# Experiences with a P2P Infrastructure for Massive Simulations

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

*ISISLab - Dipartimento di Informatica ed Applicazioni "R.M. Capocelli",*
*Università degli Studi di Salerno, Fisciano, Italy.*
*Email:* {cordasco, dechiara, vitsca}@dia.unisa.it

***Abstract*—Massive Multiuser Virtual Environments (MMVEs) are rapidly expanding both in the number of users and complexity of interactions. Their needs of computational resources offer new challenges for the computer scientists. In this paper we present an implementation and some early tests of a Massive Simulation Environments, a particular MMVE, distributed over a Peer-to-Peer infrastructure. We provide some analysis of the problems related to the workload distribution in this environment. Simulation tests show a good grade of scalability and the communication overhead, due to the peers interaction, is dominated by the computational power provided by them.**

***Keywords*-Massive Simulation, Peer–to–Peer, Load Balancing.**

## I. Introduction

The recent growth and popularity of online Massively Multiuser Virtual Environment (MMVE) have raised a lot of interests for the development and research of novel platforms for next-generation MMVEs. Examples of this trend are *World of Warcraft* and *Second Life* which have reached around 10 million subscribers worldwide and roughly 1 million of active users [1].

The design and management of MMVE, due to their highly interactive nature, poses many unique challenges compared to traditional network domains [2]. A single server, or even a small number of servers, is not able to handle the load generated by such systems. So, even if the client/server approach is quite common for small-size MMVEs, it is mandatory for next-generation MMVEs to use the computing power of a group of many servers with dedicated responsibility.

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.

Acknowledging the fact that client/server paradigm does not fit well the MMVE scenario, one of the main issues that should be considered designing a DVE system is how to split the responsibilities between the servers/workers. Several approaches have been proposed. For instance, every server can have a different assignment (communication, artificial intelligence, physics, game state) or, on the other hand, a single server acts as a *factotum* server for a portion of the whole environment (a.k.a. *shard*). In this case the actors which belong to different shards cannot interact with each other (each shard represents a distinct copy of the whole environment). Another approach to achieve scalability, named Geographic decomposition, exploits the locality of the environment, decomposing the map on which the game is played into different regions, each of which is associated to a worker. Geographic decomposition suits particularly well to support a completely distributed MMVE, built on top of Peer-to-Peer (P2P) infrastructure, where the responsibility of maintaining the whole environment is shared among all its users. In this approach the workload balancing is essential for both the overall performance and scalability. In the context of DVEs we are particularly interested in Distributed Massive Simulation Environments (DMSEs), which, for the same reasons, appear as a suitable problem that can greatly benefit from the use of a P2P approach.

### A. 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 [3]. 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

 ISBN: 978-1-61208-102-1 

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.

### B. Designing a P2PMSE

For the purposes of this paper we refer to MSEs as a family of complex interactive 3D environments whose main functionalities can be divided in Simulation and Rendering. We intend to expand the number of simulated actors in a MSE by distributing the computational load to various PCs connected to the system. In a proper DMSE both Simulation and Rendering can be accomplished by a distributed network of computer. Each of the workers offers computational power and in exchange yields the right to visualize the simulation.

Each user connected to the system will have two different subsystems running on his PC: a simulation engine and a visualization engine. The simulation engine will take the responsibility of simulating a small part of the actors in the system, while the visualization engine will let the user to choose which part of the map to visualize. The system architecture is based on a *Distributed Hash Table* (DHT) that will let the user to dynamically connect to the system in a totally distributed manner: once a new peer is available in the system, it will receive part of the simulation and will receive the updates from the system.

The visualization engine will let the user to freely place a camera in the environment and this camera will define an Area of Interest (AOI). AOI is a fundamental concept, as even though many actors and events may exist in the simulated environment, the user, as in the real world, is only interested by nearby actors or events. AOI thus specifies a scope for information which the system should provide to the user. Notice that each subsystem, simulation and visualization, will have its own AOI: the AOI for the simulation, henceforth neighborhood, is defined by the position of actors the peer is simulating, the AOI of the visualization is defined by where the user placed the camera.

