Finding the Optimal Read Buffer Size for Grid Applications

Rodica Potolea, Alin Suciu
Technical University of Cluj-Napoca, Romania

Abstract

The concept of grid computing addresses the next evolutionary step of distributed computing. The goal of this computing model is to make a better use of distributed resources, put them together in order to achieve higher throughput and be able to tackle large scale computation problems. Performance gain is intended at each and every level of an application. All grid data access is achieved by terms of local and remote located files. We present here a study on the optimal size of the read buffer with its implications concerning the overall performance of grid and non-grid applications. This paper identifies and compares two methods of data access in a grid environment - using the storage element (SE) and local access. The results presented here come from a series of benchmarks carried on our local grid (GridMOSI), where we determine with a minimum error, the optimal interval for the size of the read buffer.

1. Introduction

A challenging computational problem is making resources available for those needing them sparingly, without necessary owning them. The birth of grid computing was very often associated with the introduction of the electrical power grid due to certain similarities [1]. Back in the beginning of the 20th century, electric power generation was possible, but the real problem was making it available worldwide without the necessity of each home consumer to possess an electric generator.

What once was the electrical power is now the computational power, and what once were the electrical generators are now high performance computing systems with large storage capabilities. Offering storage and computation capabilities to users not having those resources is the ultimate goal of grid computing. The revolutionary thing to do would be to introduce a grid infrastructure to make computing power and resources available to a greater extent.

Nowadays scientists are more and more concerned with how many floating point operations per month or per year they can extract from a computing environment, rather than considering floating point operations per second [2, 3]. With the introduction of the grid concept, more attention has been devoted to such computing environments known as High Throughput Computing environments.

Parallelization has emerged as a need for improving the response time – the time needed to generate the solution to a problem. In order to achieve parallelization of a problem there are two main approaches [4]:

• Data parallelism – refers basically to running the same piece of code on different pieces of input data. The major task here is splitting the input into chunks of data on which the same algorithm may be applied. This may be done only when the particularity of the problem allows it, when working on a subset of the initial input does not alter the final solution. Constraints may occur that can limit the overall potential for parallelism.

• Control parallelism – refers to running different sequences of code on the same data in order to obtain the result. The task here is parallelizing the problem at the algorithm level, i.e. create a parallel approach to the solution by splitting the computing responsibilities among several processes. This is a much more challenging approach, as it involves handling communication among processes which might increase the overall running time. In order to obtain a better performance, communication among processes must be kept to a minimum.

No matter what approach is followed, the data still has to be read into memory, there is no escape from here; this is usually done by means of a read buffer where one piece of data is read at a time. Not surprisingly, the size of the read buffer has a dramatic impact on the performance of the read operation, something we know from domains such as operating systems and databases. We already noticed that in such domains I/O operations represent the bottleneck and therefore the cause of limitation in increasing the performance.

Old (and bad) habits die hard, and thus the temptation to read one byte (character) at a time is so powerful that a lot of people fall into it, with drastic impact on the performance of their systems, especially when large amounts of data are to be processed.

Therefore we embarked on a study aiming to find the optimal size of the read buffer for grid applications, and we performed a series of benchmarks using our local grid (GridMOSI) and a couple of C test programs [5]. There is no similar study to our knowledge, although the buffer size was extensively studied in the context of network communication, especially for the TCP/IP protocol.

2. Grid Data Access

Before referring to the data access choices the grid user has, we present shortly the general architecture of a grid. So far we have discussed the concept of grid in terms of virtual resources and capabilities. From a physical point of view, a grids may be considered as a series of interconnected nodes called computing elements or simply CEs. These nodes, in the mesh topology that they form, communicate among themselves at high data rates. The communication lines form the backbone of the grid. Each CE has under its administration a number of computers which play the role of the worker (according to the master-worker model). These computers are called worker nodes (WNs). They represent the raw computing power resource behind the grid concept. Intuitively the greater the number of WNs a CE has, the more computing power it provides to the participating grid.

