# Grid Desktop Computing for Constructive Battlefield Simulation

Alejandro Juan Manuel Repetto

Ejército Argentino – DIDEP[1]/CIDESO[2] – Universidad del Ejército/Facultad de Ingeniería del Ejército/Escuela Superior Técnica
Av. Cabildo 15, Ciudad Autónoma de Buenos Aires, República Argentina
ajmrepetto@ejercito.mil.ar

**Abstract.** It is a fact that gaming technology is a state-of-the-art tool for military training, not only in low level simulations, e.g. flight training simulations, but also for strategic and tactical training. It is also a fact that users of this kind of technologies require increasingly more realistic representations of the real world. This functional reality threatens both hardware and software capabilities, making almost impossible to keep up with the requirements. Many optimizations have been performed over simulation algorithms in order to fulfill these needs; no definitive solution, however, has yet been achieved. The question that arises naturally is, then: Does any generic global solution to the problem of the uneven growth of the computational power requirements with respect to the available capacities exist? This paper presents the problem by describing a real situation, analyzing the Batalla Virtual[3] case of study and, in answering the motivating question, it proposes a potential software architecture employing a grid desktop computing (GDC) framework to empower constructive simulation systems, pulling off an adaptive hardware infrastructure. Additionally, as the constructive simulation scenarios do not fully adapt to the market-available GDC frameworks, the solution recommends suitable modifications to these software.

**Keyworkds:** Grid Desktop Computing; Distributed Computing; Constructive Simulation; Battlefield Simulation.

## 1 Introduction

Distributed computing solutions are being applied mostly in scientific problems. However, over the last years, this technology has exceeded this boundary and conventional computing problems are been solved with it, as shown in [1].
There exists a wide span of distributed computing architectures. In particular, grid desktop computing (GDC) architecture is becoming progressively more popular for

[1] DIDEP: Office of Development, Research and Production, Argentine Army.
[2] CIDESO: Center of Software Research and Development, Argentine Army.
[3] Batalla Virtual (BV): family of constructive simulation software of the Argentine Army for training

its easiness of implementation and its unbeatable economic advantages. However, it is yet an upcoming technology that has some dangling problems that must be taken into account.
The computational power of GDC architectures and the low costs have brought an innumerable amount of applications in the military field, including cryptanalysis, decision making and image analysis software, among others.
To keep up with advance of the computer science, the Argentine Army through the CIDESO decided to investigate this technology with an innovative point of view: distributed constructive interactive simulation software.
In the present work the problem that the CIDESO wanted to tackle down using GDC will be presented. Then, grid desktop computing, describing its components and different implementations, will be illustrated. And, finally, the solution achieved will be described.

## 2 The CIDESO's Problem

The main products of the CIDESO are simulation programs. It develops constructive simulation software for military training; one of its main products is *Batalla Virtual* (BV). It uses a customizable set of simulation models in order to emulate different portions of reality.
BV immerses the players in a virtual scenario, setting up an operative military problem that they have to solve putting in practice their theoretical knowledge about doctrine. One game engages several people, considering the variety of roles involved in a real military operation. Each participant is in charge of commanding one element or playing the role of advisor (staff) of the commander.

### 2.1 Batalla Virtual's Challenges

BV was born as an upper-intermediate command and control training system, it simulates up to company level. The quantity of players involved in a game hardly passes the couple of hundreds. However, as the users get more and more involved with the product, the necessity of increasing the level of detail is growing. This requirement is leading not only to enable new simulation models in order to perform more meticulous actions but also to modify and adapt the already existing models for supporting more players interacting among them.
The *scale problem* made the CIDESO's engineers to rethink and redirect the software development efforts, realizing that the computational power needed to play a simulation game was exceeding the available capacities.
The most processor time consuming model detected is *DLI* (*Detection, Localization and Identification*). DLI is the vision simulation model which calculates the line of sight for each player. Basically, it is in charge of showing to each user only the elements of the game that the user is able to see considering his position, the position of all the other elements, the topography of the terrain and the exploring devices of the element. The only way of doing so is by try-and-error strategy. That is, for each

element, for each exploring device, it is necessary to check if each objective is visible or not considering the device properties and the topography.
The complexity of this problem is proportional to $O(n^2-n)$, being $n$ the amount of elements in the game. Moreover, for each pair of elements that are been checked, every point of the map that relies over the line which connects both elements must be analyzed in order to see if there is no geographical interference between them. This search has to be exhaustive, making the necessary processor time really high.

