Real-time grid computing for financial applications

Riccardo Di Meo, Ezio Corso
EGRID project, ICTP

Stefano Cozzini
CNR-INFM/Democritos and Egrid/ICTP

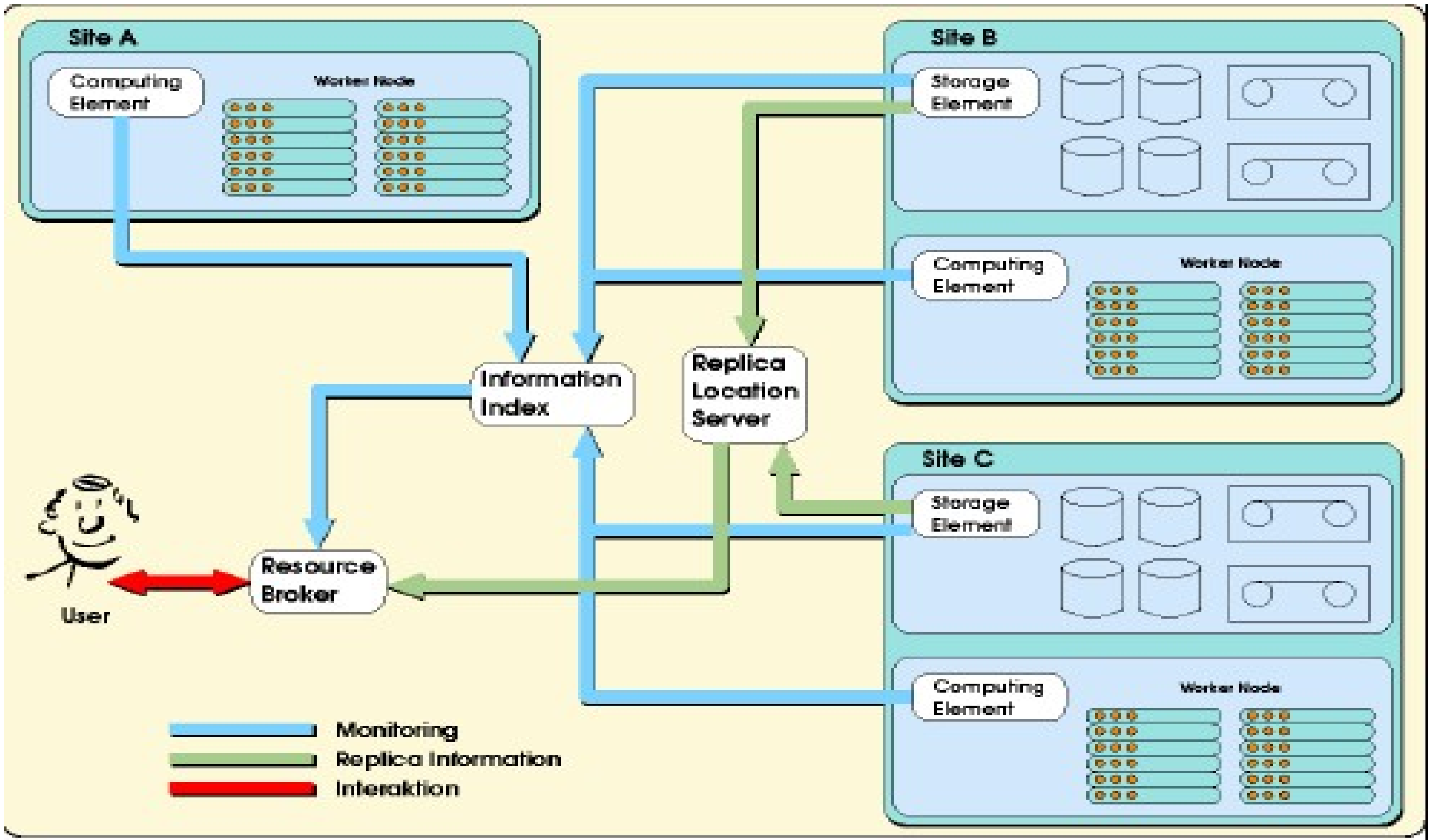
The EGRID infrastructure

- An italian national grid facility for finance
- The actual implementation (EGRID, through the LCG/EDG tools) is queue oriented and not real time: this is definitely an issue when we come to dynamic financial services.
- Our main goal is to attain real time response from both the grid and user applications.

