PROCEEDINGS OF SCIENCE

Real-time grid computing for financial applications

Stefano Cozzini*
CNR-INFM Democritos and EGRID project
E-mail: cozzini@democritos.it

Riccardo di Meo, Ezio Corso
EGRID project ICTP
E-mail: {dimeo,ecorso}@egrid.it

We describe the porting of a test case financial application on the EGRID infrastructure, and the approach we used to guarantee real time computing.

Grid Technology for Financial Modeling and Simulation
3-4 february 2006
Palermo, Italy

*Speaker.

PoS(GRID2006)007

1. Introduction

Computing grids are attractive for large scale financial applications: this is especially true for dynamic financial services where applications must complete complex tasks within strict time intervals. The traditional response has been to over-provision resources for ensuring plenty of headroom in availability, resulting in the maintainance of large inhouse and unused computational resources with a great cost in terms of infrastructure. Worse, nowadays some of these complex tasks need an amount of computing power that is unfeasible to keep idle most of the time in house.

Computing grids can deliver the amounts of power needed in such a scenario, but there are still large limitations to overcome. In this brief report we address the solution we developed to provide real time computing power through the EGRID facility [1] for a test case financial application.

2. The computational problem

The test case we consider is an application that estimates the sensitivities of a set of stocks to specific risk factors: technical details about the procedure can be found elsewhere [2]; we will present here only the computational details of the application to better define the problem we faced, and the solutions adopted for porting it to the grid.

The application employs a Genetic Algorithm *GA* approach, and so starts by randomly generating an initial population of solutions which get self-crossed; then a Kalman filter is applied to evaluate the goodness of the self-crossed solutions; the best ones are chosen and they get called the *first generation*. All subsequent generations are similarly produced: mutations are introduced in the current generation and a Kalman filter is applied to gauge each mutated solution; the best ones are selected for to the next generation. This process goes on until a predetermined number of generations is reached. In order to better focus on the most meaningful aspects of the approach, for the rest of this discussion the Kalman filtering will be considered as a separate step from the rest of the Genetic Algorithm. The application, therefore, can be seen as consisting of a Genetic Algorithm *GA* step that produces new generations, and a Kalman filter step that evaluates the goodnes of a generation.

As far as the GA step is concerned, a standard run is characterised by two different parameters: the number of generations or equivalently the number of iterations, and the size of the population in each generation. For the Kalman filter step, in a standard run the sum of all evaluations is roughly 90% of the full computational cost.

The data-size of the program depends on the number of stocks included and on the length of the time series for each stock considered. For a small size problem like the case study we started from, there are 50 stock options with time series of 100 elements, and it takes about 300 seconds to complete using 1000 generations each containing 600 individuals. The computational demand in terms of memory scales quadratically, while the total computational power needed by the Kalman filter has a complicated pattern.

The behavior, as a function of the number of generations, is depicted in figure 1. There, it is also reported the histogram of CPU time for a *single* Kalman call: clearly the contribution of a single evaluation is negligible. Still, as one can infer from 2, for all possible combinations of the number of individuals vs the number of generations, the *overall* CPU time spent in the Kalman

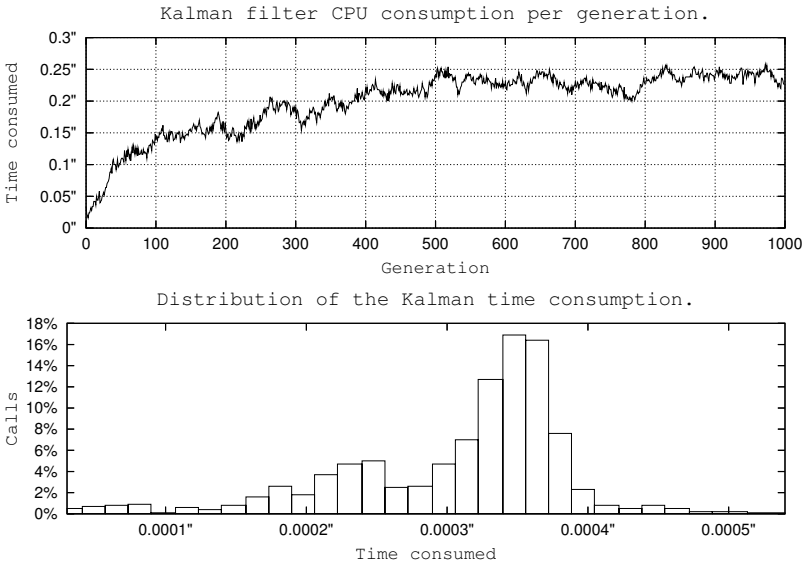

Figure 1: Behavior of the Kalman filter (see text for details)