### 2.2 The Solution Space

Three possible paths were considered in order to solve the problem: code refactoring, code rewriting and architecture reengineering.
The *code refactoring* consists on reengineering the source code in order to mathematically and/or computationally optimize the algorithms. When possible, doing this is an economical solution to performance problems.
Another - more radical - solution was to *rewrite the code* in other language or for another platform. One plausible way out was to rewrite part of the problematic code in other more-performing language, such as C or C++.
The last option was to apply an *architectural change*. The basic idea was to introduce distributed computing facilities for the problematic models and integrate them with the already functioning architecture.
Even though the code refactoring was performed and considerable optimizations have been achieved, they were not enough in order to be prepared to increasing the level of reality of the simulation. The performance was more than acceptable for large games, but large in the nowadays scale. Consequently, the problem has been momentarily fixed; however it has not been definitely solved.
In order to propose a definitive solution, the other two proposals have been analyzed and joined. The intention was to deploy GDC architecture where the same machines that are playing the game act as processing units of simulation, developing an adaptive processing infrastructure, proportional to the size of the game.
The key concept of the plan was: *if the processing requirement is proportional to the size of the game, and the size of the game is proportional to the number of clients connected, distributing the workload over the same connected clients produces an auto sufficient computing infrastructure*.

## 3 The Grid

### 3.1 Introduction to the Grid

Although Grid computing systems started to be studied in the eighties, as described in [1], in recent years, because of the continuous increase in the computational needs and the limitations in rising the computational power with the same speed, they have attracted the attention of both the commercial enterprises and the research institutions. The confluence of a set of factors have driven to the idea of what is called *third generation* Grid Computing. Among them we can mention the overcome of new

computational paradigms, *e.g.* service oriented architectures, the *availability of powerful under-utilized desktop computers* and their *interconnection* through fast high-bandwidth networks.
Buyya et al., in [2], summarize Grid computing as a dynamic network of heterogeneous computing resources which cooperate building a uniform computing environment. A more detailed definition of the Grid has been given by Ian Foster in [3], who combines three fundamental concepts of this architecture: *heterogeneity*, *transparency* (also known as virtualization) and *performance*.
The first concept refers to the major challenge that this architecture has to face. The Grid must consider a *wide range of computing and communication technologies*. Grid infrastructures should support different computer architectures, different operative systems, different means of communication and different data sources, everything *working in a synergic way*.
*Transparency* is also a must in Grid computing. Taking into account the heterogeneity, the final user cannot be charged with the responsibility of managing all the technologies in a separate way. From the user sight, the Grid should be seen as a conventional computer program that receives an input and returns an output, without even knowing if the intermediate process has been executed by a single machine or a cloud of thousands. Thanks to the lasts advances in web technologies and the arisen of *Service Oriented Architectures* – SOA–, this challenge has been worked out, as explained in [4].
Finally, the third concept refers to the crucial objective of the Grid. The major benefit of this technology is the *unbeatable performance* that can be reached in a completely *economical way*. The Grid has the ability of exploiting both dedicated and shared already-acquired underutilized resources.
Beside these three characteristics, there are others interesting points to highlight with respect to the benefits of the Grid. Among them, the *resource balancing* capabilities, as the Grid is seen as a whole by the users, the Grid system manager can balance the utilization of each resource in order to avoid bottlenecks; and the *increase in the reliability* of the processes, the lack of the single point of failure due to the distribution makes Grid systems more reliable than centralized systems.

### 3.2 Grid Components

The components of the Grid can be summarized in *Grid users*, *jobs, Grid resources*, *communications facilities* and *Grid infrastructure*:

- *Grid users* are the users (or applications) who submit jobs to the Grid.
- *Jobs* are the requests that the users do to the system. These jobs will be split up in tasks, which are the atomic units of instructions.
- *The Grid resources* are all the subsystems which give the computational power to the Grid. In [3] the resources are classified in *computational resources* and *storage resources.*
- *The communication,* are routers, switches, fiber optics, satellites or any other type of communication device that the Grid will use in order to connect all the resources among them.

– *The Grid infrastructure* is composed by all the *ad-hoc* equips and services, declared by the *Open Grid Services Architecture* – OGSA-, which will manage the Grid itself, see Fig. 1. The most important components, described in [5], are:
  - the *scheduler*, in charge of assign the tasks to the resources,
  - the *resource allocation manager*, which provides remote execution capabilities and status management;
  - the *monitoring and discovery service*, which obtains the information about the resources and maintain the service directory up to date;
  - the *data sharing module*, in charge of providing the datasets to the resources and exchange information among them, and
  - the *security module*, which provides the authentication, authorization and accounting (AAA) service to the structure.