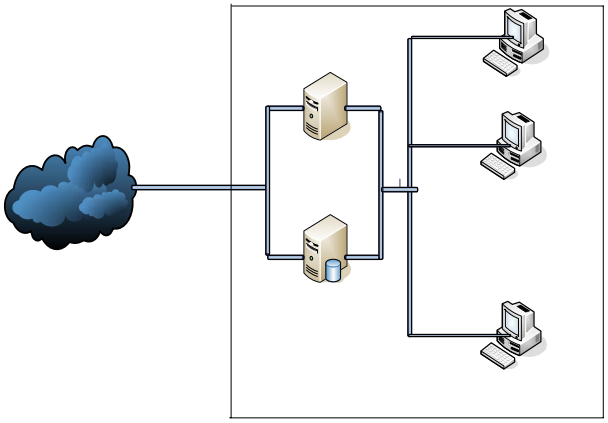

Figure 1. Simplified view of the RO-09-UTCN grid node

These WNs are not designed to be accessed publicly by any user, nor they are individually accessible from a remote site. They are meant to be administrated solely by the CE to which they subscribe. Even though physically it is possible to assign these WNs other duties (such as laboratory equipment in an university) it is not recommended to do so. Their only role is defined in the context of a grid. Each WN of our grid has a Pentium IV class processor, at 3GHz, with 1GB of RAM and 160 GB of secondary storage, running Scientific Linux 3.0.8 and gcc 3.2.3.

As stated above, access to the resources (the WNs) is achieved transparently with the help of the local CE. Generally speaking, the tasks that we submit to a grid are called jobs. In order to be allowed to submit a job in the grid, one must be authorized to do that, as the grid is a privately managed computing environment, not made publicly available but open to users from the scientific community.

At each site where there is a computing element, along with the corresponding worker nodes, there is also a storage element(SE). This gives the storage resource of the local site. Grids that are focused on high throughput computing give a unitary view of all SEs present in the grid. These grids are called data grids. Even though this unitary view of storage is not present on all grids, communication and – more importantly – data transfer among SEs is very well designed, conferring an easy, secure and reliable environment.

From what we have presented so far, one can imagine a straightforward method for data access on a grid: placing the data files on a storage element, and retrieving them during the execution of the job. The second way to access data is locally: this implies that along with the job we send the necessary input data. All these are packed together and sent as a unit called input sandbox.

We developed a test application that reads chunks of data having various sizes from a file. In order to determine an optimal read buffer size for grid applications we considered both methods for data access.

3. Reading from the Storage Element

As stated before, this access method involves the presence of a SE. In our case we used our SE from Technical University of Cluj Napoca (se01.mosigrid.utcluj.ro), part of the GridMOSI virtual organization.

In order to be able to run a job on the grid, the user must provide the necessary information about that job. This is done by using a Job Description Language. The description file (.jdl) is the one that is actually submitted to the scheduler. Below is an example of a job description file:

```
Type                = "job";
Executable          = "readFromSE";
VirtualOrganisation = "gridmosi.ici.ro";
StdOutput           = "std.out";
StdError            = "std.err";
InputSandbox        = {"readFromSE"};
OutputSandbox       = {"std.out","std.err"};
Arguments           =
"gsiftp://se01.mosigrid.utcluj.ro/home/seegrid
008/TestFile";
```

Initially the test file had to be copied on the SE. This action can be done only by a certificated user as in the case of job submission, as authentication is required for file transfer. This restriction has imposed the introduction of a transfer protocol (gridftp) that is based on the general file transfer protocol and in addition checks the needed credentials.

For the first phase of our tests, we chose medium size files and accessed them by using the storage element. Initially the buffer's size was considered to be measured in bytes, in steps of one size order, just to draw a rough conclusion. As it can be seen from Figure 2, a relatively minimum can be spotted in the vicinity of value 100,000. After this value, the read time increases in the proximity of 500 milliseconds.

