on the number of concurrent jobs is given by system resources limits if more than $\sim$ 60 jobs are submitted concurrently. This corresponds to about twice the number of concurrent jobs it takes to run out of physical memory in the case of a 250 pages Objectivity cache. This tests are going to be repeated on an other Sun machine with similar configuration [3] to see if comparable results can be obtained.

As a cross-check the same test has been repeated with a simple C++ program that reads 60 MegaBytes of binary data in 8 KiloBytes records. The main goal was to measure the maximum number of concurrent jobs before the machine starts to swap and the performance degrades.
The test has been done for two different job sizes: 9.5 and 17.7 MegaBytes. Sizes are changed by simply allocating an increasing memory stack (using `malloc`).
These values of memory sizes are comparable with those of the jobs run in the previous tests with Objectivity. As shown in fig.3, the execution time is essentially independent from the job's size and at least until $\sim$ 40 concurrent jobs there is no evidence of swapping.
The behaviour of the execution time versus the number of concurrent jobs (fig.3) is in agreement with that obtained with an Objectivity cache size of 100 pages which maximizes the number of permitted concurrent jobs.

2.2 Timing tests on the local federation

The procedure followed to perform these tests consists in measuring the behaviour of CPU and I/O time versus the number of MegaBytes read by the jobs in the following two cases:

- Sequential read:
  jobs that read sequentially Generic Tag and Event attributes have been used.
- Simple not sequential read:
  the jobs read a collection of events, passing a certain criteria (i.e. a kinematic cut), from a Generic Tag. In this case the jobs access attributes of events that are spread in the database, crossing the association between Generic Tag and Event. Lowering the kinematic cut it is possible to obtain collections made of an increasing number of events that are read by the jobs.

In the following are reported few basic definitions that will be used during the explanation of the tests:

- $t_{REAL} = t_{CPU} + t_{I/O}$
- $t_{CPU} = t_{CPU}(for\ I/O) + t_{CPU}(NOT\ for\ I/O)$

t_{REAL} and t_{CPU} are measured directly by the jobs.
The jobs read in both cases the same number of events from Objectivity/DB and perform very few calculations, actually it has been checked that the $t_{CPU}(NOT\ for\ I/O)$

can be considered negligible with respect to the total t_{CPU}. The only I/O operations performed by the jobs consist in reading data with Objectivity.
The tests have been performed as single user on the Sun machine with a single job running.
The jobs read Tag plus Event attributes for an increasing number of events and the corresponding number of MegaBytes read has been carefully measured with the Solaris system command `truss(1)` to trace system calls and signals; the number of events read covers the total statistics and goes from $\sim$ 6000 to 100.000, the corresponding number of MegaBytes read ranges from $\sim$ 50 to 4000.
The distribution of CPU time versus number of MegaBytes read for the 2 cases considered is shown in fig.4. It is possible to notice that the behaviour of the CPU time versus the number of MegaBytes read for jobs reading an increasing number of events is the same for the two cases considered and follows a straight line:

$t_{CPU} \sim 0.23$ * (number of MegaBytes read) sec.

The distribution of I/O time versus number of MegaBytes read for the two cases considered is shown in fig.5 where it's possible to notice that until a number of events read corresponding to about half of the total statistics, the two cases are represented by two parallel lines with an average temporal shift of about $\Delta t \sim 80$ seconds. It' s possible to notice that the inverse of the slope is equal to $\sim$ 4 MegaBytes/seconds.
Due to the fact that the not sequential read case tends to the sequential one increasing the number of events read, the two distributions converge for a number of events read approaching the total statistics.
Finally an evaluation of Objectivity Real Elapsed Time performances in reading compared to simple C++ performances has been done.
Fig.6 shows the behaviour of Real Elapsed Time versus number of MegaBytes read for the two cases considered. We can assume that for a high number of MegaBytes read :

$t_{REAL} \sim 0.5$ * (number of MegaBytes read)

Defining:

$$\text{speed} = \frac{number\ of\ MegaBytes\ read}{t_{REAL}}$$

it is possible to obtain with the present configuration and data clustering an Objectivity speed of $\sim$ 2 MegaBytes/seconds.
For what concerns C++ timing performances, the time that a simple C++ program takes to read a binary file in 'records' of 8 KiloBytes with the same configuration has been measured, obtaining :

$t_{REAL} \sim 0.11$ * (number of MegaBytes read)

If a C++ speed is defined as before, we obtain $\sim$ 9 MegaBytes/seconds. With the present configuration and data clustering we can roughly estimate the Objectivity overhead on Real Elapsed Time with negligible $t_{CPU}(NOT\ for\ I/O)$ as a factor 4-5 worse with respect to simple C++.

