Hybrid Architecture for Constructive Interactive Simulation: Evaluation and Outcomes

Alejandro J. M. Repetto
CIDESO, DIDEP - EST, IESE – Ejército Argentino
Ciudad Autónoma de Buenos Aires, Argentina
ajmrepetto@gmail.com

ABSTRACT

The rapid increase in requirements for computational power to meet users' functional needs in Live, Virtual and Constructive (LVC) environments challenges the available hardware capabilities of organizations. Particularly, constructive simulation is still a widespread tool for strategic and tactical company level training. The vast benefits of this technology (e.g. didactic, economic, safety) raise new requirements for accurate emulations of reality. These requirements challenge not only the skills of mathematical modelers, who need to develop new simulation algorithms with a higher degree of realism and better performance levels, but also the hardware capabilities of the organization to match the increasing computational demand. Although hardware costs have become more accessible during these past years, other issues, such as storage, heating or performance, might turn the problem unmanageable, exceeding a simple cost benefit analysis.

The present work reports the results of merging a Grid Desktop Computing Platform (GDCP) with conventional client/server design, introducing a hybrid architecture to address this issue. First, a technical description of the problem, depicting challenges and risks, is outlined. In this setting, a generic framework for any constructive simulation system running over client/server architecture is developed. Second, a full functional architecture, using BOINC (Berkeley Open Infrastructure for Network Computing) as GDCP to distribute the workload of simulation algorithms on-demand among the available computing infrastructure, is described. Third, the results of performance tests are shown and analyzed, comparing the hybrid with a previous architecture. Such tests reveal a performance optimization of more than 400%. Finally, lessons learned and architecture limitations are evaluated.

ABOUT THE AUTHORS

Alejandro J. M. Repetto, M.Sc, is a Team Leader of the research staff at the CIDESO (Software Research and Development Office of the Argentinean Army) since 2006. He received his first M.Sc with honors in Information Systems Engineering from the Escuela Superior Técnica del Ejército Argentino, Buenos Aires, Argentina in 2004, where he also completed a specialization post-graduate program in Cryptography and Computing Systems Security in 2007, and his second MSc. in Computer Systems Engineering from the Politecnico di Milano, Italy in 2008. He is alumni at the Alta Scuola Politecnica, a school for young talents at the Politecnico di Milano and Politecnico di Torino, Italy. He acted as system security engineer and as consultant in software engineering for the private area, always maintaining a focus on innovation. He acts as associate professor at the simulation department of the Escuela Superior Técnica del Ejército Argentino. His research focuses include grid computing, algorithms optimization and simulation modeling.

Hybrid Architecture for Constructive Interactive Simulation: Evaluation and Outcomes

Alejandro J. M. Repetto
CIDESO, DIDEP - EST, IESE – Ejército Argentino
Ciudad Autónoma de Buenos Aires, Argentina
ajmrepetto@gmail.com

INTRODUCTION

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 technology require increasingly more realistic representations of the real world. This functional reality threatens both hardware and software capabilities, making it almost impossible to keep up with the requirements needed. 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?

Client/server architecture is the straightforward design for constructive simulation. A group of users, trainees and trainers, connect to one or more centralized servers which execute the simulation models. The users perform actions over the simulation state and the servers calculate the new situation based on mathematical models. This way of thinking succeeds while the number of users and the models are stable. If these two factors are maintained between acceptable limits, the sizing of the servers is direct. Nevertheless, this situation seldom happens; LVC products are always-evolving systems: the amount of users grows in time, the models are improved looking for a higher level of detail and new models are added to the same environment continuously. These three issues make the computing infrastructure grow in order to give response to the new demands; yet, the growth's speed is slower than the required one.

The striking point in client/server architecture is that, while the servers are processing the simulation algorithms, the clients are idle, using none or little computational power. Moreover, incremental tests done over simulation systems show that the computational requirements increase quadratic with respect to the number of users connected to the simulation scenario. This is, basically, due to the interaction between elements during the simulation.

The hypothesis, thus, is: if the computational requirement is proportional to the size of the game, and the size of the game is proportional to the quantity of clients connected, distributing the workload over the same connected clients produces a self-sufficient computing infrastructure.

GENERIC FRAMEWORK

Grid Desktop Computing Background

Distributed computing solutions are being applied mostly in scientific problems. However, over the last years, this technology has exceeded this boundary and every day new conventional computing problems are being solved with it (Roure, Baker, Jennings, & Shadbolt, 2003). There exists a wide span of distributed computing architectures: from simple clusters to complex peer to peer networks, passing through grid computing solutions. This last architecture is becoming progressively more popular for its easiness of implementation and its unbeatable economic advantages. The availability of powerful under-utilized desktop computers, the service oriented architecture and the high-bandwidth networks, have lead to the conception of the third generation Grid Computing. A Grid system is a dynamic network of heterogeneous computing resources which cooperate building a uniform computing environment (Rajkumar Buyya, David Abramson, Jonathan Giddy and Heinz Stockinger, 2002), that is, in summary, the concept we were willing to implement.