How the job submission works

- The computer hosting the grid middleware (UI) contacts the Resource Broker (RB) and sends a job request.
- The RB searches the available resources and as soon as it finds an appropriate Computing Element (CE), it contacts it and forwards the request.
- The CE enqueues the task: when enough Worker Nodes (WN) become available, the program starts executing.
- The user, from the UI, checks the status of the submission: as soon as the program ends, he/she can download the computation results.

A graphical view



Drawbacks of this approach

- The queue approach is completely inadequate for real-time tasks: we don't know when our program will be executed.

  <u>The program should be ready to accept requests at stock market opening.</u>

- All steps, from submission to results retrieval, add a significant delay, which is unavoidable as long as standard tools are used.

  <u>The total time needed to submit a request and obtain an answer should be as small as possible (less than a minute).</u>

- The execution is not interactive: after sending the job to the RB, there's no way to alter it.

  <u>A single job should be able to process many requests.</u>

...and solutions.

- The queue approach is completely inadequate for real-time tasks: we don't know when our program will be executed.

We book resources in advance in order to have enough at a given time ("*Job reservation*").

- All steps, from submission to results retrieval, add a significant delay, which is unavoidable as long as standard tools are used.

We bypass the information system, obtaining status and results directly from the WN.

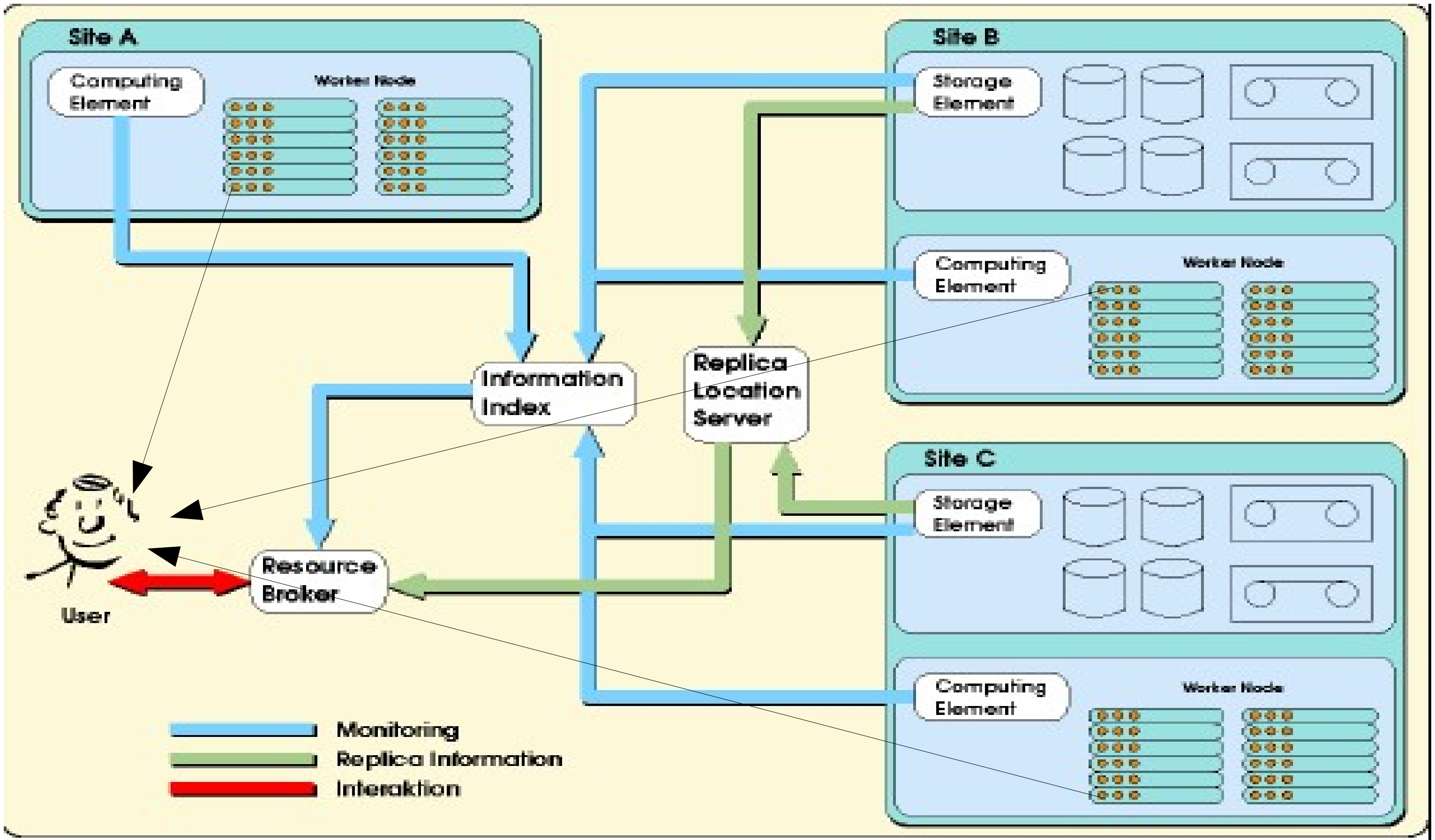
- The execution is not interactive: after sending the job to the RB, there's no way to alter it.