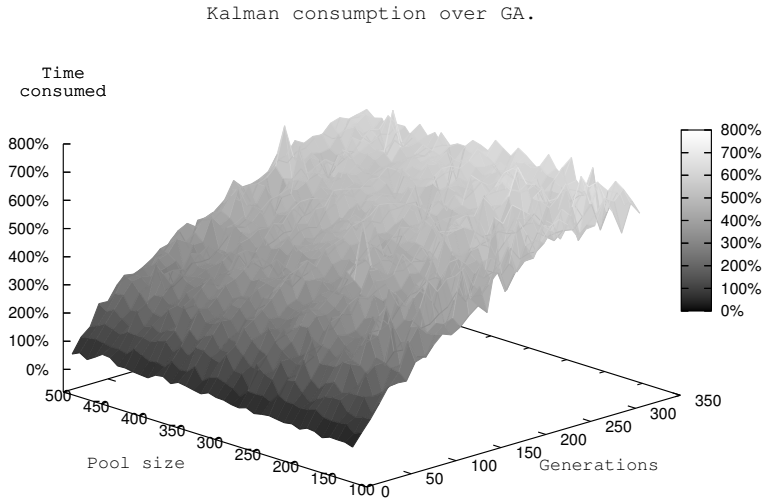

Figure 2: Kalman procedures CPU consumption over GA's

filter is several times bigger than that spent on GA even for such a simple problem (in terms of assets number). This justifies our first implementation as will be discussed in the next section.

The problem size of the previous example, although feasible on a single workstation it is far from real world problem sizes where a large number of stock options must be considered, and where the time series are much longer too. The computational power needed to solve realistic use cases within few minutes must, then, be found on a grid infrastructure.

In the next section we will discuss the technical implementations we adopted when porting the application to the EGRID computational grid. This infrastructure is mainly based on EDG/LCG middleware: it is a collection of Computing Elements (CEs) that maintain job queues to Worker

Nodes (WNs) that actually carry out the computations (either serial or MPI based). Storage Elements (SE) supply data persistence services. CEs and SEs are monitored by an Information System (IS). The Resource Broker (RB) represents the heart of the middleware: it identifies the most suitable CE for a job by querying the IS; it then submits the jobs; and finally it takes care of all the associated data management operations. As far as the end-user goes, commands to interact with the grid are collected in the User Interface software (UI) that can be easily installed on almost any Linux machine.

3. Technical implementation

We implemented different technical solutions for our application following a trial and error aproach. We will now briefly review all of the attempts.

All implemented solutions rely on a *job reservation mechanism*: grid resources are allocated in advance to eliminate latency due to the grid's job submission mechanism. In this way as soon as there are enough resources allocated, interaction can take place in real time. The drawback is that since it is an advanced booking strategy, when the underlying grid services are of a *best effort* type, it could be unfeasible. For this experimental work it is not important though, but the limitation should be taken into account when approaching production runs.

The booking mechanism has been implemented by running an early job submission that secures the availability of WNs at any subsequent time. Each node that gets *pooled* in this way, runs a program that regularly checks a specific host (usually the UI, but not necessarily). That host enrolls the calling WN as a computational resource for a user's program, as soon as the user executes that program. Therefore when the program ends the results are available without the delays introduced by WMS of the grid: they are available in real time. Since the WNs remain booked, they are ready to be enrolled again for other program executions, eventually being freed by the user.

This approach, where the WN asks to be enrolled in a computation thereby acting as a client, is needed because WNs cannot be reached directly from an external host (for example from the UI).

3.0.1 A first implementation: master/slave paradigm

The first, naive, implementation is simply based on the observation that the overall time taken by the Kalman filter is much larger than that taken by the GA (see figure 2). The idea is therefore to implement a *master/slave paradigm* where the master running on the UI outside the grid, takes care of generating individuals that will then be distributed to the grid WNs, to be evaluated by the Kalman filter. So each WN receives some load to process.

This approach, despite its simple implementation that still preserves the serial nature (and therefore obtains identical results to the stand alone application) has the main advantage of being dynamic: simulation can start with just a single WN but later on, when other resources become available and can contribute to the execution, the master can easily leverage the newly freed computing power.

It has some severe limitations in terms of network overhead: copying data between the UI and the WN is quite slow and does not scale at all. Data transferred back and forth increase linearly with the size of the problem. Moreover there is a large network latency overhead due to the fact that a

transfer takes place *each time* a Kalman filter evaluation is performed. Even if Kalman evaluations can be grouped together for each processor, latency associated with each of these groups makes the approach unfeasible. Overall, then, a new algorithm is needed.

3.0.2 A better approach: the isles algorithm

In this second approach we modified the algorithm parallelizing the GA step as well. Our approach is now to have N independent Genetic pools, the so called *isles*. At a fixed number of generations the isles exchange in a round-robin way their individuals: for example *isle 1* will receive (N-1)/N of the population, coming from the remaining $N-1$ isles. Only 1/N of the original population remains unchanged after this operation, which we define *migration*.

Both the GA and Kalman filter steps are now independently run on each isle (that is on each WN), all with different seeds for ensuring distinct solutions. Network overhead is greatly reduced: any remaining communication happens once in a while and it is a global operation among isles.

