Improved Load Balancing on Distributed Massive Simulation Environments

Cristoforo Caponigri, Gennaro Cordasco,
Rosario De Chiara, and Vittorio Scarano

ISISLab, Dipartimento di Informatica
ed Applicazioni "R.M. Capocelli",
University of Salerno,
Fisciano 84084, Italy
{cordasco,dechiara,vitsca}@dia.unisa.it

Abstract. In this paper, we report the findings we gathered in designing and implementing a system that provides a distributed massive simulation environment. Massive Battle is a system capable of simulating historical battles for the purpose of learning and to carry out historical researches (e.g. what-if scenarios). We present a distributed implementation of Massive Battle and some early tests. We report and discuss some analysis of the problems related to the workload distribution in this particular environment. We report how is possible to measure a better load balancing by adopting a more general scheme of computation that generalize the assignments that each peer has to complete together with simulation.

Keywords: Massive Simulation, Peer–to–Peer systems, Load Balancing.

1 Introduction

Distributed Virtual Environment (DVE) is an emerging research field which combines 3D graphics, networking and behavioral animation with the purpose of simulating realistic and immersive virtual environments offering a high degree of interactivity. The distributed nature of these systems widened the scenarios of use that now ranges from online videogames to serious games for training including online cooperative systems for learning and problem solving.

In this paper we will discuss the design of a Peer-to-Peer (P2P) system for Distributed Virtual Environments. In particular we will report our experiences in implementing it and we will discuss some experimental results, as well. Distributed Virtual Environments (DVEs) constitute a challenging research area in the wider area of the Distributed Systems. DVEs are designed with the intent to convey and highly interactive experience to a number of users widespread geographically [17]. An architecture based on a single server, or even a small number of servers, is not able to handle the load generated by such systems, on the other hand a centralized server architecture allows to target serious problems like accounting and security.

A motivation supporting the adoption of P2P architectures is that the most of the DVEs intend to develop Massively Multiuser Virtual Environment (MMVE). Videogames are particular MMVEs, and their success is directly proportional to the number of subscribers they have. For example *World of Warcraft* and *Second Life* have reached around 10 million subscribers worldwide and roughly 1 million of active users [13]. Indeed, during last years, a lot of interests has raised for the development and research of novel platforms for next-generation MMVEs. The idea is to use the P2P architecture to let peers to communicate with other peers through the overlay network, without going through the server. Furthermore each peer may also alleviate the server work by sharing some of the tasks such as maintaining game state. The division of such responsibilities between peers is an interesting research topic.

1.1 Massive Simulation Environments (MSEs)

The simulation of groups of characters moving in a virtual world is a topic that has been investigated since the 1980s with the purpose of simulating a group of entities, dubbed *autonomous actors*, whose movements are related to social interactions among group members.

A classical example of use of this approach is the simulation of a flock of birds in the most natural possible way. Elements of this simulated flock are usually named *boids* (from *bird-oid*) and got instilled a range of *behaviors* that induces some kind of *personality*. A widespread approach to this kind of simulations has been introduced in [15]. Every boid has its own *personality* (e.g. the trajectory of its flight) that is the result of a weighted sum of a number of *behaviors*. The simulation is performed in successive steps: at each step, for each boid and for each behavior in the personality, the system calculates a request to accelerate in a certain direction in the space, and sums up all of these requests; then the boid is moved along this result. The behaviors are, in the most of cases, simply geometric calculations that are carried out for each boid considering the k-neighbors it is flying with: for example the behavior called *pursuit* just let the boid to pursuit a moving target (e.g. another boid). Each boid reacts to its k-neighbors, which constitute its neighborhood. Given a certain boid out of a flock of n boids, the most simple way of identifying that boid's neighborhood is by an $O(n^2)$ proximity screening, and for this reason the efficiency of the implementation is still to be considered an issue.

1.2 Designing a P2PMSE

In [7] we presented Distributed Massive Battle, a DMSE that is able to simulate historical battle from the past. The purpose of our system is to expand the number of simulated actors in a MSE by distributing the computational load to various PCs (peers) connected to the system. In Distributed Massive Battle the functionality of a peer are divided into two categories, Simulation and Rendering. Each of the workers participate to the simulation while one single peer is devoted to the visualization of the simulation.

