Paper 019-2010

Distributed Enterprise Scheduling, Workload Balancing, and Parallelized Workloads for Productive Performance at Bank of Italy Research Unit with SAS® Grid Manager on an HPC Cluster

Giuseppe Bruno[1], Bank of Italy, Rome

ABSTRACT

In an attempt to provide more efficient and more cost-effective access to a wide range of econometric and statistical packages, to the different company databases, and to a comprehensive range of Web-enabled services, the Economic Research Unit of the Bank of Italy has decided to replace its Symmetric Multiprocessors (SMP) RISC6000/AIX-based system with a Red Hat Linux High Performance Computing cluster. The new system involving increased data volumes, more complex and longer runtime processes, required an efficient and flexible tool enabling the Research department of the Bank to perform SAS® job scheduling and workload balancing and to parallelize workloads in a distributed environment. Hence the Research dept of the Bank is deploying SAS® Grid Manager to deliver highly productive capabilities that help achieve optimal usage of computing resources and leverage the organization infrastructure in order to increase performance and reduce costs.

INTRODUCTION

The last fifteen years has witnessed shrinking technological improvements in the design and mass production of faster computing machines. The straightforward technique of cramming more CPUs, controllers and on board memories overdriven with ever faster clock rate is approaching the physical limits concerning the heat dissipation per unit of volume. Other physical barriers related to the quantum behavior of the matter prevent us from relying on the well known Moore empirical law stating a doubling in the performances of any computing component every 18 months.

Today in order to scale up performances as fast as in the past or even at a faster rate we have to take into account a different framework requiring the employment of different hardware as well as software computing platforms based on the cooperation of many computing engines under the oversight of a suitable software environment.

The production of parallel computing engines leveraging the computing power of many standard off-the shelf computers has now become very popular. Unfortunately, the same thing cannot be said for the development and release in production of easy to deploy software package effectively taking advantage of the parallel structure of these new computing engines.

The last years have sprout some programming paradigm aimed at helping the job of computer aware people. With the latest versions of the SAS package taking advantage multiprocessors and multicomputers engines comes within the reach of statisticians and economists.

THE HARDWARE TECHNOLOGICAL PICTURE

Today's technological offer for parallel computing comes in essentially two flavors:

- Simmetric Multi Processor (SMP also called Uniform Memory Access);
- Non Uniform Memory Access (NUMA).

With SMP architecture all memory access request are issued to the same shared memory bus. This works fine for a relatively small number of CPUs, but the latency problems arise with the shared bus appears when you have more 16/32 CPUs competing for access to the shared memory bus. Examples of this architecture are: IBM RISC6000, SUN Ultrasparc and the SGI/MIPS machines.

[1] The views expressed here are those of the author only and do not involve in any respect the views of the Bank of Italy.

The NUMA architecture was designed to overcome the scalability limits of the SMP architecture. NUMA attempts to remove the bottleneck on the shared bus by limiting the number of CPUs on any single memory bus, and connecting the various *nodes* by means of a high speed network among the different processors. Examples of platforms using this paradigm are Sequent server and the SGI 2000 and 3000 series. Besides from these systems also LINUX provides its NUMA answer with clustering a set of X86_64 nodes by means of an high speed Inter processor communication network.

The recent trend in hardware has been towards increasing the number of system bus, each one devoted to a small number of processors. Each group of processors has its own memory and possibly its own I/O channels. However, each CPU can access memory connected with the other groups in a coherent way. Each group of processors is called a NUMA node. The number of CPUs within a NUMA node depends on the hardware vendor. On NUMA hardware, some regions of memory are on physically different buses from other regions. Because NUMA uses local and foreign memory, it will take longer to access some regions of memory than others. *Local memory* and *foreign memory* are typically used in reference to a currently running thread. Local memory is the memory that is on the same node as the CPU currently running the thread. Any memory that does not belong to the node on which the thread is currently running is foreign. The ratio of the cost to access foreign memory over that for local memory is called the NUMA ratio. When the NUMA ratio is 1, we are working with symmetric multiprocessing (SMP). The greater the ratio, the more it costs to access the memory of other nodes.

To summarize, all these multi-processor or multi-computer architecture provide means to overcome the technological and physical barriers posed by the original single processor design in a very cost efficient way.

THE AVAILABLE SOFTWARE FRAMEWORK

The availability of a parallel computer doesn't provide straightforward speed-up of users' applications. In order to take advantage of a parallel hardware platform a suitable software framework is required. For highly specialized programming people there are two main programming paradigms:

- OpenMP which works on SMP systems,
- MPI which works on both SMP and distributed memory systems (ie, clusters).

OpenMP carries out multithreading, where the master thread forks spawning a specified number of slave threads assigning each one of them a fraction of a task. The section of code that is meant to run in parallel is marked accordingly, with a preprocessor directive (FORTRAN, C and C++) that will cause the threads to form before the section is executed.