In Grid Desktop Computing Platforms (GDCP) pooled resources are used to achieve high throughput on a set of computational jobs. Such systems allow large numbers of machines to be added as a single resource in a higher-level grid system, achieving significant benefits in reduced management effort and grid complexity, (Constantinescu-Fülöp, 2008). There exist two fashions of GDCP: intragrid and intergrid (Ferreira,

et al., 2003). The former is confined to an institutional boundary, where the spare processing capacity of an enterprise's desktop PCs is used to support the execution of the enterprise's applications; implementations of this technology are Entropia (Chien, Calder, Elbert, & Bhatia, 2003) and Condor (Condor Project, 2010).

The later is an arrangement in which volunteers provide computing resources to projects, which use the resources to do distributed computing and/or storage. Volunteers are typically members of the general public who own Internet-connected PCs. Two main aspects are worth noting: the volunteers are anonymous thus they are not accountable. The anonymity of the volunteers requires the platform to provide extra security tools in order to guarantee the correctness of the results obtained. The most widespread intergrid platform is Berkeley Open Infrastructure for Network Computing (BOINC), detailed in (Anderson, 2004).

To sum up, GDCPs deal with highly heterogeneous computer infrastructure (from both, hardware and software perspective) with an enormous amount of subsystems in order to obtain a virtual high throughput super computer. The most recent measurement of BOINC, in April 2010, threw an average of 5.1 PFLOPs (BOINC, 2010).

The internal GDCPs' architecture is, basically, the same for all of them. The key components are: the resource scheduler, the job manager and the node manager. The resource scheduler is in charge of matching a job with the most appropriate grid's node and schedules its execution; this is a critic task in a heterogeneous grid taking into account the irregular hardware and software capabilities of the participants. The job manager is responsible of receiving the job from the grid users and administering it, i.e. dividing it in smaller units, classifying it by its computational requirements, controlling the execution flow, joining the partial results and returning the final outcome to the user. Finally, the node manager is in charge of the node validation and node tracking maintaining an updated record of the whole grid (see Figure 1).

Additionally, on the boundary of the GDCP, there is the volunteer software which is, actually, the software that performs the calculus. The clients' architecture in GDCP varies with implementation: there are some self-contained software which communicate over TCP/IP using closed protocols, e.g. BOINC clients, and others which use open interfaces like Web Services for exchanging information between the clients and the server, e.g. Globus Toolkit. It is important to underline that the design of the clients' architecture will define the development of the distributed algorithm; nevertheless, most GDCPs offer APIs for facilitating its implementation.

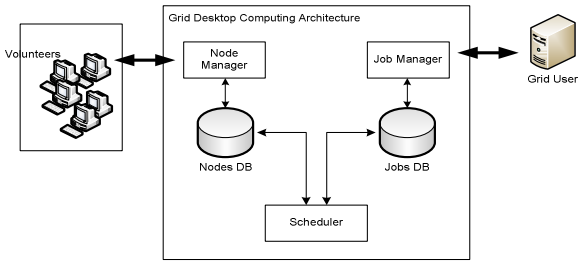

Figure 1. GDCP Architecture.

The Solution

GDCP begin with a collection of computing resources – heterogeneous in hardware and software configuration, distributed throughout a corporate network or over open environments and subject to varied management and use regimens – and aggregate them into an easily manageable and usable single resource. Furthermore, a desktop grid system must be unobtrusive, i.e. it must use the resources in a mode that ensures there is little or no detectable impact on the use of the computing resources for other purposes (Foster & Kesselman, 2004).

Considering that the objective of the present work is to distribute the workload of the simulation algorithms over the users of a client/server architecture and, possibly, over other donators, in a transparent way (i.e. without interfering with the user) exploiting the idle computational capabilities, the solution proposed is to introduce a GDCP into the architecture of a conventional client/server providing the capacity of dynamically activate/deactivate the workload distribution, building what we called Hybrid Architecture for Constructive Interactive Simulation (HACIS).

The classic architecture of the monolithic simulation systems can be described as a compound of models which, given a vector that describes a situation, performs a series of calculus and returns a new state vector. The majority of simulation systems use their models in a pipeline, i.e. executing one model after the other, relying on a central manager to coordinate execution. Decoupling the models in such systems is straightforward. It is sufficient to determine the entry point to the model, defining the interface and forking the execution pipe to another system, then returning the

control to the main thread and continuing the simulation process. The interfaces between the server and the users (i.e. players) remain without changes.

Joining both architectures results in a HACIS where the conflictive models can be decoupled and sent to a GDCP in order to exploit the spare capacities of the players' computers. The main, and unique, modifications that must be performed over the monolithic simulation system is the decoupling of the model and a forking intelligence subsystem which permits the execution controller to decide whether to use the GDCP or not depending on the workload. In addition, the decoupled model must have been rewritten in terms of the GDCP APIs (Application Programming Interface), and the platform must be configured for the application, see Figure 2.

The point of input/output of the GDCP is the job manager. It receives the job and returns the final result. Therefore, this will be the point of contact between the client/server architecture and the GDCP.