### C. Our result

We present our experiences with the design and implementation of a fully distributed Massive Simulation Environment. We report and analyze several experimental results we have obtained with our system. Simulation tests show a good grade of scalability and the communication overhead, due to the peers interaction, is dominated by the computational power provided by them.

### D. Paper structure

In section II we describe the system architecture and the details of the implementation, while in section III we report the tests setting, the test results and some discussion. In section IV we discuss some possible future works.

## II. An architecture for a DMSE

The core of this paper is to describe our experiences in designing, implementing and assessing performances of a DMSE built on top of a P2P system. The motivation for such architecture lays in the usage we envision: users will connect to such system in order to simulate more and more complex scenarios, and the system will exploit the computational power provided by connected computers. This scenario of use allows the system performances to scale together with the number of users, while a multi-processor architecture would bound the scalability to the number of processors that are readily available. The study has been carried out starting from our past experiences in designing Massive Battle [4]. 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.

### A. Background

*Peer–to–Peer systems:* In peer-to-peer (P2P) network, computers can communicate and share files as well as other resources. As opposite to the client-server approach, participants are at the same level (peers). By means of a client users can connect to the network; communication among nodes is done by message exchanges aimed to: announce their presence in the network, ask for resources, serve resources requests. 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. Proposed systems are based on various forms of distributed hash tables, they include Chord [5] (based on the hypercube), CAN [6] (based on the torus), P-GRID [7] (based on trees), Pastry [8], or Tapestry [9]. 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 $\approx 250000$ soldiers, while Massive Battles, running on an off-the-shelves PC,

 ISBN: 978-1-61208-102-1 

is capable of simulating only $\approx 5000$ units, at an interactive rate ($\approx 25$ 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).

### B. Design Issues

The design of the DMSE has been carried out by addressing four main issues: world partitioning, world state propagation, self-synchronization and load balancing [10].

*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 whole environment map is partitioned into regions. Regions are assigned to peers, by mapping both regions and peers to the DHT key space: regions as well as peers are associated with an ID computed by using a consistent hash function [11]. The peer whose ID is the closest to the region ID is dubbed *Region Master* and is responsible for that region (cf. Fig. 1).

Each Region Master is responsible for:

- Simulate all actors which belong to the region;
- Deliver the state of the region (that is the state of each actor which belongs to the region) to the peers whose AOI overlaps with the region;
- Handle *handovers* of actors between regions.

The choice of the world partitioning technique is important for the efficiency of the whole system. Two key factors need to be considered:

- Static or Dynamic Partitioning;
- The *granularity* of the world decomposition.

Dynamic partitioning can be used, for instance, in order to balance the workload across the peers, but on the other hand, the management of dynamic regions requires a large amount of communication between peers that consumes bandwidth and introduces latency [12], [13]. For this reason, in this work, we opt for a Static Partitioning.

Similarly 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:* In order to implement a P2P architecture for MMVEs, a communication infrastructure is needed to deliver messages to a wide group of users. Being multicast communication not available on geographical network, application-layer multicast provides a workaround [14]. A well-known mechanism used to propagate world state information is based on the Publish/Subscribe design pattern: 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. For instance, SimMud [15] uses Scribe [16], an application-layer multicast built on top of the DHT Pastry [8]. Scribe is decentralized and highly efficient because it leverages the existing Pastry overlay.

*Self-synchronization:* One of the goal of the design is to implement a *self-synchronizing* system (the whole simulation should proceed simultaneously, without the use of a central coordinator, which might represent a bottleneck). We use a standard approach to achieve a consistent synchronization of the distributed simulations. Each simulation is decomposed in time slots (henceforth steps). Each step is associated with a fixed state of the simulation. Regions are simulated step by step. Since the step $i$ of region $r$ is computed by using 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 number of steps since the beginning of the simulation is used as a clock so that each event can be associated with a system timestamp.

*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 two factors: (i) the node id on a DHT are distributed uniformly, and this means that each peer in the system has equal probability of receiving a region (ii) it is possible to tune the granularity of world decomposition. This decision allowed us to implement a totally decentralized system: more effective load balancing techniques would have required some degree of coordination.