We also provided some tests to assess the performances of such DMSE. Tests revealed that the architecture presents a quite good scalability and the communication overhead due to the peers interaction is dominated by the computational power provided by the peer.

Our result. In this paper we present a generalization of Distributed Massive Battle where each peer provides CPU power, to contribute to the simulation, and, in exchange, it can visualize a part of simulation. As we said before, the most reasonable scenario of use of such systems are MMVEs. In a MMVE, on each peer, together with the simulation, several other complex tasks needs to be carried out (e.g. artificial intelligence, social interactions, complex user interfaces interaction etc...), in this paper we will refer to such computations with the name "assignments". We intend to take into account assignments tasks by generalizing the model. The rationale behind this decision is to let the uneven load distribution in the simulation, we measured in [7], be balanced, with no particular effort by the system, by a randomization of the assignments. For further details see Section 3.

2   Distributed Massive Battle: The Architecture

In this Section we describe the architecture of Distributed Massive Battle. Massive Battle is a MSE capable of animating autonomous actors with the purpose of reconstructing interactive scenes from a battlefield showing a number of platoons fighting each others [2].

2.1   Background

Peer–to–Peer systems. Peer-to-peer (P2P) is a class of network applications that takes advantage of existing computing power, computer storage, and networking connectivity, which are available at the edges of the Internet. Hence P2P allows users to leverage their collective power to the benefit of all. After the initial popularity of centralized Napster and flooding based networks like Gnutella, several research groups have independently proposed a new generation of P2P systems which are completely distributed and use a scalable Distributed Hash Table (DHT) as a substrate. A DHT is a self–organizing overlay network that allows to add, delete, and lookup hash table items. One of the most important benefits provided by P2P systems is the scalability. In a P2P system, each consumer of resources also donates resources.

Typically, DHTs are based on similar designs, while their search and management strategies differ; they include Chord [16] (based on the hypercube), CAN [14] (based on the torus), P-GRID [1] (based on trees), Pastry [8], or Tapestry [18]. One of the reasons for the success of the DHT approach is that DHTs provide a generic primitive that can benefit a wide range of applications.

Massive Battle. Massive Battle is an example of *serious game* system that offers an effective way of simulating historical battles for the purpose of learning (e.g. providing new insights for battles to engage students) and to carry out

historical researches (e.g. what-if scenarios). The simulation of historical battles also imposes some constraints on the number of agents the system is capable of simulate: as an example the Waterloo Battle involved ≈ 250000 soldiers, while Massive Battles, running on an off-the-shelves PC, is capable of simulating only ≈ 5000 units, at an interactive rate (≈ 25 frames-per-seconds). Massive Battle is implemented in C++ and is based on Ogre3d, a rendering engine, for this reason Massive Battle can be considered a reasonable approximation of a real DVE as a Massive Multiplayer Online Role-Playing Game (MMORPG).

2.2   Design Issues

The design of Distributed Massive Battle has been carried out by addressing four main issues: world partitioning, world state propagation, self-synchronization and load balancing [6].

World Partitioning. A scene in Massive Battle is defined by a map, platoons and checkpoints: each platoon pass through the checkpoints assigned to it. To achieve scalability, we adopt a Geographic decomposition approach: the environment map is partitioned into a set of static Regions. The peer which is responsible for a region is dubbed *Region Master*. The granularity of the world decomposition (that is, the region size and, consequently, the number of regions, which a given map is partitioned into) determines a trade-off between load balancing and communication overhead. The finer is the granularity adopted, the higher is the degree of parallelism that, ideally, can be reached by the system. However, due to regions' interdependency and system synchronizations, fine granularity usually determines a huge amount of communication. Our system is designed to be used with different granularity.

World State Propagation. We adopted a simple Publish/Subscribe mechanism: a multicast channel is assigned to each region; users then simply subscribe to the channels associated with the regions which overlap with their AOI to receive relevant message updates.