3 Conclusions

Some first results on Objectivity testbeds for the Monarc project were presented; these results should be considered as a starting point for further measurements of Objectivity performance.

In particular, a stress test showed that one can achieve a high number of concurrent Objectivity jobs with nearly no performances degradation, by tuning the Objectivity cache size. With the current configuration, increasing this parameter from 100 pages ($\sim$ 800 KiloBytes) to 250 pages ($\sim$ 2 MegaBytes), induces an exponential increase of the Real Elapsed Time needed to execute more than $\sim$ 30 concurrent jobs.

Furthermore, CPU and I/O times have been measured for two different data access modes (sequential, not-sequential): the CPU time per MB read is the same for the two cases. I/O time per MB read is represented, until half of the statistics read, by two lines with the same slope and constant temporal shift. The not sequential case converges to the sequential one approaching the reading of the total statistics.

Finally, for the current configuration and data model, Objectivity's I/O seems to be 4-5 times slower than a simple C++ application reading the same amount of data.
The previous results show the main role played by data modeling and clustering, on which further studies are already under developement.
Some more ideas for further tests are:

- understand Objectivity performance in terms of CPU and I/O time with different configurations and systems. It is worthwhile to repeat these tests for example with CERN configuration and on LINUX.
- understand other dependencies from Objectivity parameters like i.e. page size, cache size, object size.
- test Objectivity performances on a 'realistic' analysis scenario, with more users accesing the same event sample and performing their own analysis on personal collections of events.
- understand the reasons of the relatively low rate

  in Objectivity reading , in order to try to get better performances
- all the tests should be done in three main steps:
  - on a local federation
  - on 2 local machines using AMS
  - on a multiple site federated database

References

[1] http://atlasinfo.cern.ch/Atlas/GROUPS/PHYSICS/HIGGS/Atlfast.html

[2] Thanks to the help of Alessandro Brunengo (INFN, Genova)

[3] Thanks to Massimo Sgaravatto (INFN, Padova)