After the execution of the parallelized code, the threads join back into the master thread, which continues onward to the end of the program. Each thread executes the parallelized section of code independently. "Work-sharing constructs" can be used to divide a task among the threads so that each thread executes its allocated part of the code.

Message Passing Interface (**MPI**) is a communication protocol used to write parallel program on distributed memory computers. MPI is a message-passing Application Programmer Interface (API), together with protocol and semantic specifications for how its features must behave in any physical implementation.

Today, MPI has become the standard for communication among processes employing the parallel programming paradigm for a distributed memory platform. Actual distributed memory supercomputers such as computer clusters often run these programs. Although these definitions, MPI programs run regularly on both shared and distributed memory computers. Designing and writing programs around the MPI model takes advantage of the NUMA architectures by extensively employing the memory locality concepts. Most MPI implementations consist of a specific set of routines (API) callable from Fortran, C or C++ and from any language capable of interfacing with such routine libraries.

These two programming models are used with classical third generation compiled languages like FORTRAN, C and C++. Most of the economists and statisticians don't employ these programming languages because they would be much less productive than using other statistical packages where the basic estimation and simulation algorithms are tested, well documented and already available.

In order to bring the performance gains provided by the parallel hardware to many applied researchers we need to modify the very basic structure of our single-thread statistical and econometric packages. Therefore it is important to equip these end-user packages with full-fledged parallel procedures.

In the Research Department of the Bank of Italy, many different statistical packages are employed for different purposes. Among them the following packages:

- Matlab (MPI distributed memory)

- STATA/MP (SMP shared memory)
- SAS EBI (shared/distributed memory)

provide native tools for reducing computing times by taking advantage of the parallel features of our computing platforms. In this paper we focus on the capabilities provided by the SAS engine for distributing its workload among different computational engine.

THE HPC CLUSTER FOR THE ECONOMIC RESEARCH AREA OF THE BANK

The Economic Research area of the Bank of Italy is composed by about 500 people. Among its various tasks, the Research area of the Bank of Italy collects many statistics, analyses the economic conditions and makes short and medium term forecasts of the economic evolution. Actual implementation of these tasks is carried out by using an ample set of information tools and a reliable and performing hardware and software infrastructure.

At the end of 2010, the Economic Research area of the Bank of Italy will replace its Symmetric Multiprocessors (SMP) P570 AIX-based 12 CPU system with a RedHat LINUX HPC cluster. Each new computing node is based on an AMD Opteron processor with 4 CPU, each one equipped with 4 core.

The main purpose of the new Computing infrastructure is to provide a flexible and cheap access to a wider range of econometric & statistical packages, a seamless access to the different company databases and a comprehensive range of web-enabled services.

The computing infrastructure is totally duplicated on two separated geographical sites for disaster recovery reasons. Each site is equipped with 8 Computational and 4 Infrastructure service nodes. A scheduler based on the Load Sharing Facility (LSF) is employed to assign users to a particular node chosen according to a given workload balancing policy.

Figure 1 shows a draft schema of the new computing infrastructure.