Self-synchronization. One of the goal of the design is to implement a *self-synchronizing* system. Each simulation is decomposed in time slots (henceforth steps). Each step is associated with a stable state of the simulation. The number of steps is used as a *wall-clock* so that each event can be associated with a timestamp. Regions are simulated on a step base. Since the step i of region r depends on the states $i-1$ of r's neighborhood (the regions which confine with region r), the step i of a region cannot be executed until the states $i-1$ of its neighborhood have been computed and delivered. In other words, each region is synchronized with its neighborhood before each simulation step. The peer in charge for each region has a double buffer in which it stores the state of the region neighbors. One buffer is used for even steps states while the other one for odd steps. When the simulation of a even (resp. odd) step is terminated, the region master waits until the odd (resp. even) buffer is full. Using this synchronization approach, we have that the timing of two closed region may differ by at most 1. So two buffers are enough to realize the synchronization barrier.

Load Balancing. One of the motivations of the P2P infrastructure described here is to address the needs for more computing power. In order to better exploit the computing power provided by the peers of the system, it is necessary to design the system so that the simulation always evolves in parallel, avoiding bottlenecks. Since the simulation is synchronized after each step, the system advances with the same speed provided by the slower peer in the system. For this reason it is necessary to design the system in order to balance the load between the peers. We addressed this problem by relying on three factors: (i) the node id on a DHT are distributed uniformly; (ii) it is possible to tune the granularity of world decomposition. This decision allowed us to implement a totally decentralized system without introducing too much communication overhead; (iii) we split the duties of each region in assignments that are carried out by different peers in such a way to improve load balancing.

2.3   System Architecture

In Distributed Massive Battle [7] we adopted FreePastry, the open source version of Pastry [8], as the underneath network infrastructure and we used Scribe [5], the multicast infrastructure built on top of Pastry, to disseminate the simulation state and, at the same time, synchronize the system.

   It is worth annotating here how we addressed the problem of distributing the simulation which is carried out by Massive Battle. Massive Battle has been designed and implemented as a simulation written in C++ to be executed on a single PC and this needed to be adapted for the network infrastructure that was implemented in Java. To address this problem we used Java Native Interface (JNI) that allowed us to invoke Java method from C and vice-versa. Once the technological issues have been worked out, we had to take a decision on how to distribute the computational load of the simulation. We just added a World State Propagation step after the Simulation and Rendering steps: each region r publishes the updates to all regions that are subscribed to r and (ii) r waits for the updates from all the regions r is subscribed to (r's neighbourhood). The World State as well as the self-synchronization logic is implemented in the Java part, while the Simulation methods are invoked once the buffer contains all the necessary information to perform a step. The P2P infrastructure is totally agnostic respect to the payload that is propagated among regions: upon receiving/transmitting the updates are handled (marshalling/unmarshalling) in the Simulation engine.

3   Improved Load Balancing

The main finding of our previous paper was that even if Distributed Massive Battles showed a good scalability, the general load balancing of the system, which relied on the distribution of IDs on the DHT, was not enough. Hence an enhanced load balancing strategy is needed. Several approaches have been proposed to balance the workload across the peers, for instance a dynamic partitioning can

be used, but on the other hand, the management of dynamic regions requires a large amount of communication between peers that consumes bandwidth and introduces latency [3,9].

As we said before in a MMVE each peer will carry out a number of different assignments, each of this assignments will consume CPU time. Together with the Simulation, part of the CPU time will be employed in performing, for example, high quality visualization of the simulated scene, gathering input from user, implement a voice chat system. For sake of the clarity of the description we will consider how the Rendering can be performed in such MMVE, without loss of generality. In our experiments every region will have two different assignments: Simulation and Rendering. At the very beginning of the execution, each peer will be Region master for a number of region, the coupling will be performed by the IDs in the DHT. The Region master will publish the updates it calculates and will received updates from border regions.