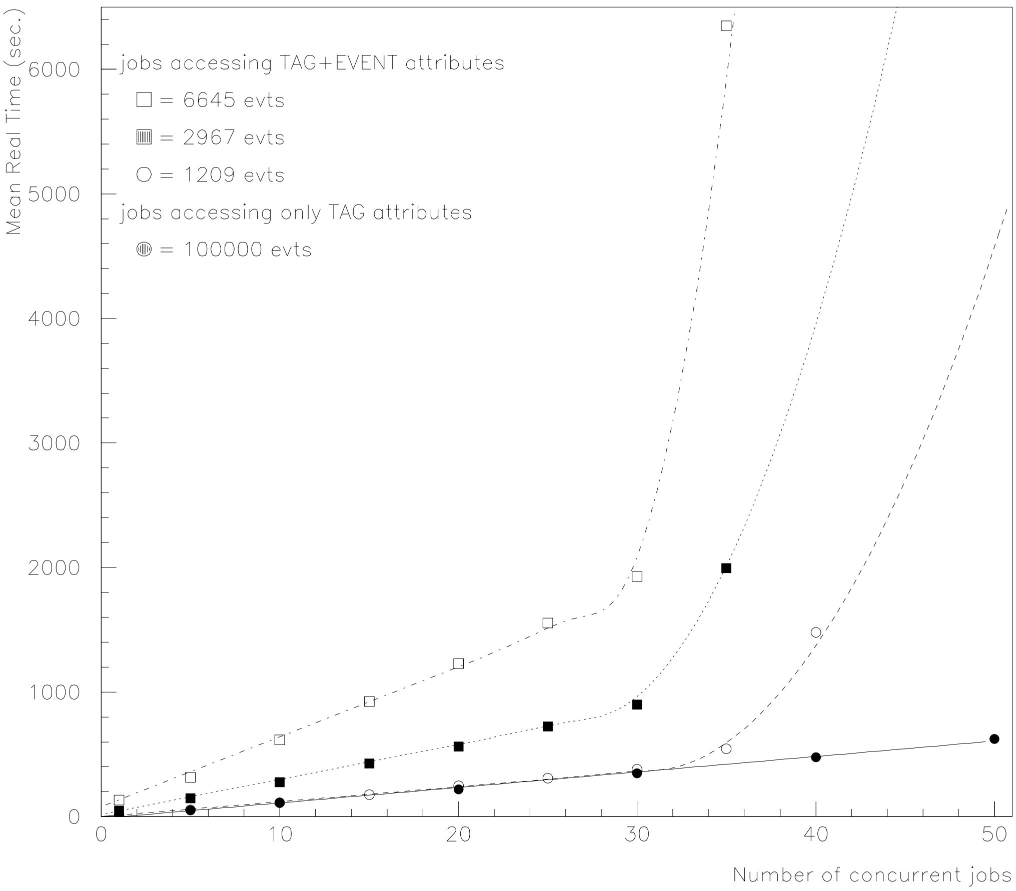

Figure 1: *Mean Real Elapsed Time (seconds) versus the number of concurrent jobs for an Objectivity cache of 250 pages*

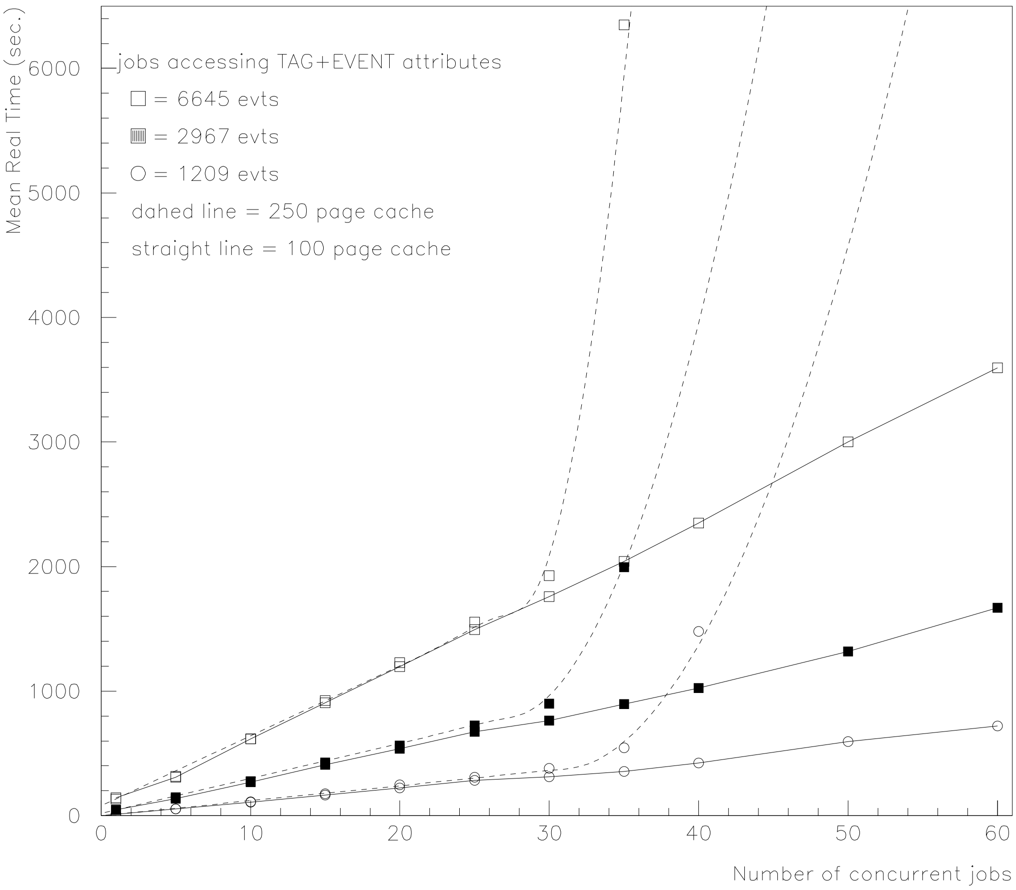

Figure 2: *Mean Real Elapsed Time (seconds) versus the number of concurrent jobs for two different cache sizes (250 and 100 pages)*