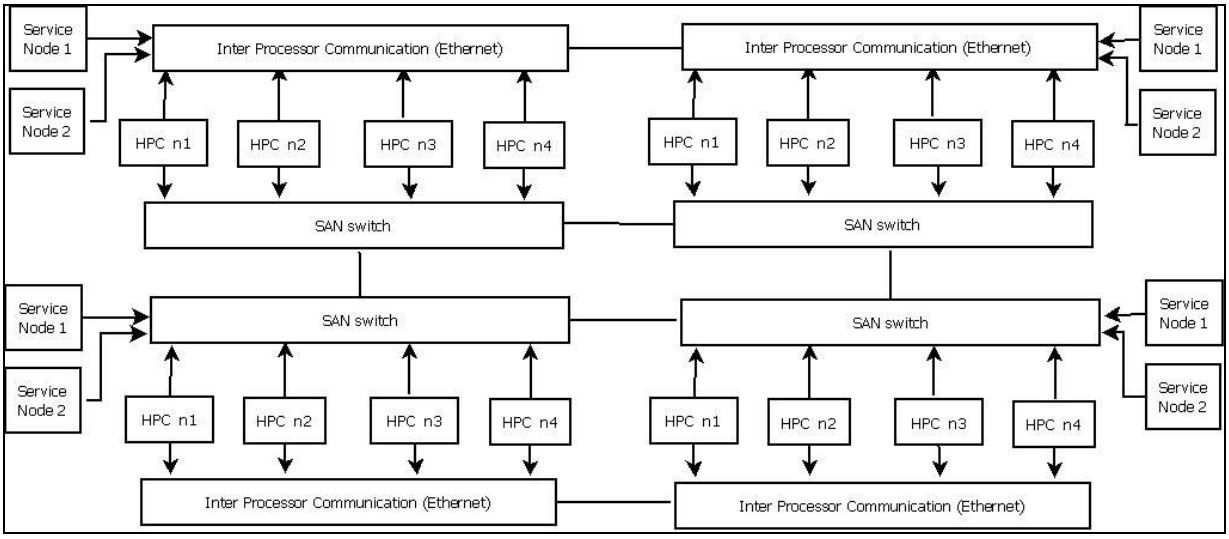

Figure 1

Even though the current SMP based architecture provides simpler means for parallelization, so far, most of the packages available to the users did not take advantage of the multiprocessors features of the computing systems. The SAS Grid Manager component is going to be deployed on our High Performance Computing Cluster to explore and measure the flexibility and performance gains in the parallelization of some CPU bounded computations.

EXAMPLES OF CPU BOUNDED PROBLEMS

In order to compare the working of the SAS package we ran a set of CPU bounded procedures. The CPU bounded program employs the PROC SURVEYSELECT. Statisticians often employ survey methodologies to obtain information about the whole population by selecting a small sample from that population.

Given these sort of problems we have tried to compare execution time for some benchmark with the previous SMP RISC6000 platform. As a second step we have employed standard MP/connect features to provide a rough cost measure of the work for parallelizing current SAS program and to check how our problems were scaling with the number of nodes. Here the focus is to check the actual reductions in computational time with these new architectures. These time savings can be employed to increase the statistical accuracy in the information needed by the Bank for carrying out its policies.

Appendix A has a SAS program we ran on our computing platforms. This example code has been run on both the AIX/RISC6000 SMP and the new LINUX cluster under three different conditions:

- single thread,
- two threads,
- four threads.

The preliminary results show a single thread time reductions of about 60% induced by the new CPU technology. Splitting the original single thread program in 2 and 4 threads we have achieved the results summarized in the following table:

	AIX/RISC6000 (SAS 9.1.3)			RedHat EL 5.4 (SAS 9.2)		
	Single thread	Two threads	Four threads	Single thread	Two threads	Four threads
Op. system measure	62 sec	33 sec	56 sec	25 sec	29 sec	28 sec
SAS measure	62 sec	11 sec	14 sec	25 sec	5.7 sec	6.9 sec

The table shows the times reported by the UNIX command time and those written by SAS at the end of the log files of the programs. Although these are very preliminary results and deeper investigation is required, the measure provided by the SAS logs show a definite improvement going to two threads. For this particular problem there is nothing to gain going to four threads. This behaviour could depend upon the trade-off between task-computation and task-distribution. Further experimentation will shed light on the different conditions of the problem (number of replications, variables etc) affecting these results.

WHAT PROBLEMS CAN WE SOLVE WITH OUR GRID INFRASTRUCTURE?

In the porting phase of the whole set of econometric and statistical packages we decided to deploy the SAS EBI 9.2. In this pre-production phase we are going to install Grid Manager tool. The need for this component comes from different reasons:

1. Increased data volumes growth
2. Running larger and more complex problems
3. Deploying a more flexible IT infrastructure

The test of the SAS Grid Manager allows us to completely check the degree of utilization of all computing resources. Moreover we are going to run a thorough test on the set of load balancing and grid management functionalities. These tools automate the management and optimization of application processing in the grid computing

environment. When running processes in a grid, SAS Grid Manager dynamically determines node availability and monitors grid nodes to determine which node is the best candidate to process the next workload request. This determination can be based on many factors, but it often considers the current load under which all grid nodes are running at any given time. The node that has the lowest CPU load becomes the best candidate on which to run the next workload segment. This dynamic capability greatly increases job runtime performance by distributing processes across a wider array of resources capable of handling the greater computing load.

In our first experimental phase we test:

1. The cooperation between the SAS grid manager and a scheduler such as the LSF. Using LSF along with SAS Grid Manager, scheduled jobs are targeted to run on the grid. This makes the whole pool of resources in the grid available to run scheduled jobs. The scheduling server manages the resources in the grid so that jobs are efficiently dispatched across the available machines in the grid.
2. The performance comparisons of some CPU intensive jobs ran on the old and the new computing infrastructure. Many variables can affect performance; for example, the number of machines in the grid, the amount of memory or cache per node, I/O capabilities, and the current workload. Furthermore we test the ability of SAS Grid Computing to allow maximization of the use of spare CPU cycles and thus achieving significant improvements in average processing speed for SAS programs.