Under this concept, the former client/server architecture acts a GDCP consumer exploiting both the simulation's users and other possible donators. Furthermore, the same simulation server can be configured as a volunteer inside the execution cloud considering that, because of the pipeline architecture of the simulation systems, while the infrastructure is processing the distributed algorithm, the server is idle. Therefore, the introduction of a GDCP results in an efficient exploitation of the overall computing infrastructure.

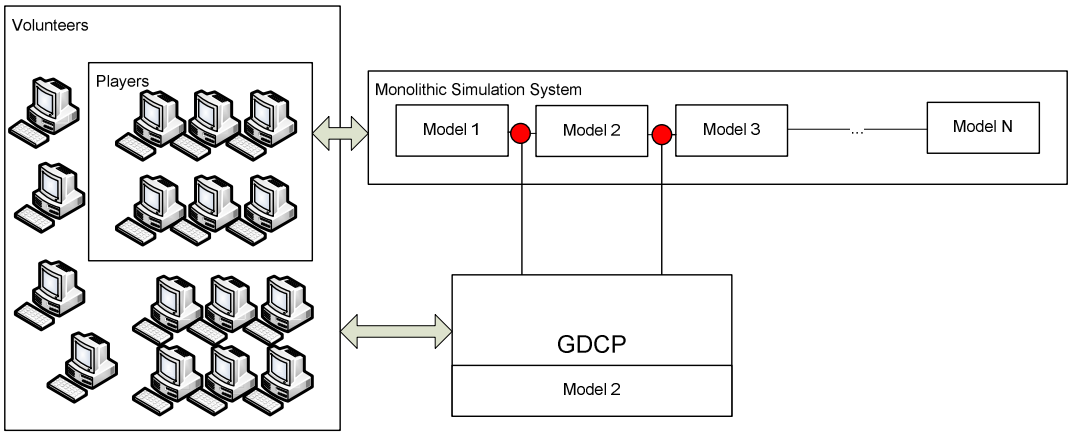

Figure 2. Hybrid Architecture for Constructive Interactive Simulation.

The dynamic addition of computational capacity, proportional to the game size (i.e. number of clients), will result in a response time's improvement of the decoupled model. If the response time with the former architecture is quadratic with respect with the quantity of users connected, adding computational power in this way will lower the degree of the curve, converting it in an almost linear relationship. Nevertheless, the internal process carried on by the GDCP in order to administer, distribute and execute the jobs introduces an overhead that impacts on the overall system's response time. Thus, it is necessary to find out the equilibrium point where the GDCP starts to improve the performance in order to decide whether to use it or not depending on the amount of users connected.

Characteristics of the Architecture

This strategy not only improves the throughput (response time) of the simulation's job but also provides other interesting points that are worth noting, such as robustness, reliability, scalability, manageability and economics.

Robustness is one of the main features of the GDCPs; the jobs are completed with predictable performance, and the platform masks underlying resource failures. It tolerates job, machine, and network failures and includes a variety of mechanisms for ensuring timely job completion in the instance of such malfunctions. This property is based in redundancy and verification algorithms. The jobs can be distributed to more than one computer with two goals: to avoid a single point of failure and to compare the results for increasing or decreasing the trustworthiness of the volunteers. These two facts also increase the reliability of the system, avoiding the introduction of errors in the calculus.

The scalability is another direct benefit of the architecture. The addition of players and, therefore, the

addition of computers to the cloud produces directly an increase in the computational capabilities of the infrastructure, introducing new donators and giving more power to the whole. Therefore the infrastructure is adaptive; its computational power is proportional to its computational needs. Furthermore, HACIS permits the addition of computers that are not playing in the simulation game, facilitating an effortless increase in the capabilities.

GDCPs are meant to work with hundreds of thousands of computers with completely heterogonous hardware and software configurations, even running different operating systems; the manageability of the infrastructure must be assured. The available GDCPs provide easily managed tools for administrators which allow visibility into the evolution of the computing cloud, monitoring the nodes' situation and controlling them remotely, if the node is correctly configured. They also provide statistical tools which allow the administrators to perform projections about the performance, the robustness, the heterogeneity, and the reliability of the whole system. Additionally in the most well-known GDCPs, such as BOINC, the volunteer software is easy to deploy and can be customized to be controlled through command line, reducing the awareness of the final user as to the use of his or her machine, in the case that it is being used concurrently for other work inside the organization.

Another important aspect is the ease of implementation and deployment derived from the simplicity of the concept. The modifications necessary for the simulation software are straightforward if the system has been designed as a pipeline. On the GDCP, the implementation is also simple. There are two main points to focus on: the interfaces and the algorithm. The interfaces should be defined and developed in order to correctly connect the simulation system to the GDCP. This is usually effortless considering that models are self-sufficient, hence, the input/output interfaces are well defined. On the other hand, the algorithm must be rewritten using the GDCP API and reformulated in order to divide the workload to parallelize it. This latter issue is seldom problematic in view of the parallelizable nature of the simulation systems, usually making interactions in two-by-two fashion. Furthermore, using the same environment multiple models can be executed developing and deploying multiple projects over the same GDCP. This allows the scheduler to take care of the distribution strategies and priorities.