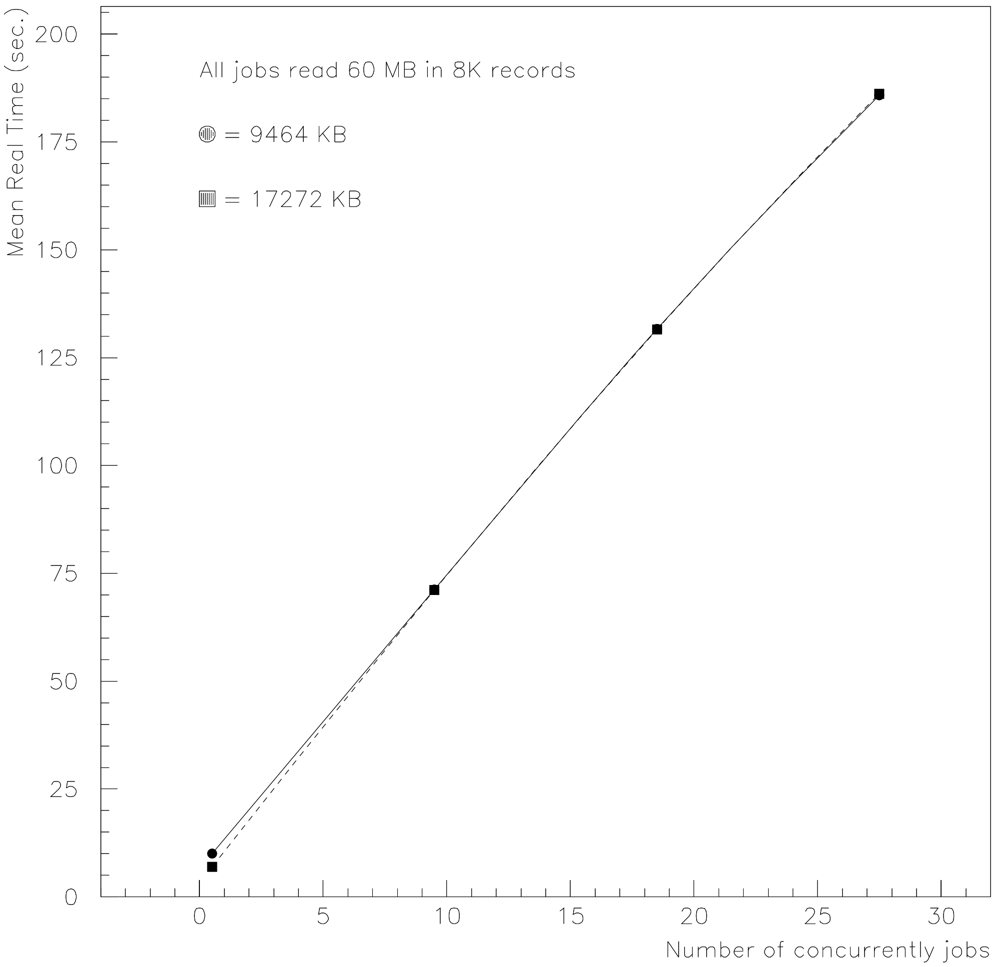

Figure 3: *Mean Real Elapsed Time (seconds) versus the number of concurrent jobs for a simple C++ program reading 60 Megabytes in records of 8 Kilobytes*