We establish a direct connection between WN and UI, thus letting them interact.

Job reservation

- We submit many requests in advance in order to have resources ready when needed.
- Once each job is running, it waits until the user has some data to process.
- No outside host (e.g. the UI) can establish a connection to the WN since they are on private networks: it is the WN itself that must poll periodically the host (which must be resolvable).
- On the UI there's a server program that accepts connections from the WN and sends computational requests to them.
- Once connected, every communication between WNs and UI bypasses the Grid infrastructure and takes place in real time.

A graphical view



Our test case

- A risk management application based on Genetic Algorithms (GA) and Kalman Filter (KF).
- The application takes the history of a set of assets and produces a forecast.
- The original implementation is serial.
- We focus on a specific set of parameters of the original application, which is used as reference when evaluating our optimizations.

Components description

Genetic Algorithm

- Evolutionary approach to optimization: improved new solutions are developed by crossing and mutating old ones.
- The key parameters are the mutation and crossing probabilities, the number of generations and the number of solutions to process in each generation (the "*genetic pool*").
- Only the latter two significantly affect (almost linearly) the simulation time cost.

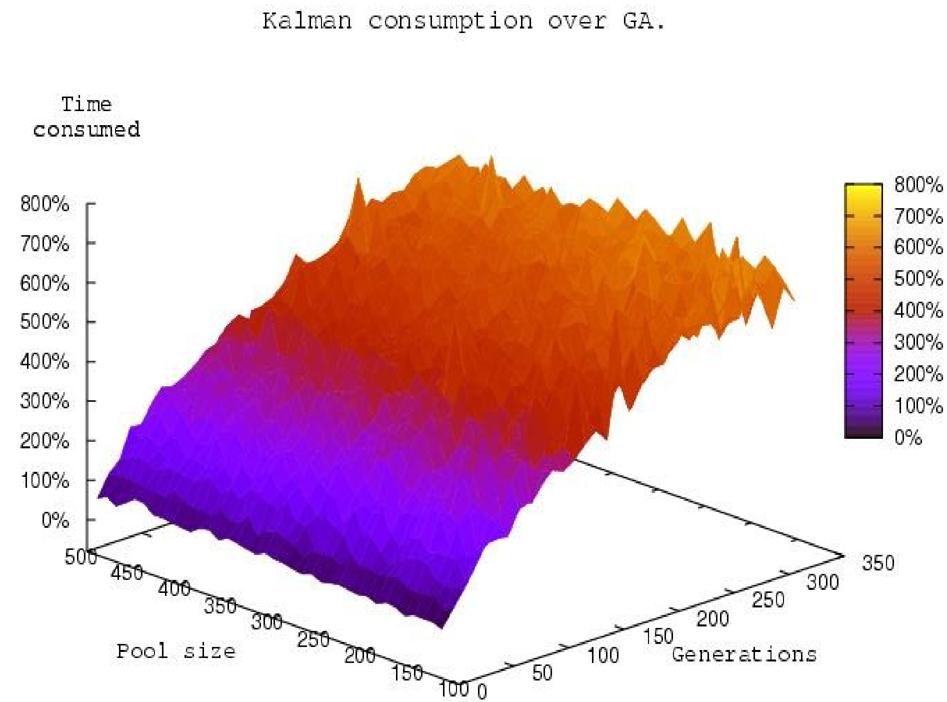
Kalman Filter

- A set of math. equations that describe the state of a system by providing a description of past and present states, and a forecast as well.
- It's computational cost is a complicated function of the data, but increases with the size of the input.
- In our case implementation, the number of assets and the number of past observations are the key parameters both for simulation accuracy and time cost.

Time cost of the program

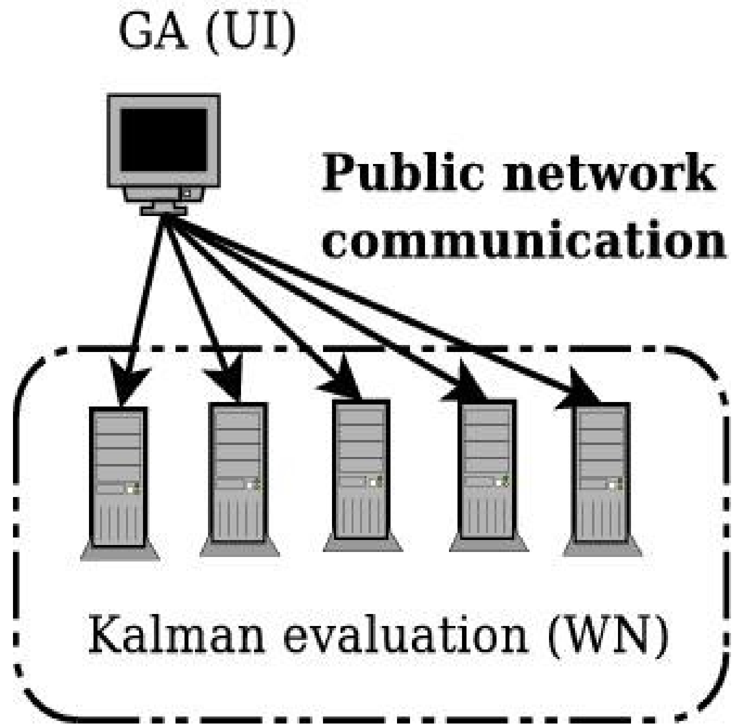
- Most of the time is spent inside the Kalman procedures.

KF is the main target to be optimized.



First implementation: a master/slaves solution.

- we separate the GA, which remains on the UI (the "*master*"), from the Kalman which takes place in the WN (the "*slaves*").
- Static Input data (the DB needed to evaluate the KF) is transferred at the beginning.
- The GA on the UI uses the WNs to evaluate the fitness of the solutions.



Pro

- Obtains the same results of the original implementation.
- Highly dynamic: new WNs are recruited as soon as they are available.

Cons

- Unbearable network overhead: copying data to/from UI is slow and scales badly with the size of the problem.
- **Increased** total simulation time!!!

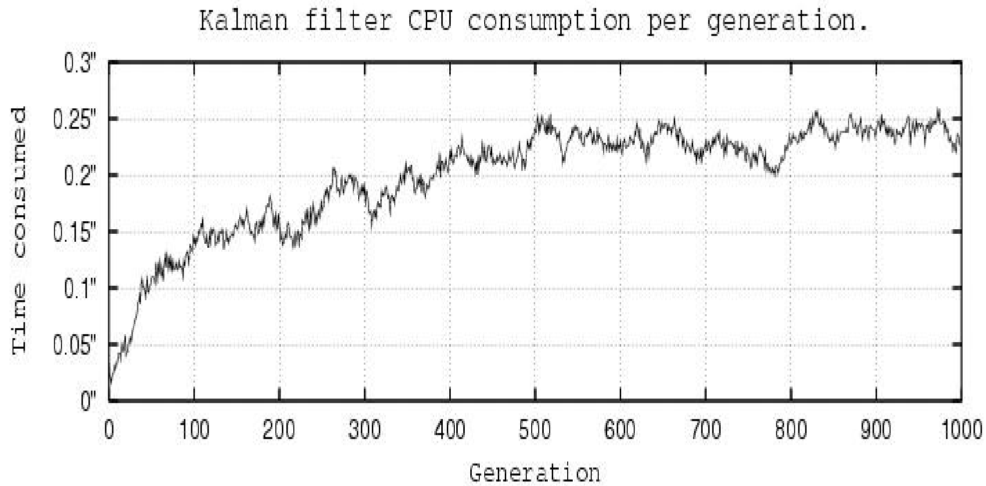
Clearly we need another approach...