The main problem we faced was how to make each of the allocated WNs aware of the others, since their hostnames were not known in advance. This was solved by having each WN contact the UI for transmission of the respective hostname. The UI then broadcasts back the complete list of hostnames, once the right number of WNs has communicated its hostname. Communication can now take place between WNs through usual sockets.

There is an evident limitation in the case of heterogeneous grid environments. When a migration takes place there's a barrier on all isles that limits the computation's speed: the slowest machine must be waited for.This can be easily solved by implementing an algorithm that starts migrations at specific *times* instead of waiting for a specific *number of generations* to be reached. Migrations are, therefore, carried out at predetermined time intervals.

At the end of the simulation a WN receives the best solutions from the other ones and tries to connect to a computer with a resolvable hostname (usually the UI), supplying the user with the final results in real-time.

A more complex limitation arises from the fact that a doubling of computational power is not likely to yield as much a gain in solution quality, with respect to the first algorithm. This is so, not withstanding the fact that the isle approach mitigates the natural tendency of GA to evolve too homogeneous populations, which then tend to get stuck in local minima.

This approach could also be implemented using MPI but we consider our solution superior because it preserves the dynamic nature of the first one (this is not easily done with MPI). Moreover it can be executed on any grid facility and not just on the ones where MPI is supported.

3.0.3 The third approach: the continent algorithm

Since WNs belonging to different grid sites cannot communicate, the above algorithm is limited to a single site: i.e. all the reserved WNs must be in the same grid site. To overcome this limitation we implemented a third approach where one or more *resolvable hosts* outside the grid, act as a *bridge* among different grid sites for receiving and sending data. We identify all the isles belonging to the same grid site as a *continent*. Within each continent there is only *one* WN that can communicate with the bridge: it is this isle that after each migration on its continent, swaps its new generation with that of a peer in another site, through the bridge.

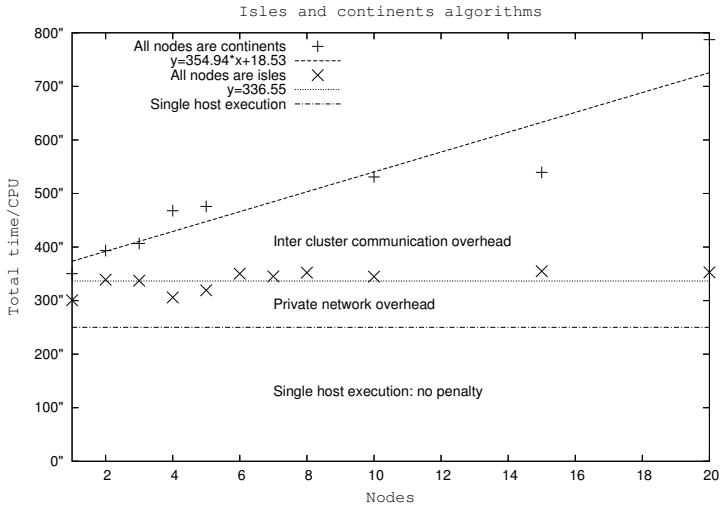

Figure 3:

This approach again introduces a network bottleneck (since the communication among continents is done on the public network and increases linearly with the number of sites), but it makes available a much larger number of resources. Moreover the reduced data transfer between different continents (we exchange just a population belonging to one isle) guarantees that the network delay will remain limited to a manageable level.

To test the performance of this strategy we carried out a simulation where all allocated nodes behaved as separated *continents*, exchanging data through the previously described mechanism. We compared it to the *isle* strategy, as well as to the serial one (see figure 3): a realistic simulation is very likely to behave in an intermediary fashion since only a few (10% or less) WNs will transfer data through the bridge host in the public network, while the remaining ones communicate in a high performance local network.

4. Conclusions

We took a stand alone application that run on a single processor computer and after several trials we were able to grid enable it. We overcame many difficulties bypassing the infrastructure's limits and we came up with essentially two approaches: a *master/slave paradigm*, and the *isles/continent idea*. Both approaches used advanced booking of resources to allow real time computations to take place.

The first one turned out to be non-performant because of the cheap computational cost of a *single* Kalman filter procedure; still, we feel that it could be used as a building block inside other algorithms with computationally more expensive evaluation functions. A very attractive feature is its dynamic nature which, unlike MPI, allows the addition of extra computing power on the fly as more WNs become available. Moreover our specific application allows such addition of extra power to occur *while* it is running.

The second approach was developed to allow both intra-site and extra-site communication. For direct internal communication, a foreign host such as the UI receives from all WNs their respec-

tive hostname; the complete list then gets broadcast back, allowing WNs to communicate directly among themselves. For external communication, a foreign host acts as a bridge. These important communication functionalities were achieved while maintaining the dynamic features of the first approach.

References

[1] www.egrid.it

[2] S.D'Addona and M. Ciprian, submitted to this conference