What we have just described until now is the situation we depicted in the previous paper. We now intend to take into account a variation: we split each computation step into a set of assignments. Besides the Simulation assignment, a range of other assignments shall be considered, for instance the Rendering of some regions. Each peer, that actually is a PC with a user, will be interested in visualizing a number of regions in the map, to see how the Simulation is running. For this reason such peer will subscribe to updates from the regions it means to visualize, so, together with the updates it will use to carry on the Simulation, it will also receive the updates from the region whose it perform the Rendering. The workload of the Rendering assignment will depend on a number of factor like quality of rendering, screen resolution, complexity of the model, and, is proportionally to the number of agents it intends to render: the more agents are present in the regions the peer intend to visualize, the higher will be the workload, and larger will be the portion of CPU time subtracted to the Simulation and absorbed by the Rendering.

In the most general case, together with Simulation and Rendering we can consider other tasks, as we said above. For this reason, in our experiments, each peer will be Region master for a number of cells and will also subscribe to a number of cells for what we have defined assignment. For the purpose of the testing, after performing a Simulation step, each peer spend an amount of idle time proportional to the number of agents that belongs to the regions it is interested to. This time consumption is performed by using a `sleep()` system call, for a number of milliseconds proportional to the number of agents present in the region managed by the peer.

In Figure 1 we have summarized the whole process we have described above: in the Figure is depicted a single step of computation as it is performed on a single peer. Each peer is actually made by two parts, one written in C++, that is in charge for the Simulation and the other assignments, and a Java part that handles the communication on the P2P overlay network. The Simulation assignments executes the Distributed Massive Battle engine with the purpose of animating the agents. Other assignments are executed in C++ and corresponds
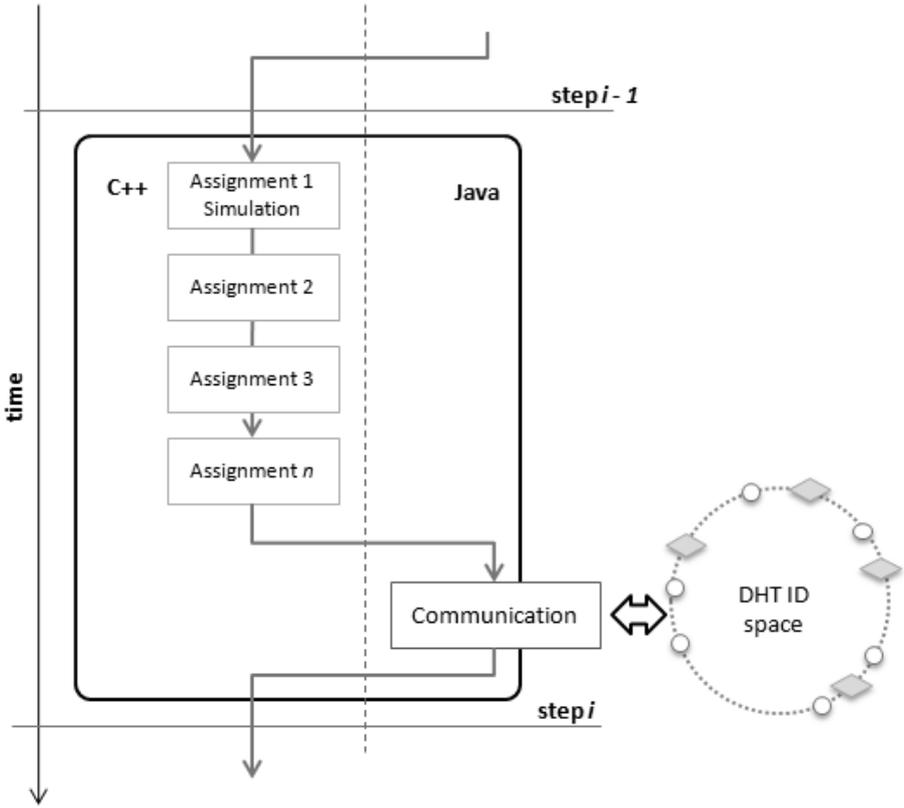
Fig. 1. The execution of a single step of computation for a single peer

to a `sleep()`. The communication task is where the synchronization takes place and the double buffers mechanism is used. It is worth noting that the Java part is agnostic respect to the data it actually handles, and this allows to independently expand and modify the C++ part to add new functionalities.