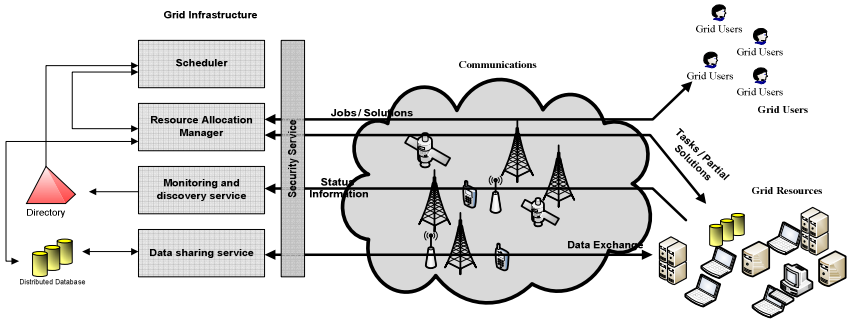


**Fig. 1.** Grid Architecture.

### 3.3 Grids' Classification

One possible classification of GDC divides these systems in two big sets: *intragrid* and *intergrid* [5]. The first are the ones which their bounds are in the intranet of an organization. This is the typical scenario of enterprises which exploit the computational power of their idle resources to run their own business processes. However, with the geographically expansion of the network and the requirements of collaboration among organizations, the Grid can cross the intranet boundaries and become an intergrid. This is a typical case of laboratories collaborating in a specific research. The difference between the intergrids and intragrids is that the firsts uses public communication resources to empower the computational capacity.
From another point of view, the Grid can be divided in *volunteer* and *not volunteer* systems. The volunteer systems are the ones which the user decides to share resources with the Grid. She decides which resources to share, how to share them and for which project. These systems are public, *e.g. BOINC* projects [6]. On the other side, there are systems in which an administrator is in charge of assigning portions of computational power of the resource to the Grid. This last class can be applied in closed systems.

### 3.4 Existing Grid Computing Infrastructures

Two grid computing architectures are the most well known in the market: *The Globus Toolkit,* developed by *The Globus Alliance,* and *BOINC*, developed by *Berkeley University*. Both are GDC infrastructures but with differences in what referrers to the profile of the target user.
Globus is a set of independent grid computing tools that can be used together or inserted into other grid computing system in order to perform any of the grid tasks. All these tools are *community-based*, *open-architecture* and *open-source* and they were developed by *The Globus Alliance*, an alliance of universities, laboratories and government U.S. agencies [7]. It is working in several laboratories in highly heterogeneous environments. Principally, it is used for solving scientific problems which require large amounts of computational time in order to obtain results, e.g. in [8] [9].
In contrast, *BOINC,* detailed in [10], is oriented to open *volunteer* projects where the resources are given by Internet users. It is an *open-source* infrastructure which gives to the users the possibility of creating projects over it. The projects can be either close or open. The close are those where the donators are in a close set of possible volunteers while the open are the ones submitted to the *BOINC* servers and any Internet user can voluntarily donate part of she's own computational time to the project. The main difference between *BOINC* and *Globus* is that the first can also work in open environments due to its special *validation module* which checks the results given by the volunteers in order to certify its accuracy. This infrastructure is well known for been the first volunteer grid computing approach and for building the most powerful computer facility ever with the project *SETI@HOME*, described [6].

### 3.5 Framework Selection

*BOINC* has been selected as target architecture principally because of scheduling problem. Without any previous experience in implementation of distributed solutions, with low knowledge of UNIX platform and with only two resources (one senior engineer and one assistant), the project should be delivered six months. So, after analyzing both products, we decided that *BOINC* was enough easy-to-use and have a more active supporting community which permit us to fulfill our objective in time.

## 4 The solution

### 4.1 Architecture

The proposed architecture is relatively simple, see Fig. 2. As the simulation models in *BV* are conveniently packaged, the only change that must be introduce is a control before the execution of the model to decide whether to process the simulation internally or to send the work to the distributed infrastructure. If the second option is chosen, then it has to wait for the global result and continue processing.

When the platform receives the job, the *work-creator* program splits it in several atomic tasks and sends them to the *BOINC* scheduler. The scheduler builds up the execution program and the clients (*BV* clients plus, eventually, other volunteers) starts pulling the jobs, processes them and returns them to the *BOINC* server.