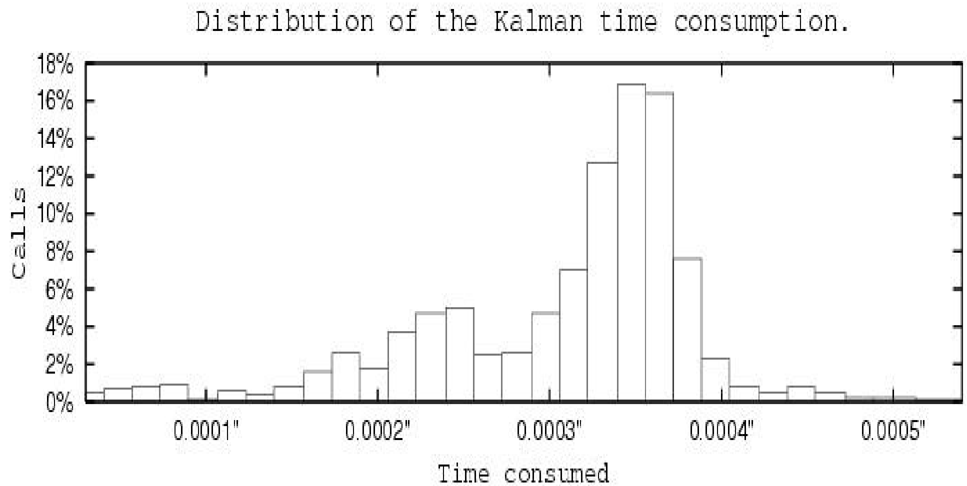
(though it can be reused in other programs)

A more in-depth analysis...

- Though the Kalman Filter contributes 90% of the cost of the simulation, it's overhead is distributed over a very large number of short KF evaluations (~600.000!).
- Global data exchanged: ~ 960 MB

- GA and KF **should reside on the same host**.
- Communication **should take place between WNs** in a private network.