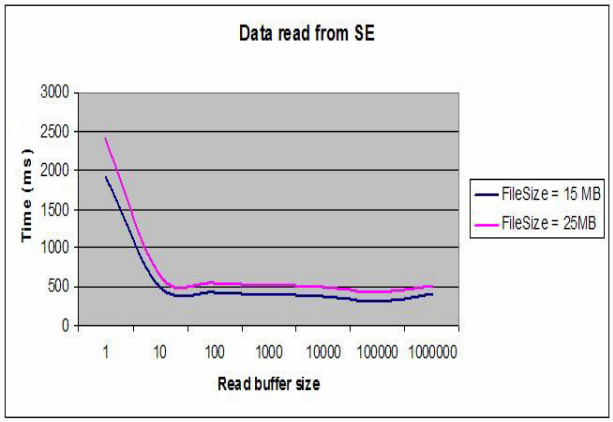

Figure 2. Variation of time needed to read a file depending on the read buffer size

The next step in our attempt to determine the buffer size for which the optimal performance is achieved was to refine the previous test. For this second test we made two changes: the buffer size was considered in increments of 10,000 and in the rage [10,000; 500,000], and also we performed our test on larger files to have a better view over the actual read process.

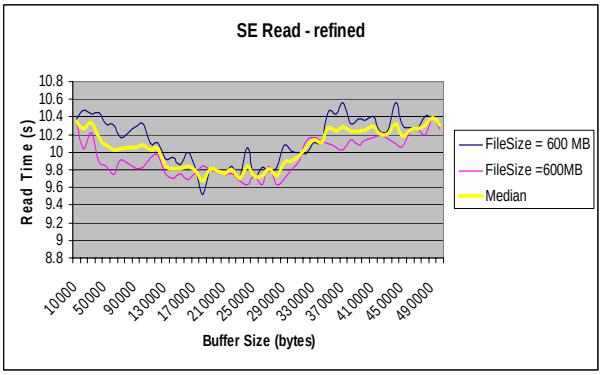

Figure 3. SE read test for larger files

The test results on larger files are influenced by the network traffic at the moment of the test. This interference is due to the nature of the actual read process when this is performed on the SE. The general process when submitting a job to the grid scheduler is: the manager identifies a CE that can satisfy the job requirements, assigns the job to the CE manager. The CE then identifies the free WNs which may start executing the job and delivers them the executable files.

When specifying in a .jdl file that the reading is to be performed from the SE, the input data is not copied locally on each WN (along with the executable files), so when performing a read operation, the WN sends a read request to the SE. The SE identifies the file to be read, reads the amount of information needed and sends it back to the WN that issued the request. All this data exchange is performed on the local network, so having a read performance graph influenced by the network traffic is obvious and inevitable.

Even though the results are influenced by network traffic, a general trend may be observed from the graphical representation (Figure 3), which may be roughly viewed in a parabolic shape, with the minimum situated in the interval [150,000; 300,000].

4. Local Read

This is the second method for data access in a grid environment. The input files are not remotely placed on the SE, they are copied locally, on each WN local disk. This implies the existence of several replicas in case the job is intended to be run on multiple WNs (as in the case of MPI jobs). In case the input data is of significant amount it is not advisable to store multiple replicas, a lot of storage space would be wasted.

The concept of data grid, described earlier, works on huge data and it is not uncommon for a job to need access to hundreds of GB of data. Not even the distribution of such data would be extremely expensive, but WNs are simple computers, workstations with simple architecture where the storage of such an amount of data may be impossible. As a consequence, it has been imposed a limit on the amount of data that can be transferred in one input sandbox, and this is 10 MB. Due to this limitation our tests had to be done on files smaller than this maximum value.