3.1 Tests

We performed a number of tests of the system in order to evaluate the load balancing improvement obtainable by decomposing, on each peer, the computation task into two components: Simulation and Assignments.

Test setting. Tests were conducted on a scenario consisting of 64 regions (a 8×8 grid). On each run the distribution of the regions to peers, both for the simulation and for the assignments, is decided by randomly associating DHT identifiers to both regions and peers (each peer is given two IDs, one is used for the simulation and one for assignments). Sixteen platoons of 100 soldiers (overall 1600 actors) were placed on the map. Each platoon follows a prefixed
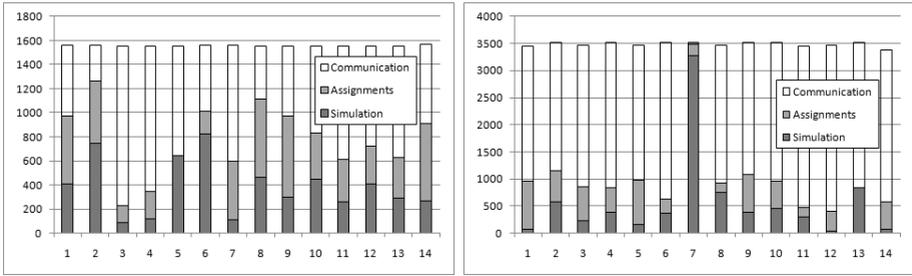
Fig. 2. Two runs of the system 1000 steps: (left) best case (right) worst case. Times are expressed in seconds.

path which guarantees that all the regions become non-empty at least once during the simulation. We ran the simulation with 14 peers while the number of simulation step is 1000. Since the performances of the system are also influenced by some arbitrary factors (for example, the allocation of ID to peers and regions can lead to more or less balanced workload), we executed each test 10 times All experiments are performed on 14 mid-range PC having similar characteristics: Intel Xeon dual-core processor running at 2.80 GHz, with 2 GB of main memory. All the PCs are interconnected with a Gigabit Ethernet network.

In order to evaluate the load balancing we compute, for each peer, the total time spent for the Simulation and the total time spent for Assignments. By extrapolating these times from the whole running time, we obtain the idle time (that is, the time spent for both communication and synchronization). We then compare the variability of this results with a similar test where no Assignments take place. Since the two data sets provide quite different means (the tests with no Assignments are shorter) we can not use neither the standard deviation nor the variance to compare the variability of these results. Both standard deviation and variance are strictly dependent to the mean of the data. We compare the variability of our data set by using the coefficient of variation, which is the ratio between the standard deviation and the mean. The coefficient of variation is a dimensionless number and is independent from the mean, so it fits our purpose.

Discussion. In Figure 2.(left) we report the time distribution among the peers, for two executions which correspond to the best (left) and the worst case (right). The results are encouraging, decomposing the computation task into two tasks meaningfully reduce the variability of the results. The improvement ranges from 43% to 65%. For instance, in Figure 2.(left), the amount of computation peer number 6 carried out for the simulation has been counter balanced by a small amount of computation for the assignments; the opposite happened to peer 14. Even in the worst case, Figure 2.(right), the balancing is improved since the most loaded peer (number 7) received an extremely small amount of assignments.

4   Conclusion

The rationale behind our research is to design a system for Distributed Virtual Environments in which work distribution, synchronization and load balancing are implemented in a totally distributed manner. Plenty of works that address each of this aspects are available in literature, but we intend to study them in an integrate manner, in order to verify how they interact.

In this paper, in particular, we addressed the load balancing problem we scoped in a previous research, by considering how, in a more general case of a Distributed Virtual Environment where each PC was not only busy in Simulation but was also employed in more general assignments (rendering, social interactions, user interaction, etc...). In the situation where the computational load is heavier, tests say it is possible to obtain a better load balancing because the unbalancing we measured when peers were busy only in Simulation of certain regions, was balanced by the execution of assignments (e.g. Rendering) of other regions.