First of all this experience teaches us how to improve the management of the whole SAS workload submitted by the users. As a second result we learn how efficiently parallelize a SAS program among a cluster of distributed memory computing engines. Using the grid with the right scheduling tools allows the exploitation of all the load balancing capabilities, such as queuing, prioritization, workload balancing, and resource management, for this class of interactive submission. High-priority jobs can even preempt lower priority work so that the most critical business processes execute first. This enables users to leverage all of the available resources in their distributed environment for job processing, thereby speeding up long-running tasks and increasing user's productivity.

CONCLUSION

Computing clusters based on distributed memory feature a tremendous potential to reduce costs and increase performance. The key issue for leveraging the most a grid environment is a design ensuring a good trade-off between the time spent in computing and that spent in communicating. When testing and benchmarking the parallelizing techniques it is critical examining each SAS application and determine the tasks effectively dispatched to different nodes for successful execution. This preliminary exploratory phase ensures that the computing infrastructure will meet the performance expectations and helps us in porting SAS workload to the grid.

SAS Grid Manager along with the EBI framework might prove very helpful in reaching better performance goals. SAS Grid Manager experimentation allow us to evaluate the effectiveness in making use of grid resources for the parallelization of long-running task.

REFERENCES

- C. Doninger, T. Keefer and N. Wills **"Best Practices for Data Sharing in a Grid Distribuited SAS Environment"** 2008. SAS Best Practice Documents
- C. Doninger, R Tobias, "**The %Distribute System for Large-Scale Parallel Computation in the SAS® system**" 2009. http://support.sas.com/rnd/app/papers/distConnect.pdf
- J.C. Gober, A. Sutha "**Using Formats, MP Connect, and Other SAS Efficiency Techniques to save Time and Disk Space**" SAS Global Forum 2009

ACKNOWLEDGMENTS (HEADER 1)

I would like to thank all the technical team responsible for the optimization and tuning of the Cluster system at the Bank of Italy.

CONTACT INFORMATION (HEADER 1)

Your comments and questions are valued and encouraged. Contact the author at:

Name: Giuseppe Bruno
Enterprise: Banca d'Italia
Address: Via Nazionale, 91
City, State ZIP: Rome, 00169 Italy
Work Phone: +39 0647924164
Fax: +39 06 4747820
E-mail: giuseppe.bruno@bancaditalia.it
Web:

APPENDIX A – SAMPLE CODE

CPU bound example:

```
options fullstimer;
options autosignon=yes;
options sascmd="!sascmd";
LIBNAME Ibf "/_896sia/brunogi/sasuser9/CPUbound";
LIBNAME mydata "/_896sia/brunogi/sasuser9/CPUbound";
data ibf.a;
MERGE ibf.carcomp ibf.RISFAM04;BY NQUEST;
if cfred=1; acom3=acom4c+1; if acom4c=4 then acom3=3;
strato=(ireg*10)+acom3; IF STRATO= 21 THEN STRATO= 11;
%LET VAR=C; keep nquest pesofl &VAR strato ;
proc means  mean stderr; var &VAR;
weight pesofl;run; proc SURVEYmeans DATA=ibf.A mean std cv;
var &VAR;
WEIGHT PESOFL; STRATA STRATO;
run;
proc sort ; by strato;
proc freq; tables strato /out=ibf.dim;
data ibf.dim;
set ibf.dim;
rename count=_NSIZE_;
rename PERCENT=_RATE_;
run;

rsubmit process=task1 wait=no;
   %let var=C;
   libname mydata '$HOME/sasuser9/CPUbound';
   proc surveyselect data=mydata.a rep=750 SAMPSIZE=mydata.DIM out=mydata.camp1
   method=urs;
   strata strato;
   id nquest pesofl &VAR;RUN;
   PROC MEANS DATA=mydata.CAMP1 SUM;
   VAR NUMBERHITS;   CLASS REPLICATE;   RUN;
   DATA mydata.CAMP1;
   SET  mydata.CAMP1;
   PESO=PESOFL*NUMBERHITS;
   proc sort;by replicate;
   proc means noprint;
   var &VAR;output out=stat1 mean=media;
   WEIGHT PESO;by replicate;
   proc means mean std cv;
   var media;
   run;
endrsubmit;
/* remote submit second task */
rsubmit process=task2 wait=no;
   %let var=C;
   libname mydata '$HOME/sasuser9/CPUbound';
   proc surveyselect data=mydata.a rep=750 SAMPSIZE=mydata.DIM out=mydata.camp2
   method=urs;    strata strato;
   id nquest pesofl &VAR;RUN;
   ….
   ……
endrsubmit;
waitfor _all_ task1 task2;
data mydata.finale;
     set mydata.camp1 mydata.camp2;
     proc means data=bruno.finale;
run;
```

In the previous program I have split the original surveyselection task for 1500 replications with two independent tasks with 750 replications each. The final step consists in joining the two different datasets for computing some cumulative statistics.