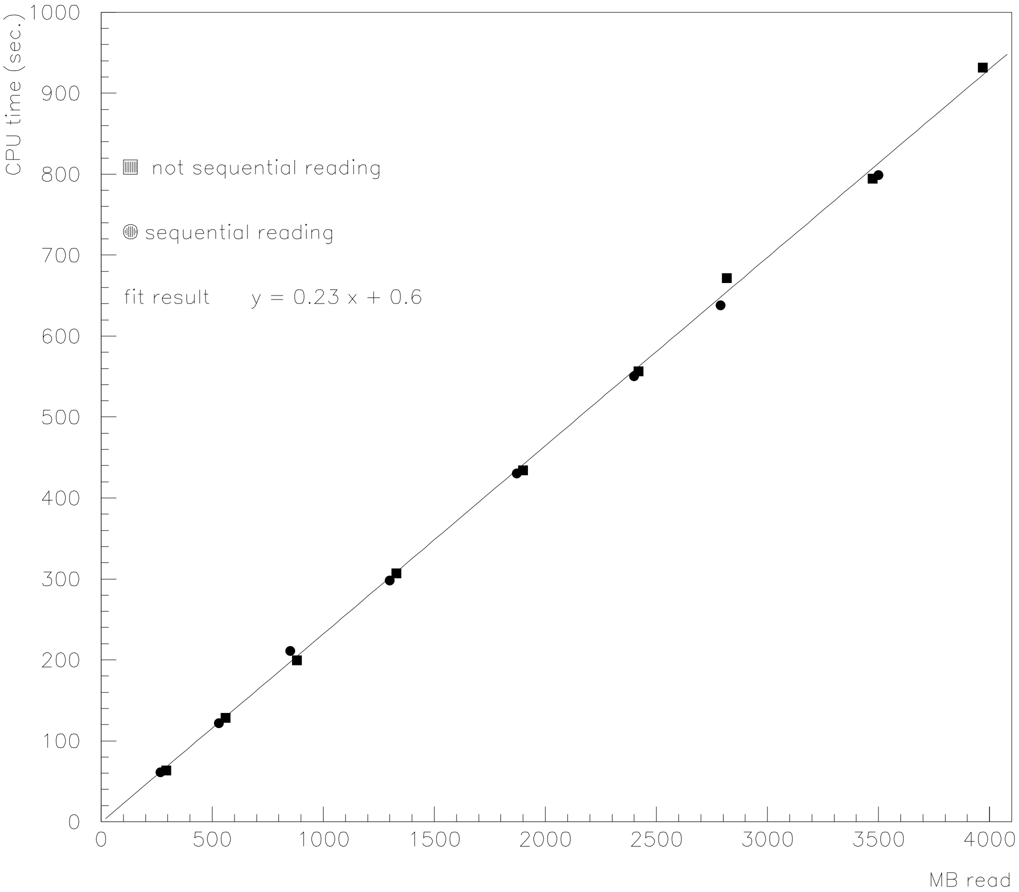

Figure 4: *CPU time (seconds) versus the number of MegaBytes read for the cases studied*