### C. System Architecture

Like SimMud [15], we decided to adopt FreePastry, the open source version of Pastry [8], as the underneath network infrastructure and we used Scribe [16], 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

 ISBN: 978-1-61208-102-1 

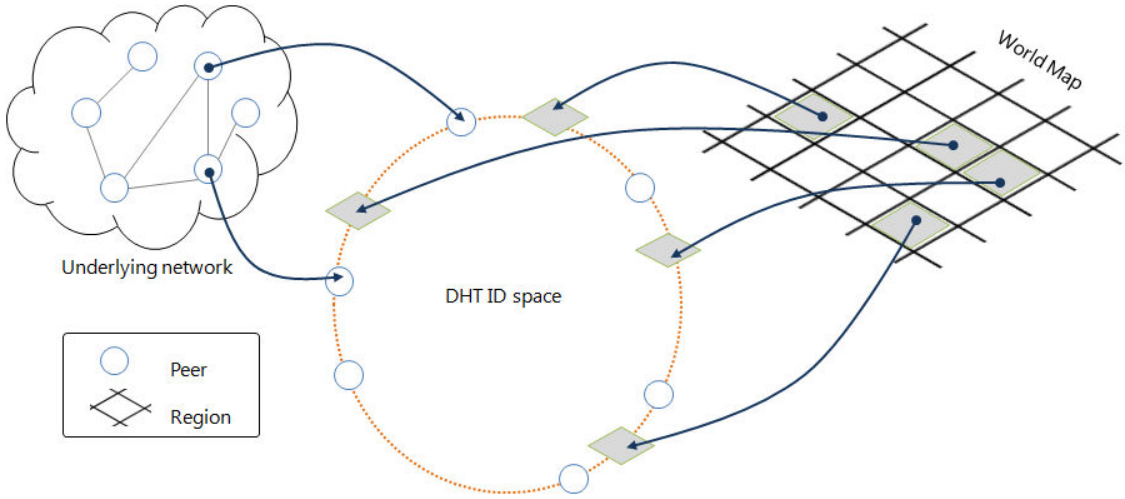


Figure 1. Geographic decomposition of the environment map over a DHT key space.

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.

## III. Tests

We performed a number of tests of the system in order to assess both scalability and the resilience of the system to uneven loads of calculation.

*Test setting:* Simulations were conducted on a scenario consisting of $64$ regions (a $8 \times 8$ grid). On each run of the simulation the distribution of the regions to peers is decided by assigning randomly DHT identifiers to both regions and peers. Sixteen platoons of $100$ soldiers (overall $1600$ actors) were placed on the map. Each platoon follows a prefixed path which guarantees that all the regions become non-empty at least once during the simulation. It is worth noting that both the number of actors to be simulated and the number of iterations to be performed represent the workload of a test (i.e., the number of computations required for performing the whole simulation). 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 (we empirically observed that $10$ runs of test are enough to obtain stable results). For each test we will present the results in terms of both means and boundaries. All experiments are performed on $16$ 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. For the purposes of the performances evaluation we decided to not perform any rendering, in this way all the computational power is devoted to simulation, communication and to maintain the P2P infrastructure.

*Scalability:* The rationale behind this test is to evaluate the performance improvements obtained by increasing the number of peers. Thus, the question we would like to answer is: is the communication overhead, due to peers' interaction, dominated by the computational power provided by peers? The interaction between peers is a direct consequence of the fact that exchanging information between peers (e.g., actor positions, events, transitions across regions) is usually needed. The efficiency of the system is measured by evaluating the completion time of $500$ simulation steps. We ran the simulation described above with $1, 2, 4, 8, 12$ and $16$ peers (each peer is executed on a dedicated PC), in order to depict the scalability trend of our architecture. Figure 2 depicts the obtained results: the top chart shows both the mean (diamonds) and the maximum/minimum values registered during the tests. The bottom chart depicts a comparison between the average simulation times against the optimal linear speedup. The tests show that our schema presents