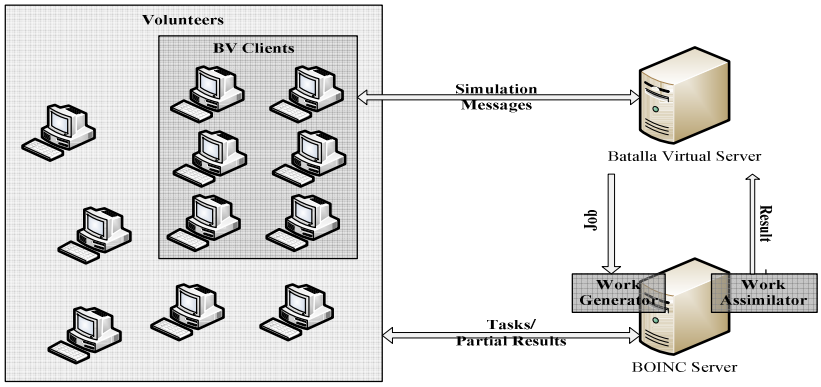


**Fig. 2.** BOINC-BV Architecture

As the server receives the partial results, the *work-assimilator* joins them in a single global result. When all the partial results are joined, the work-assimilator returns it via TCP/IP to the *BV* server.

### 4.2 Algorithm distribution

The target algorithm for the beta deployment was DLI. As it has been explained, DLI exhaustively examines the set of elements seen by all the participants of the game. Therefore, if there are five players deployed in the game, the algorithm for checking whether one element sees another is executed twenty times ($5^2-5 = 20$). It is important to highlight that each execution of the algorithm is absolutely independent from the others, making it possible to parallelize the overall execution.
The algorithm to check the visibility of one point from another point of the map was rewritten in ANSI C, and deployed over the *BOINC* clients. Each atomic task contains a set of pairs of points that must be analyzed. So, the volunteer performs the analysis and return the single result to the *BOINC* server.

### 4.3 Adapting *BOINC* to the CIDESO's Problem

However *BOINC* is one of the most spread grid desktop computing frameworks, it does not fully adapt to the BV requirements. It was actually thought to solve other kind of problems. Two main challenges have been faced: *the pull strategy* and the *short-running processes*.
The fact of being thought for intergrid scenarios makes *BOINC's* client to work in pull mode. The client asks for tasks to the server when his computer conditions reaches a set of predefined preferences.

For the objective of DLI, the ideal situation was the opposed. A push-like strategy fits better than a pull strategy. Nevertheless, through simple configurations, it is possible to preset the pulling time. Thus, reducing this parameter to the minimum (1 second) we functionally change the strategy making the client to continuously ask the server for new jobs, if there are no jobs to be processed, nothing is sent.
With respect to the short-running processes, the problem is more complex. Most grid desktop computing infrastructures are meant for processing long-running processes, the result is not expected to be in quasi-on-line fashion. Yet, BV, for being an interactive system, requires immediate results.
In order to optimize the distribution circuit, the overheads introduced by the distribution platform have been analyzed. Three critical points have been observed:

1. *Communication overhead:* the time used for transmitting the job to the platform, the time for transmitting the tasks to the volunteers, the time for returning the partial results to the platform and the time for transmitting the final result to BV server, were optimized by using raw TCP/IP communication, avoiding any kind of web services and XML parsing.
2. *Job splitting overhead:* the platform receives the entire job that must be split in atomic tasks to be processed. This software has been written in ANSI C, using low level instructions in order to minimize the splitting time.
3. *Job assimilation overhead:* this overhead is related with the reception of the partial results and their integration. It was finally reduced to a simple task sender program. The shape of the partial results was modified in order to be able to directly send them to BV without any summarization.

These three overheads have a peculiarity: the first is almost constant, no matter the amount of jobs submitted, the time expected will be roughly the same; and the two others are linearly related with the amount of elements of the game. This makes the overhead time to grow slower to the processing time in a non-distributed strategy.

Although these overhead have been optimized, there exists an intrinsic limit for which the distributed solution does not increase the performance of the overall system but harms it. As Fig. 3 shows, the equilibrium point, where both strategies perform equally, must be found in order to decide whether to activate or not the distribution.