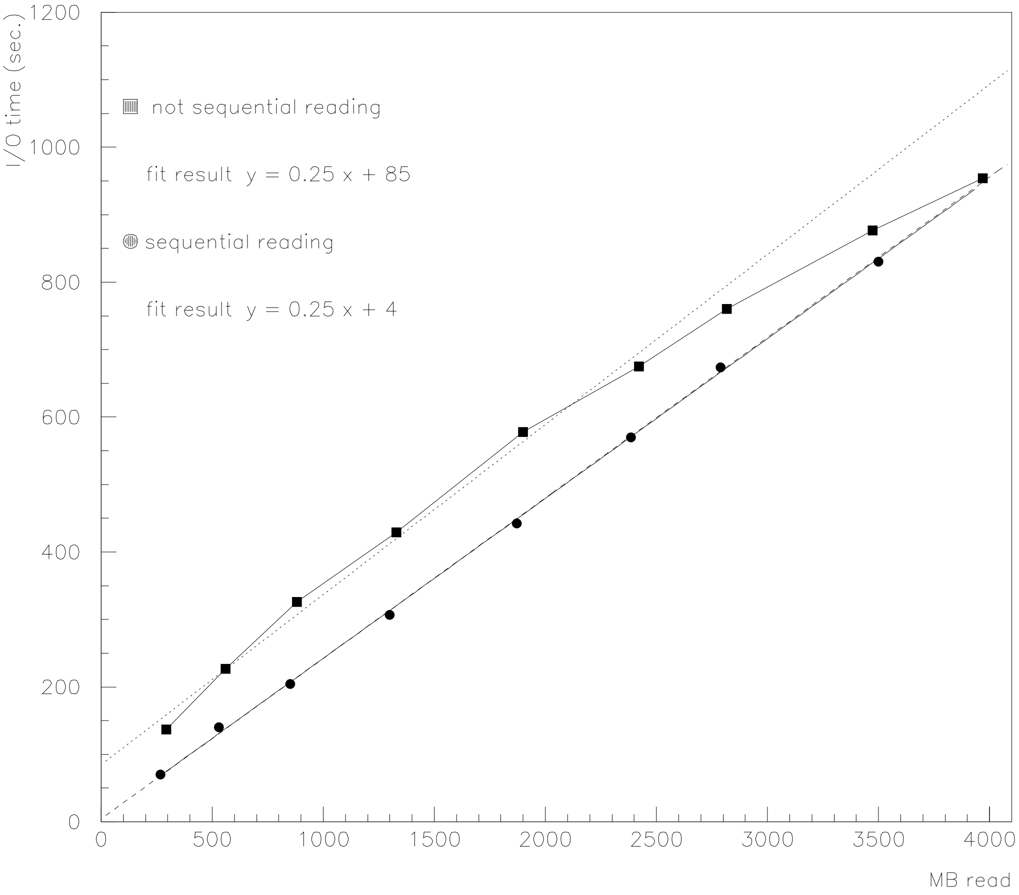

Figure 5: *I/O time (seconds) versus the number of MegaBytes read for the cases studied*

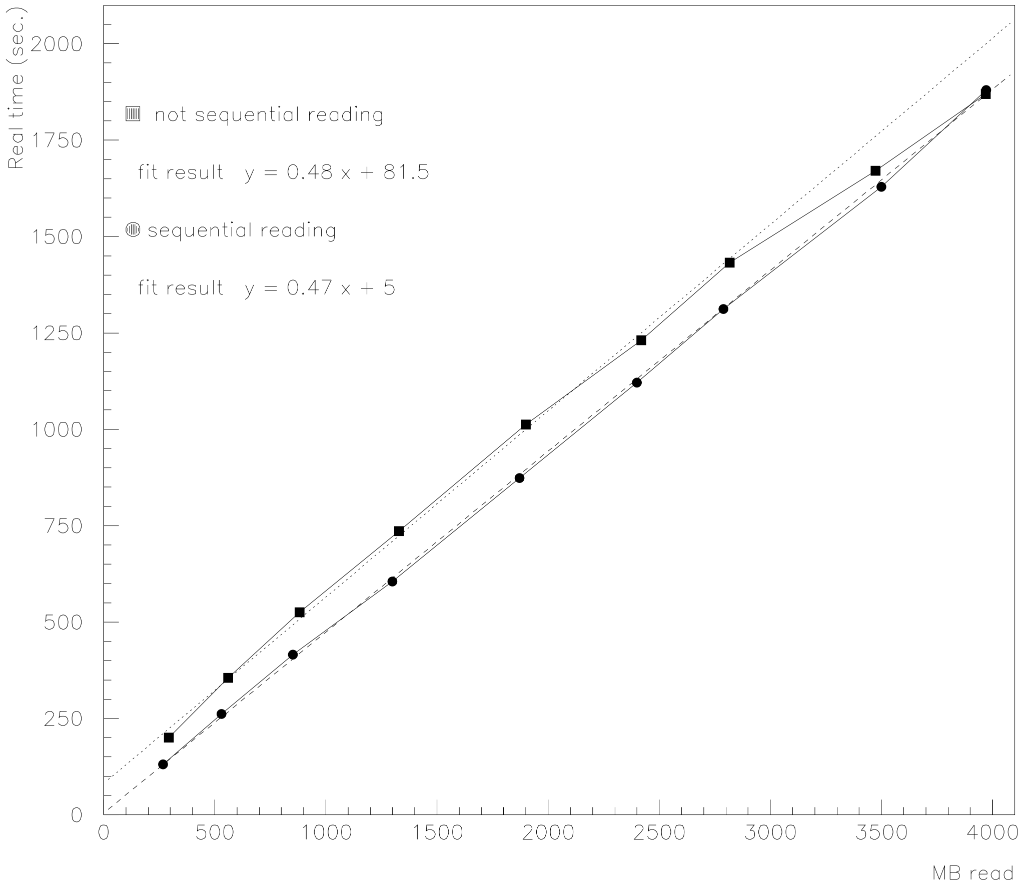

Figure 6: *Real Elapsed Time (seconds) versus the number of MegaBytes read for the cases studied*