All these benefits translate directly into an economic improvement. The lack of necessity of adding ad-hoc computing power, i.e. dedicated servers, lowers the cost of not only hardware acquisition but also of hardware/software maintenance and other side expenses such as cooling systems, datacenter floor-space and electricity. Moreover, the adaptive behavior reduces to the minimum necessary the emanations of CO2 and thermal output of the system, maintaining on-line only the resources that are being used, preventing the server overloading.

From the information security perspective, GDCP used inside a controlled environment presents no security risks. However, if it is used in open environments two main issues must be considered: security of the information – in particular, confidentiality – and misuse of the grid. Other minor problems are described in BOINC, 2007. If any data processed in a job is at a classified level of access, the administrator must take into account that the data is usually transmitted to and from the clients in plain ASCII text files or open XML Web Services. This risk can be mitigated by using symmetric cryptography; before transmitting data to the volunteers, the server can encrypt the data file with a symmetric master key composed of the file's hash appended to a nonce (Luna & Dikaiakos, 2008). Dispersal Algorithms can also been applied to provide high availability and assurance for the GDCP by means of data fragmentation (Rabin, 1989). With this approach, the job is split into *n* fragments; all of these are signed and distributed to *n* remote donators. The segmentation should be done in order that the access to a single fragment does not reveal the whole information (Beberg & Pande, 2007).

Another possible security concern is that such computational power may be taken over by an attacker for the purpose of running distributed attacks, such as DDoS (Distributed Denial of Service) or for performing brute force code breaking, or transforming the grid in a botnet. This risk can be mitigated by using custom certificates for validating both the clients and the servers as applied in the BOINC project MTA SZTAKI (BOINC, Verifying application signatures using X.509 certificates, 2007).

With respect to the correctness and reliability of the results produced by the grid, the accuracy is assured by activating the voting and redundancy features. GDCPs provide sabotage-resistance techniques, i.e. protocols for trust management in desktop grids. For this purpose, low-level strategies are employed to gather valuable information for the creation and maintenance of local reputation lists (Domingues, Sousa, & Silva, 2007).

CASE OF STUDY

Batalla Virtual

The main products of the Software Research and Development Office of the Argentine Army (CIDESO) are simulation programs. The CIDESO develops constructive simulation software for military training, Simupaz[1] and Batalla Virtual[2](BV). Both use a customizable set of simulation models for emulating different portions of reality. These models range from movement to close combat, passing through logistics, line of sight, artillery, communications, and weather, among others. The strategy of modeling carried on by the CIDESO is top-down. The engineers model the organizational behavior and, then, if necessary, they evolve it lowering the level down to single vehicle or personal actions.

BV immerses the players in a virtual scenario that they have to solve using their theoretical military knowledge, applying the doctrine, in the best possible way. It currently implements sixteen simulation models. One game engages several people, considering the variety of roles involved in a real military operation. Each participant commands one element, i.e. command a unit or a company and its sub-units, and specials roles such as the commander and the staff control the whole brigade situation. BV was born as an upper-intermediate commanding training system, i.e. it simulates up to a company level. That means that the quantity of elements does not pass the couple of dozens for each force deployed, training concurrently up to 140 officers. However, in order to reach a higher degree of realism, some models simulate the behavior of every single vehicle or person deployed in the game, e.g. line of sight (LoS) model. Moreover, as users get more involved with the product, the necessity of increasing the level of detail is growing. This requirement is leading not only to the development of new simulation models in order to perform more meticulous actions but also to modify and adapt the already existing models for supporting more elements.

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 hardware capacity. Even though the code of the algorithm has been reengineered several times and considerable optimizations have been achieved, it was not enough in order to be prepared for modeling a higher level of granularity and fidelity in the simulation.

In order to propose a definitive solution, a fundamental change in the architecture has been performed: the migration from conventional client/server architecture to a HACIS which merges the original approach with a GDCP. The architectural change actually included rewriting the code. For implementing a distributed architecture, the code of the target simulation model has been re-implemented in C++ though it was originally written in SmallTalk.

Implemented HACIS

The HACIS' implementation in BV was divided in five stages: (1) the selection of the GDCP, (2) the evaluation of the models for detecting the most computational consuming, (3) the model's decoupling, (4) the implementation of the algorithms using the GDCP APIs, and (5) the integration and deployment.

BOINC as GDCP

The selected Grid Desktop Computing Platform for implementing the solution in the CIDESO was BOINC. It was selected mainly because it covers the requirements and it is easy to implement. Other GDCPs have been evaluated, e.g. Entropia and Globus Toolkit, however as BOINC has been conceived for working in an open environment, it provides extra security and consistency verifications natively. Moreover, since we were looking for improving the throughput in short running process – i.e. interactive simulations–, the code development for both client and server in raw C++, without using any abstraction such as Web Services, provides an additional factor of performance optimization.

In a nutshell, BOINC is a GDCP oriented to open volunteer projects where the grid resources' are provided by Internet users (Anderson, 2004). It is an open-source infrastructure which gives the users the possibility of creating projects built upon it. The projects can be either closed or open. The closed ones are those where the providers of computational resources are in a controlled set of possible volunteers while the open are the ones submitted to the BOINC servers and any Internet user can voluntarily donate part of her own computational time to the project. As it is employed for working in open environments BOINC provides a special validation module which checks the results given by the volunteers in order to certify its accuracy. In other words, it checks that the results are

[1] **Simupaz**, constructive simulation software for United Nations scenarios.