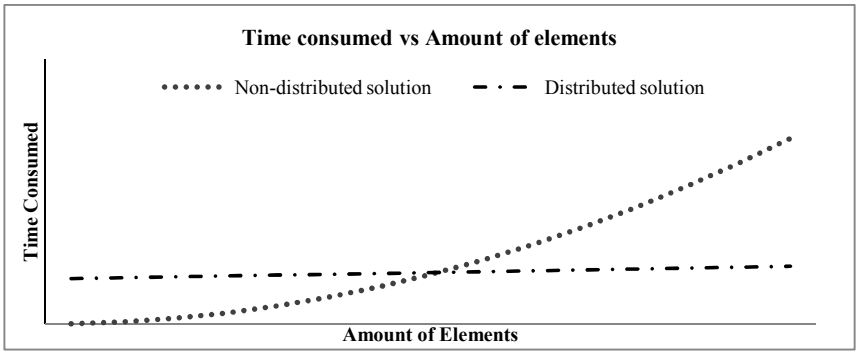


**Fig. 3.** Time Consumed vs. Amount of Elements.

Surpassing the equilibrium point it will be convenient to use the distributed platform. Before that point it will be convenient to use the non-distributed solution. Inside BV will reside a decision algorithm that will chose whether to send the job outside the simulation server or process it inside.

## 5 Conclusions

The main contribution of this article is the *new application for grid desktop computing* that has been presented. Analyzing the prior works over this field of computing engineering, no previous applications which use GDC infrastructures for short-running process applied to interactive computing have been found.
Moreover, the *challenges that have been faced* in order to adapt this infrastructure to a non-conventional problem have been described and analyzed, showing the solutions applied in a real case of study. These challenged also put in evidence the possible limitations in the use of this technology.
From the CIDESO's point of view, the introduction of GDC technology in its applications enabled a new world of research and *moved the boundary* of its products *in which computer performance respects*. This research also permitted to develop *new skills* widening the knowledge of the laboratory.
In conclusion, the experience performed with GDC over constructive simulation models contributed not only inside the laboratory, enabling new technologies, but also outside, opening a new possible line of research over this tool.

## 6 Further work

Particularly, in what *BV* case of study respects, two main works will be preformed:

- *Benchmark with other infrastructures*: as it has been highlighted, the *Globus Toolkit* was also analyzed as a possible solution for the presented problem. This infrastructure will be set up and compared with the *BOINC* implementation, looking for pros and cons of both architectures trying to generate tools for deciding for which problems performs better each platform.
- *Extend to other algorithms*: other algorithms of BV will be implemented in a distributed manner with the objective of reducing the overall simulation time.

Overcoming the CIDESO's boundary, two modifications to *BOINC* will be analyzed:

- *Push mode client*: taking advantage of the openness in the source-code, an adaptation in the *BOINC* strategy will be analyzed, trying to change from pull to push mode client.
- *Extension for mobile computing*: having applied *BOINC* infrastructure for short running processes, a mobile client can extend the power of the overall infrastructure by adding other processing units such as handhelds or smart phones.

## 7 Bibliography

1. Roure, D., Baker, M.A., Jennings, N.R, Shadbolt, N.R.: The evolution of the grid. In Grid computing - making the global infrastructure a reality. John Wiley and Sons Ltd (2003)
2. Rajkumar Buyya, David Abramson, Jonathan Giddy and Heinz Stockinger: Economic models for resource management and scheduling in grid computing. (2002)
3. Foster, Ian: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In : In Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing. (2001)
4. Carpenter, Brian: Grid Computing. (2003)
5. Ferreira, Luis, Berstis, Viktors, Armstrong, Jonathan, Kendzierski, Mike, Neukoetter, Andreas, MasanobuTakagi, Bing-Wo, Richard, Amir, Adeeb, Murakawa, Ryo, Hernandez, Olegario, Magowan, James, Bieberstein, Norbert: Introduction to grid computing with Globus. IBM Red BOOKS (2003)
6. Anderson, David, Cobb, Jeff, Korpela, Eric, Lebofsky, Matt, Werthimer., Dan: SETI@home: an experiment in public-resource computing. Communications of the ACM 45(11) (2002)
7. Foster, Ian: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Globus Project (2002)
8. Stef-Praun, T., Clifford, B., Foster, I., Hasson, U., Hategan, M., Small, S., Wilde, Zhao, M, Y.: Accelerating Medical Research using the Swift Workflow System Health Grid. (2007)
9. Nefedova, V., Jacob, R., Foster, I., Liu, Z., Liu, Y., Deelman, E., Mehta, G, Su, M., Vahi., K.: Automating Climate Science: Large Ensemble Simulations on the TeraGrid with the GriPhyN Virtual Data System. In : In Proceedings of the Second IEEE International Conference on e-Science and Grid Computing. IEEE (2006)
10. Anderson, David: BOINC: A System for Public-Resource Computing and Storage. In : In Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing. IEEE (2004)