References

1. Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., Schmidt, R.: P-grid: a self-organizing structured p2p system. SIGMOD Rec. 32(3), 29–33 (2003)
2. Boccardo, A., De Chiara, R., Scarano, V.: Massive Battle: Coordinated Movement of Autonomous Agents. In: Proc. of the Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS) (2009)
3. Buyukkaya, E., Abdallah, M., Cavagna, R.: VoroGame: A Hybrid P2P Architecture for Massively Multiplayer Games. In: Proc. of the 6th IEEE Consumer Communications and Networking Conference (CCNC 2009), pp. 1–5 (January 2009)
4. Castro, M., Jones, M.B., Kermarrec, A.-M., Rowstron, A., Theimer, M., Wang, H., Wolman, A.: An Evaluation of Scalable Application-Level Multicast Built Using Peer-to-Peer Overlays. In: Proc. of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2003 (2003)
5. Castro, M., Druschel, P., Kermarrec, A.-M., Rowstron, A.: SCRIBE: A large-scale and decentralized application-level multicast infrastructure. IEEE Journal on Selected Areas in Communications (JSAC) 20, 100–110 (2002)
6. Cordasco, G., De Chiara, R., Erra, U., Scarano, V.: Some Considerations on the Design of a P2P Infrastructure for Massive Simulations. In: Proceedings of International Conference on Ultra Modern Telecommunications (ICUMT 2009), St.-Petersburg, Russia (October 2009)
7. Caponigri, C., Cordasco, G., De Chiara, R., Scarano, V.: Experiences with a P2P Infrastructure for Massive Simulations. In: Proceedings of the second International Conference on Advances in P2P Systems (AP2PS 2010), Florence, Italy, October 25-30 (2010) (to Appear)
8. Druschel, P., Rowstron, A.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: Proc. of the 18th IFIP/ACM Inter. Conference on Distributed Systems Platforms (Middleware 2001), pp. 329–350 (November 2001)

 9. Kang, H.-Y., Lim, B.-J., Li, K.-J.: P2P Spatial Query Processing by Delaunay Triangulation. In: Kwon, Y.-J., Bouju, A., Claramunt, C. (eds.) W2GIS 2004. LNCS, vol. 3428, pp. 136–150. Springer, Heidelberg (2005)
10. Karger, D., Lehman, E., Leighton, F., Panigrahy, R., Levine, M., Lewin, D.: Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In: Proc. of the 29th Annual ACM Symposium on Theory of Computing (STOC 1997), pp. 654–663 (1997)
11. Knutsson, B., Lu, H., Xu, W., Hopkins, B.: Peer-to-Peer Support for Massively Multiplayer Games. In: Proc. of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004), p. 107 (2004)
12. Nandan, A., Parker, M.G., Pau, G., Salomoni, P.: On index load balancing in scalable P2P media distribution. Multimedia Tools and Applications 29(3), 325–339 (2006)
13. Pittman, D., GauthierDickey, C.: A Measurement Study of Virtual Populations in Massively Multiplayer Online Games. In: Proc. of the 6th ACM SIGCOMM workshop on Network and system support for games (NetGames 2007), pp. 25–30. ACM, New York (2007)
14. Ratnasamy, S.P., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proc. of ACM Special Interest Group on Data Communication (ACM SIGCOMM 2001), San Diego, CA, US, pp. 161–172 (August 2001)
15. Reynolds, C.W.: Flocks, Herds, and Schools: A Distributed Behavioral Model. Computer Graphics 21(4), 25–34 (1987)
16. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, M., Dabek, F., Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. IEEE/ACM Transactions on Networking (TON) 11(1), 17–32 (2003)
17. Waldo, J.: Scaling in games and virtual worlds. Commun. ACM 51(8), 38–44 (2008)
18. Zhao, B., Kubiatowicz, J., Joseph, A.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Report No. UCB/CSD-01-1141, Computer Science Division (EECS), University of California at Berkeley (April 2001)