[2] **Batalla Virtual (BV)**, constructive simulation software for tactical military training and operation planning.

correct. This cross check is necessary because the volunteers are not one hundred percent trustworthy. This requires BOINC infrastructure to work using a redundant architecture where every single task is processed by more than one resource and the results are compared among them using voting algorithm to decide which one is valid. This infrastructure is well known for being the first volunteer grid computing approach and for building the most powerful computer facility ever with the project SETI@HOME, (Anderson, Cobb, Korpela, Lebofsky, & Werthimer., 2002).

In BOINC, the job manager is divided in two subsystems: the work generator and the work assimilator. The first is the input interface; it receives the jobs, divides it in tasks and submits them to the scheduler. The tasks are the atomic units of jobs that will be processed in the volunteers' computers. Once the scheduler takes control of the tasks, it assigns them to the volunteers who process them and returns the partial results to the work assimilator. The work assimilator is responsible of joining the partial results in a single response which gives solution to the job sent by the grid user. Both, the work generator and the work assimilator have APIs for developing custom programs, for each solution.

Thus, the implementation of BOINC consists of the development of the work generator, the work assimilator and the distributed algorithm. After the development of these software's pieces, some configurations must be carried on for tuning the environment.

Decoupling the Model: Line of Sight

The performance tests' results executed over every single model of BV have shown that the most computational time consuming model was DLI (Detección, Localización e Identificación – Detection, Localization and Identification). DLI is the Line of Sight (LoS) simulation model of BV. It is responsible to show to each user only the game's elements that he is able to see considering his position, the position of all the other elements, the terrain's topography and the exploring devices (binoculars, radars, etc.) of the commanded element. The only way of doing so is by trial-and-error strategy. For each user commanded element it is necessary to check, using each exploring device, whether or not every other element is visible considering the capabilities of the said device, e.g. maximum and minimum range, direction, etc., and the topography.

The complexity of this problem is proportional to $O(n^2-n)$, n being the number of elements in the game. In addition, for each pair of elements that are being checked, every point of the map that lies on the line which connects both elements must be analyzed in order to see if there is a geographical obstruction between them. This search must be exhaustive, making the necessary processor time really high.

As input, the model receives a computational model of the terrain's topography as a matrix of elevations, the position of all the elements in the map, and the exploring devices of each element with its properties, e.g. an optic device defined by a magnification and a field of view. As output it returns an array which shows which elements are visible to each player and through which device. The decoupling is clear-cut and two interfaces have been developed for output and input. The output interface generates a plain text file representing the elements' positions and its exploring devices, and sends it via TCP/IP to a predefined address where the job generator of the GDCP is listening. While the input interface is a listening port which waits until the output of the job is received from the work assimilator. It then introduces the results in the simulation pipeline for continuing the process.

Distributed Algorithm

It was necessary to migrate and re-engineer the original source DLI's code in order to use BOINC's APIs in the development of the distributed algorithm since BV is fully developed in SmallTalk, which uses only OOP (Object Oriented Programming), and BOINC client uses C or C++. This code re-engineering resulted in two major benefits: the ability to use BOINC and also a realized performance improvement for the non-distributed execution. The use of ANSI C produced by itself an enhancement in the response time of the model through its exploitation of the benefits of memory pointers.

Despite the re-coding, from the design point of view no changes have been necessary. The algorithm evaluates for each element, for each exploring device, the possibility of seeing the other elements of the game based in the absolute positions, the exploring devices' properties and the topographical information. The only conceptual change is that, for the distributed fashion, the algorithm takes only a subset of elements which confronts with all the others, dividing the problem space. The difference between the distributed and the non-distributed algorithm is not in the algorithm itself but rather a difference in the input set of data.

Customization & Integration

For merging BOINC with BV the work generator and the work assimilator have been developed using the

BOINC's APIs. When the platform receives the job, via the TCP/IP output interface deployed in BV, the work generator program splits it in several atomic tasks and sends them to the BOINC scheduler. Each task consists of a set of observers with its exploring devices and the positions of all the other elements – targets – of the game. On the other side, as the server receives the partial results from the volunteers, 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, closing the distribution circuit. Figure 3 shows the implemented architecture of HACIS.