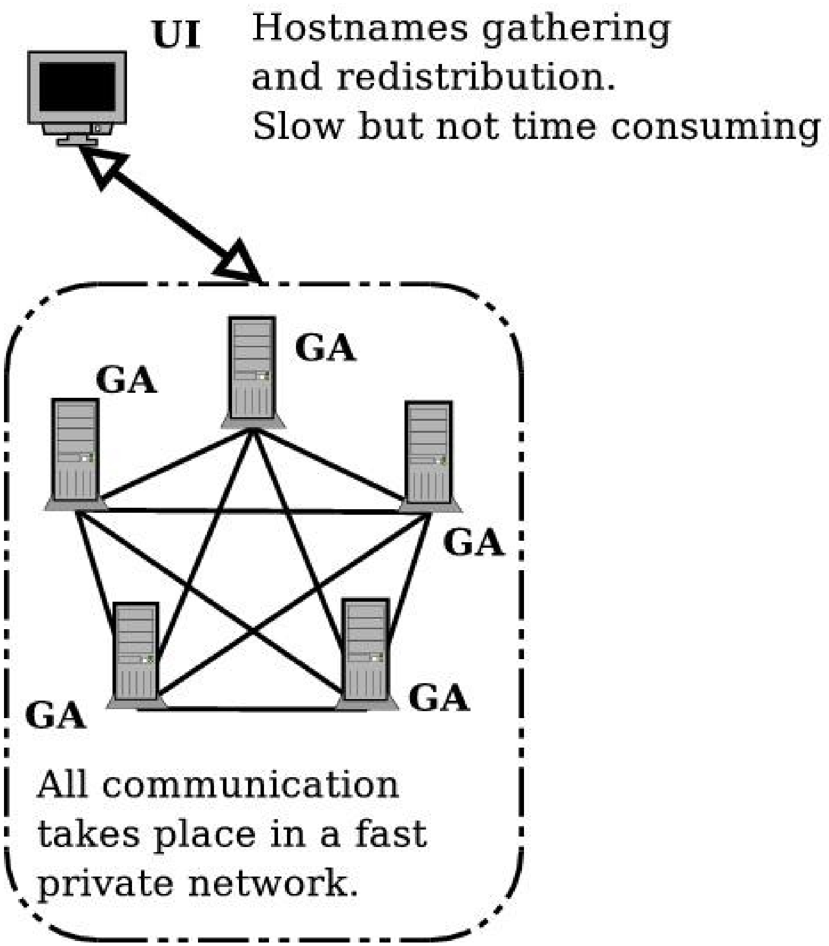


How can we make WNs communicate with each other?

- A first simple approach would be to use MPI but it:
  - limits our program to MPI enabled sites.
  - implies less/no dynamic WN management.
- Our approach:
  - submit a bunch of serial jobs and make each assigned WN aware of the other jobs..
- technique
  - UI collects WN hostnames as soon as they are recruited..
  - Before start any calculation UI propagates the full list to all the Wn recruited

A better approach: the isles algorithm

- The UI reserves N Worker Nodes and accepts their incoming connections.
- After enough WNs are connected to the UI (or a set timeout elapses), the WNs addresses are received and redistributed to enable intra cluster communication.
- A modified version of the original program is executed on N Worker Nodes ("*isles*").
- After a given number of generations the WNs exchange among them in a round robin way $\frac{(N-1)}{N}$ of their data ("*migration*").
- In the end, the best solution is selected.



Pro

- Greatly reduced communication between hosts.
- Almost all data transfer takes place in a fast private network.
- GA is parallelized too: performances of the algorithm doesn't depend anymore on the user's machine (which could be a **handheld**).
- Though the slowest WN constitutes a barrier to the execution, this drawback can be removed setting the migration and the simulation's end at a preset time.
- With the latter improvement a simulation ends **precisely** when the user wants to.

Cons

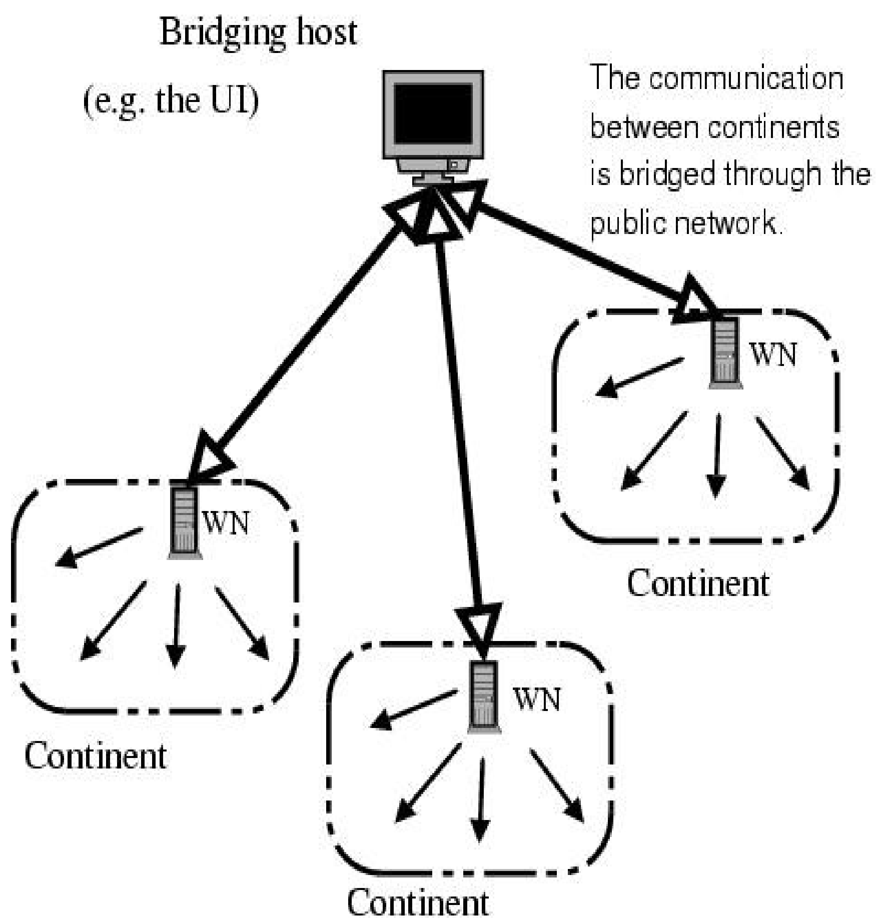
- We lost complete compatibility with the original program.
- Communication between different clusters is still unfeasible (as with MPI).
- Every GA works almost independently on its dataset for many generations, which *could* lead to worst results; on the other hand this approach reduces the GAs natural tendency to produce homogeneous solutions (thus *maybe* improving them!).