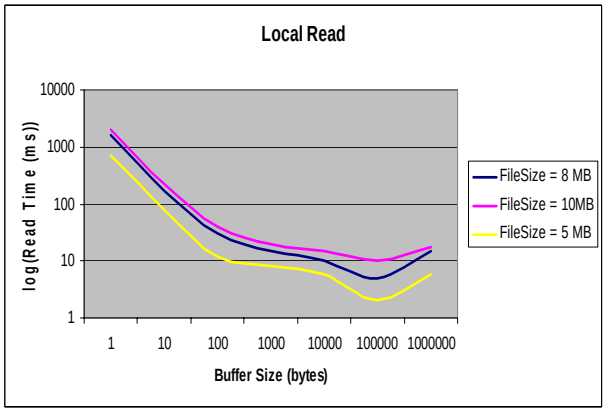

Figure 4. Time variation for local read

It can be noted from the graphic above (Figure 4) that the best response times are obtained with a buffer size of approximately 200,000 bytes. Due to the fact that file size is limited, the refinement process, applied to the read from SE method, is not applicable here.

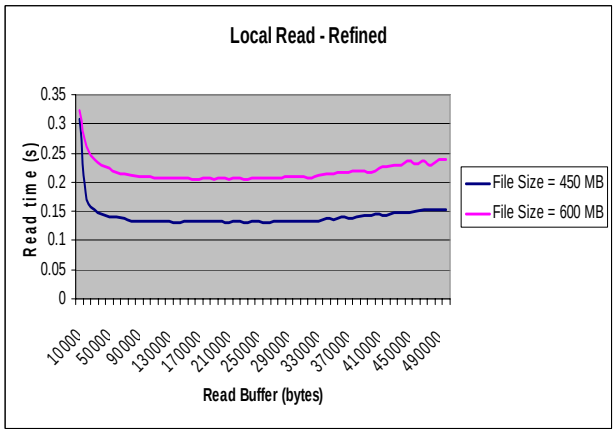

Figure 5. Refined local read test

In its essence the local read mechanism on the grid, is the same as any local read on a standalone station.

This analogy has allowed us to extend the refinement process for the local read to a standalone computer. (Figure 5).

The general behavior observed when using smaller files is maintained when using larger files. There is a massive improvement until buffer reaches size 50,000 and then the read time remains at a fairly constant value in the interval [130,000; 290,000].

5. Conclusions

We have analyzed the two methods for data access in a grid environment. Based on the results obtained we have reached the conclusion that the optimal read buffer size is situated in the interval [130,000; 290,000] for local read method, and [150,000; 300,000] for the SE read method. In order to make the design of an application uniformly, to have the same access parameters in both cases, we propose a buffer size of 200,000 bytes. This value was chosen so that to be at a relative equal distance from the limits of both intervals, and also to have a greater recall factor (to be easy to remember).

In this paper we have presented the benefits of grid computing, the novelty it brings to the local academic community, with its presence at the Technical University of Cluj Napoca, namely GridMOSI. The results obtained from this study are used in the implementation of our GridMOSI library [6] and have a major impact in improving the performances of the methods implemented so far.

6. References

[1] I. Foster, C. Kesselman and S. Tuecke, "The Anatomy of the Grid", *International J. Supercomputer Applications*, 15(3), 2001.

[2] Ian Foster, "The Grid: A New Infrastructure for 21st Century Science", *Physics Today*, 55(2): 42-47, February 2002.

[3] I. Foster and C. Kesselman, "*The Grid: Blueprint for a New Computing Infrastructure*", Morgan Kaufmann Publishers, 1st edition, 1998.

[4] L. Ferreira and V. Berstis, "*Fundamentals of Grid Computing*", IBM Redpaper, 12 November 2002, IBM Form Number REDP-3613-00.

[5] B. W. Kernighan, D. M. Ritchie. "*The C Programming Language, Second Edition*". Prentice Hall, Inc., 1988, ISBN 0-13-110362-8.

[6] R. Potolea, A. Suciu, A. Mascasan. "*Benchmarking the GridMOSI Library*". Echallenges Conference, 2007, The Hague, Netherlands, (accepted).