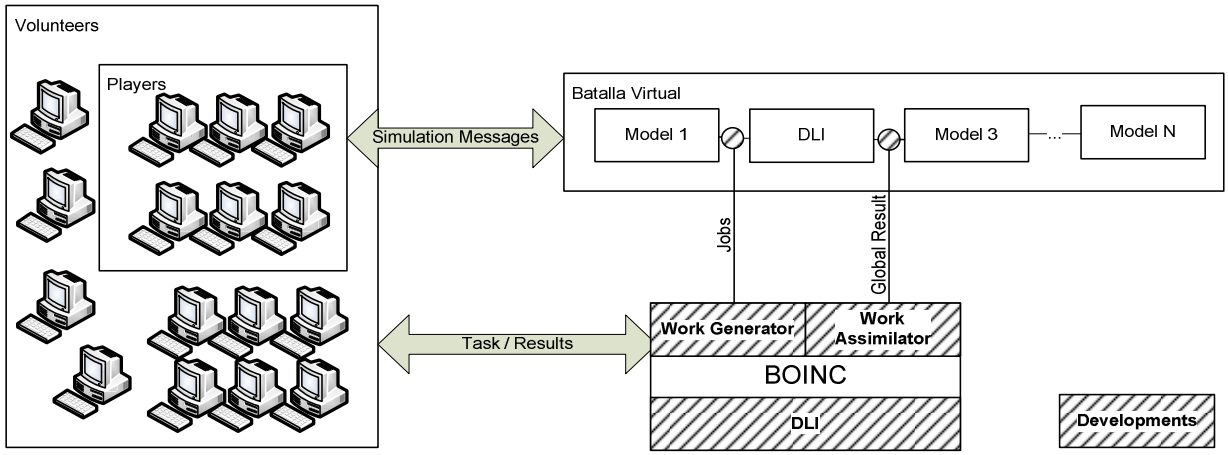

Figure 3. HACIS Implementation for BV.

In addition to the development, some critical configurations must be applied over the BOINC server in order to obtain the expected results. Although BOINC is one of the most well known grid desktop computing frameworks, it does not fully adapt to the interactive simulation systems requirements. It was actually meant for solving other kinds of problems. Two main challenges have been encountered: the pull strategy and the short-running processes.

As explained before, BOINC was conceived for an intergrid volunteer use, i.e. wide geographical distributed and untrustworthy volunteer. This fact requires BOINC's client to work in pull mode. The client asks (pulls) for jobs to the server when his computer conditions reaches a set of predefined preferences. When a client is installed, it is necessary to configure how much RAM, hard disk and processor the volunteer is willing to donate to the project. Then, the local client's scheduler decided whether to pull jobs from the server to be processed or not.

For the objective of HACIS, the ideal situation was the opposite. A push-like strategy fits better than a pull strategy. Thanks to the flexibility of the platform, through simple configurations, it was possible to preset the pulling time, which is the time interval in which the client is forced to ask for new jobs. By reducing this parameter to the minimum, i.e. setting this variable to 1 second, we functionally change the strategy which compels the client to continuously ask the server for new jobs. If there are no jobs to be processed then nothing is sent.

With respect to the short-running processes, the problem became more complex. Most grid desktop computing infrastructures are designed for processing long-running processes, i.e. computations which the result is not expected to be quasi-linear. However, BV, for being an interactive system, requires immediate results.

These delays have a low impact in long-running processes because they are considerably smaller than the time required for executing the task itself. Nevertheless they must be considered in short-running processes. In order to optimize the distribution circuit, the overheads introduced by the distribution platform have been analyzed. Three critical points of delay have been observed: communication, work splitting and job assimilation; in long-running process these delays vanish in front of the heavy processes executed by the clients. With the intention of reducing these delays as much as possible, all the developments have been made in ANSI C, using low level instruction, i.e. pointers and raw TCP/IP communication interfaces. It is important to note that these three overheads have a peculiarity. The communication overhead is almost constant, regardless of the amount of jobs submitted; the time expected will be roughly the same. The work splitting

and job assimilation overhead are linearly related to the amount of elements of the game. This makes the consumed time using the distributed algorithm increase more slowly relative to that of a non-distributed strategy. In spite of the optimizations, there exists an intrinsic limit for which the distributed solution does not increase the performance of the overall system. The equilibrium point (where both strategies perform equally) must be found in order to decide whether to activate the distribution or not.

TESTS

With the objective of evaluating the performance gained using HACIS, several tests have been performed. The evaluation objective was always centered in the response time (RT) –interval between the start of the requirement and the reception of the complete response–. Table 1 shows the average response times, expressed in seconds, of all the distributed configurations, with 2, 3 and 5 volunteers (V) plus de client/server (C/S) and the average gain with respect to the quantity of deployed elements.

These tests have been performed in a controlled environment using a BV server provided with an Intel® Xeon® E5502 and 2 GB of RAM, running a Windows ® 2003 Server Edition and five client machines, used as BV client and, also, as BOINC volunteers, with an Intel® Dual Core (E5200) and 2 GB of RAM, running Windows® Vista. The BOINC server was mounted in a virtual machine, using VM Ware Workstation, with one processor and 512 MB of RAM, running a Debian Kernel 2.6.

The variables analyzed during the tests were: the amount of observers for each task, the number of clients acting as donators, the number of elements deployed in the game and the size of the map where the elements interact. The first variable represents the task's size that each volunteer will process within the limit of time allowed for a single cycle. If the task contains all the deployed elements, the performance should be roughly the same as in the client/server architecture. This variable must be customized for each setting, depending on the computational power of the donators. If it is set very low, the donators process the tasks so quickly that the transmission overhead exceeds the processing time, degrading the system's performance. Conversely, if it is set very high, few tasks are created and the computation is performed as if it was not distributed. For our configuration, the equilibrium was found in eighty (80) observers for each task.