Enabling inter-cluster communication.

- WNs are shielded from the outside network:
  - they cannot be reached directly from the outside.
  - they **can connect** to other hosts, as long as their names are resolved (as seen in the first implementation).
  - this problem is **not solvable** by sharing a complete list of WNs' hostnames (as in the “isles” algorithm)!
- The only way to exchange data between CEs is to use one or more resolved hosts, which *accept* connections from WNs in different clusters, acting as “bridges”.

A further improvement: the "continents" algorithm

- Multiple copies of the "isles" run on *M* Computing Elements (which we will call "*continents*").
- In each one, a privileged WN is selected to carry out the communication with the bridge.
- After each migration $\frac{(M-1)}{M}$ of the WNs' data (which contain a mixed "sample" of all the solutions in the continent) is shared in a round robin way, in analogy with the "isles" algorithm.
- At the end of the simulation, the best solution is chosen.



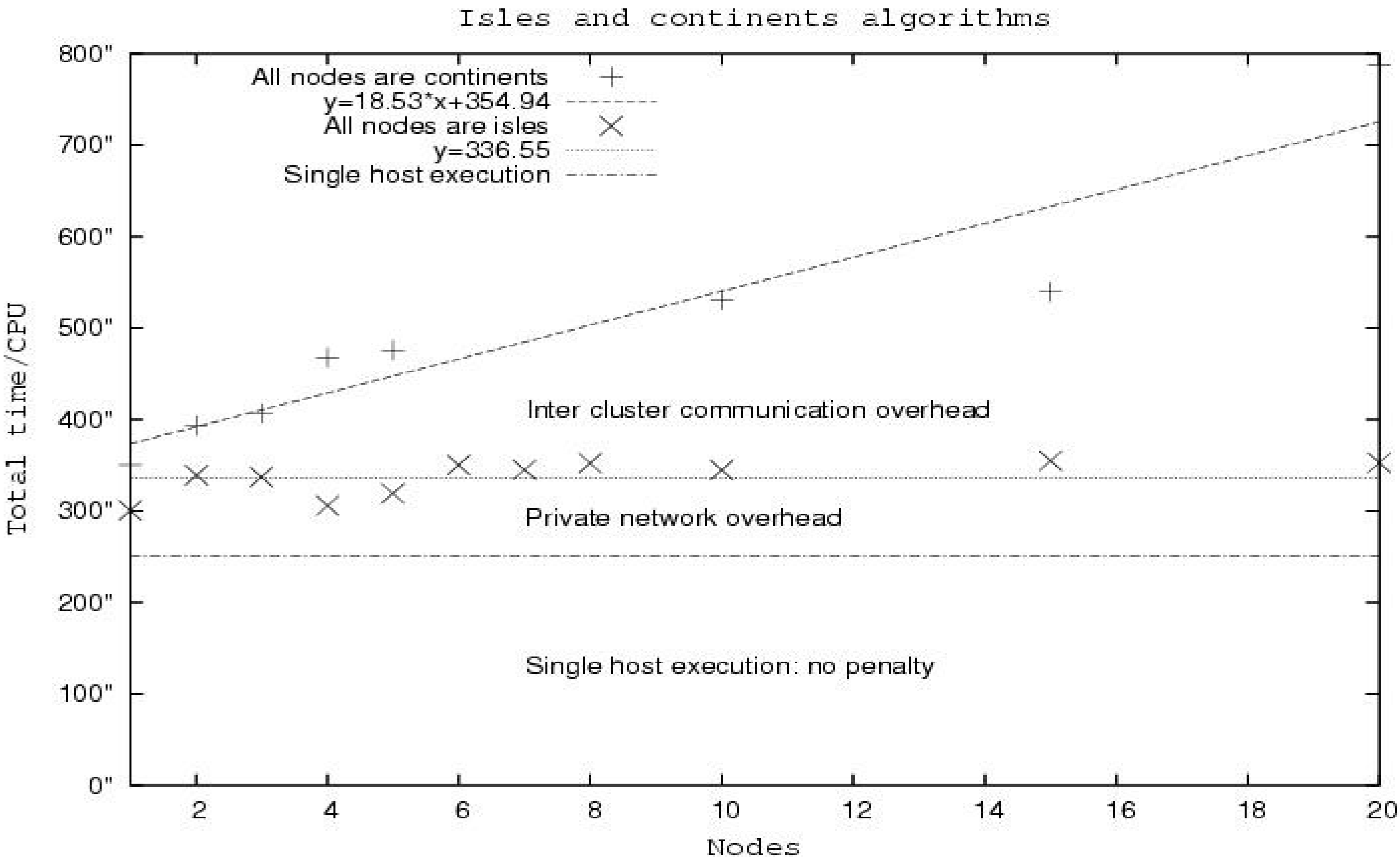
Pro

- Inter cluster communication can take place.
- Only a small quantity of data is transmitted trough the public network: most remains inside the CEs.
- More than one bridge can be used, and their placement carefully studied, to further reduce the network overhead.
- **Grid resources can be mixed with non-grid ones**.
- As long as the number of continents doesn't increase too much, network overhead remains on a manageable size.

Cons

- Communication through a bridge is twice as slow as that taking place directly between CEs, since packets travel through more routers.
- The fraction of data shared among different continents is even smaller than that among isles (although this *could* be advantageous).
- The WN that receives the information from the bridge uses it until the next migration, acting as a filter: the innermost isles will almost never receive the original data.
- Solutions propagate very slowly between continents.

Simulation results.



Performance considerations: master-slaves algorithm.

- Though not convenient for our case study, it could be applied proficiently to other scenarios where:
  - evaluation is very expensive.
  - solutions are described by less data or can be highly compressed.
  - the “master” node (the UI in our implementation) is moved to a WN in the same CE where the “slaves” reside, for faster communication.

Performance considerations: isles algorithm.

- The overhead is almost constant, no matter the number of WNs involved.
  - Good scalability.
  - A significant fraction of this slowdown is due to a synchronization barrier between WNs, which can be greatly reduced by modifying the program to operate on a time basis.
  - With the latter improvement even inhomogeneous clusters can carry out the simulation without additional costs.
  - The delay between migrations can be tailored for better performance *and* results (more frequent communication doesn't necessarily mean a better GA optimization!).

Performance considerations: continents algorithm.

- The overhead increases linearly with the number of continents.
  - Not a significant issue, since it is likely to be very small compared to the number of islands.
  - The same “time basis” strategy can be applied to this scenario too.
  - More than one bridge as well as a complex net topology can be used to reduce the communication cost.
  - Although inter CE communication is as good as it could be with the available resources, for an “economic oriented grid” the need of a bridging host could be removed (e.g. by opening some WNs to incoming connections).

Conclusions

- Though the grid infrastructure was developed with batch oriented applications in mind, this limit can be overcome:
  - fast response is possible through job reservation.
  - dynamic interaction between WNs and UI is feasible through a reversed client-server approach.
  - WNs can communicate with each other via hostnames gathering on the UI and redistribution.
  - inter cluster communication can be obtained through an external bridge.
- Although tailored to the optimization of our test case, all the above solutions can be adapted to a wide range of applications.
- The grid has the potential to become a key tool for economics, providing resources not only for academic simulations, but for “on the field” applications too.