With respect to the number of clients donating processor for small games, introducing a high number of volunteers does not improve the performance considering that the number of tasks is not big enough to distribute the workload over all the volunteers. The tests have been performed with two, three and five donators.

Table 1. Results Obtained.

Qty of Elements Deployed	Response Time (in seconds)												Average RT Gain (%)		
	Environment 1				Environment 2				Environment 3						
	C/S	Distributed			C/S	Distributed			C/S	Distributed					
		2 V	3 V	5 V		2 V	3 V	5 V		2 V	3 V	5 V	2 V	3 V	5 V
200	33	74	68	65	21	76	52	63	19	73	57	60	-67%	-59%	-61%
300	50	75	71	64	45	75	52	60	40	75	60	60	-40%	-25%	-27%
500§	238	80	81	71	215	74	54	58	151	78	65	62	161%	208%	216%
800	263	103	105	80	238	87	100	72	220	90	60	63	158%	185%	236%
1000	491	182	111	105	490	90	119	88	358	210	72	61	228%	350%	437%
1500	903♠	312	102	109	927♠	290	138	110	575♠	289	95	62	169%♠	621%♠	766%♠
2000	1453♠	404	181	119	1541♠	392	157	118	878♠	417	141	92	221%♠	703%♠	1061%♠

♠ Client Server System Crash, value calculated using a quadratic regression. § Equilibrium Point.

The map's size impacts indirectly in the computing requirements. In small maps containing many elements, said elements are drawn closer therefore increasing their interaction and, by extension, the computational needs. The topography of the map also affects the performance. The more irregular the environment is –i.e. more mountains have– the higher the computational requirements are. This is due to the complexity of the trigonometric calculus it has to execute in order to trace the line of sight, if the terrain is planar, the calculus are easier. With the objective of assessing the impact this variable three environments have been tested: Corrientes (Environment 1), which is almost plane, Buenos Aires (Environment 2), which has some irregularities, and Neuquen (Environment 3), which is over the Andes Mountains.

For each configuration consisting of a set of number of elements, environment, quantity of donators, ten tests have been executed. Five tests with the elements dispersed on the map and five with the elements in a contact situation (fighting).

Tests' Analysis

There are four major aspects that are worth note: the equilibrium point, the behavior of the client/server architecture and the distributed solution with respect to the quantity of elements, the problems observed when playing with more than 1000 elements, and the average gain using HACIS.

Figure 4 shows results obtained for Environment 2, Buenos Aires. The three environments show the same behavior. For the client/server architecture the growth presents, as expected, a quadratic behavior with respect to the number of elements while for the distributed architecture the growth presents a linear correlation. The results also confirm the hypothesis that for a given number of elements the number of volunteers affects the overall performance. In particular, exceeding 1000 elements, the setting with two volunteers starts behaving like the client/server architecture (C/SA), degrading the response time in more than a linear correlation.

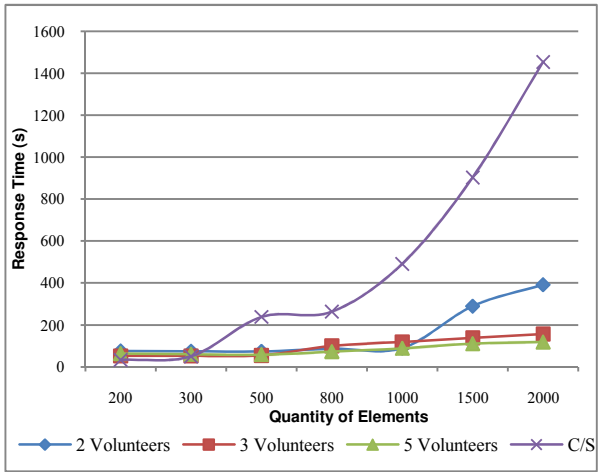

Figure 4. Results Comparison: C/SA vs. HACIS.

Another important coincidence in the tests is that the equilibrium point is between 300 and 500 elements. With less than 500 the performance obtained by the client/server architecture is better, thus the distribution of the workload is not recommended. With 500 or more elements, the improvement is evident. Furthermore, upon increasing the number of elements deployed to 1000, with the hardware used as BV Server, the system crashed. The times shown in the Table 1 for 1000 and 1500 elements in the client server case have been estimated using a quadratic regression.

Finally, the Table 1 shows the average gain of performance using HACIS compared to the client/server architecture. This gain was calculated using the Equation 1 for each pair of values and then summarized in a single value taking the average by quantity of elements and number of volunteers for each environment, e.g. for 500 elements, in Environment 1, with 5 volunteers, $T_{c/s}$=238s, $T_{HACIS} = 71$, which equates to a performance gain of 235%. Doing the same calculus for the other two environments, we obtain 270% and 143%, what results in an average Gain (G) of 216%.

$$Gain\ (G) = \frac{\sum_{i=1}^{3} \frac{T_{c/s} - T_{HACIS}}{T_{HACIS}}}{3} \qquad (1)$$

For the presented results, the gain ranges between 161% and 437% taking into account only the real values. Using the progressions, the gain realized could potentially increase to more than 1000% for 1500 deployed elements and five volunteers. Figure 5 shows the average gains obtained.

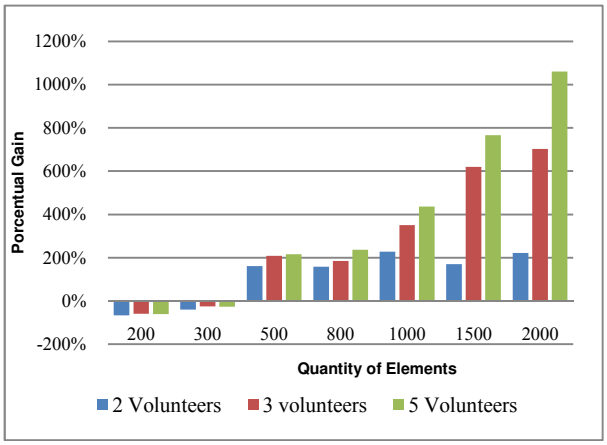

Figure 5. Average Gain using HACIS.

CONCLUSIONS

The hypothesis that a HACIS architecture can increase performance in a distributed constructive simulation has been confirmed. The exploitation of the clients' idle computer power connected to a simulation game using a distributed platform improves the performance of the overall system without interfering with the flow of the game, in some cases realizing performance increases of

more than 400%. Furthermore, the HACIS proposal allows playing games of sizes that are impossible to execute with the current infrastructure which removes the limitation on the number of concurrent users. This improvement in the performance has a didactical impact considering that the diminishing in the response time enhances the realism of the simulation, avoiding dead times during the simulation.

This framework is economically feasible not only for reducing the hardware costs (both direct costs such as acquisition and indirect costs such as maintenance) but also for permitting the simultaneous training of a larger number of users without the necessity of acquiring new infrastructure for playing. For instance, three parties' games in which the simulation system can emulate the interaction of three forces concurrently and thus, train up to 210 officers – three complete brigades - in a single game can be played.

HACIS also stands as a technical advance from the GDCP starting point in that it demonstrates that this technology, originally meant for long running processes (i.e. processes that take days or, even, months), can be used for relatively short running processes in using hardware configurations of convenience. This outcome represents the most important finding considering that it permits application of the framework to any interactive system in which the response time required is in the order of a couple of minutes thereby extending the use from any constructive simulation system to any other computer system that requires a high throughput.

In conclusion, great performance results at low costs can be obtained by applying a simple architecture modification, introducing an Open Source GDCP solution.

REFERENCES

Anderson, D. P. (2004). BOINC: A System for Public-Resource Computing and Storage. *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing* (pp. 4-10). IEEE.

Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., & Werthimer., D. (2002). SETI@home: an experiment in public-resource computing. *Communications of the ACM , 45* (11).

Beberg, A. L., & Pande, V. S. (2007). Storage@home: Petascale distributed storage. *Parallel and Distributed Processing Symposium, International* (p. 482). Los Alamitos, CA, USA: IEEE Computer Society.

BOINC. (2010). *BOINC Stats*. Retrieved April 25, 2010, from BOINC: http://boincstats.com/stats/project_graph.php?pr=bo

BOINC. (2007). *Security issues in volunteer computing*. Retrieved April 28, 2010, from BOINC: http://boinc.berkeley.edu/trac/wiki/SecurityIssues

BOINC. (2007). *Verifying application signatures using X.509 certificates*. Retrieved April 28, 2010, from BOINC: http://boinc.berkeley.edu/trac/wiki/CertSig

Chien, A., Calder, B., Elbert, S., & Bhatia, K. (2003). Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing , 63* (5), 597-610.

Condor Project. (2010). *Condor High Throughput Computing*. Retrieved April 28, 2010, from Condor: http://www.cs.wisc.edu/condor/

Constantinescu-Fülöp, N.-Z. (2008). *A Desktop Grid Computing Approach for Scientific Computing and Visualization.* Norway, Trondheim: Norwegian University of Science and Technology.

Domingues, P., Sousa, B., & Silva, L. M. (2007). Sabotage-tolerance and trust management in desktop grid computing. *Future Gener. Comput. Syst. , 23* (7), 904-912.

Ferreira, L., Berstis, V., Armstrong, J., Kendzierski, M., Neukoetter, A., MasanobuTakagi, et al. (2003). *Introduction to grid computing with Globus.* Riverton, NJ, USA: IBM Corp.

Foster, I., & Kesselman, C. (2004). *The grid: blueprint for a new computing infrastructure.* Boston: Morgan Kaufmann Publishers.

Luna, J., & Dikaiakos, M. D. (2008). *Providing security to the Desktop Data Grid.* Cyprus: Institute on Knowledge and Data Management.

Rabin, M. O. (1989). Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM (JACM) , 36* (2), 335 - 348.

Rajkumar Buyya, David Abramson, Jonathan Giddy and Heinz Stockinger. (2002). Economic models for resource management and scheduling in grid computing. *Concurrency: Practice and Experience, Grid computing* , 1507 - 1542.

Roure, D. D., Baker, M., Jennings, N., & Shadbolt, a. N. (2003). *The evolution of the grid. In Grid computing - making the global infrastructure a reality.* John Wiley and Sons Ltd.