 ISBN: 978-1-61208-102-1 

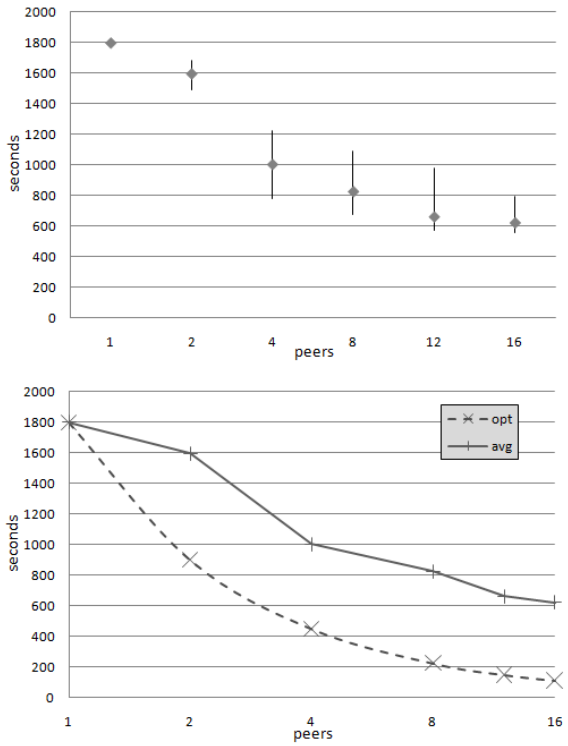


Figure 2. Simulation duration: (top) average and boundaries (bottom) compared with the optimal linear speedup.

almost linear scalability. However, as the number of peers increases, the gap between the results achieved by our architecture and the ideal speedup grows. We conjecture that this trend is due to the fact that in throughout the tests we used the same granularity of decomposition (64 regions) while, in order to obtain better performances, the granularity of the decomposition should be tuned in such a way that the ratio between the number of regions and the number of peers is constant.

*Time distribution:* The second test is focused on how the whole simulation time is spent by each peer involved in the computation. In this test the number of peers involved into the computation is always 16 while the number of simulation step is 1000. Our purpose is to determine how effective is the straightforward load balancing strategy adopted by our architecture and how much unbalanced distributions may affect the whole completion time. In order to evaluate the load balancing we compute, for each peer, the total computation time spent. By extrapolating this time from the whole simulation time we obtain the idle time (that is, the time spent for both communication and synchronization). The results show that the computation time spent by the peers during the computation is pretty variable. In order to provide the reader a better perception of the results, we show the time measures obtained by two simulations which reflect the best and the worst observed case. In Figure 3 each bar represents a peer, the dark portion of each bar represents the computation time.

Even in the best case (cf. Figure 3 (top)) we observe that more than 70% of the whole computation time is executed by only 5 peers. The best case simulation took 1198 seconds while the worst case took 3208 seconds; we also run a test of pure computation on a single PC without the P2P infrastructure and we measured, under the same circumstances, a running time of 2820 seconds. We also observed that the load unbalancing is not only due to an unbalanced distribution of regions to the peer. Indeed, the computation time required by each region is different and varies during the computation.

As said before the idle time comprises: (i) the communication time, which represents the time needed to deliver the state of the region after the computation of each step. This time does not depend on the workload distribution and is equal for each peer; (ii) synchronization time, that is the time spent by each peer waiting for slower peers. The latter time strongly depends on the system load balancing. We can estimate the communication time by considering the idle time of the most loaded peer. This peer has synchronization time close to 0 because it is always the last peer to complete the simulation step. Let us consider the second bar (from left) in Figure 3 (down): the communication overhead for the peer that obtained such performance, and consequently, for each other peer in the simulation, is only 38 seconds. As a result we have that the synchronization time represents a very big portion of the idle time. This consideration explains why the completion time is strongly influenced by the workload distribution.

*Discussion:* The tests show great variance of the system performances; nonetheless it is possible to see some potentialities of such architecture: in each of the test setting we had really fast simulation runs, together with slow runs. The running time is influenced by uneven load distribution and how to measure such a load can be a serious issue. It appears that the number of regions per peer is not correlated with the measured performance (see the first two bars in bottom part of Figure 3). More sophisticated load balancing techniques need to take into account the number agents each peer has to simulate.

## IV. Conclusion

DMSEs, due to their scalability requirements, appear to be a natural application for P2P architectures. However DMSEs are quite different from classical P2P applications which are mainly devoted on sharing files as well as storages. On DMSEs the shared resources consist of CPU cycles while the purpose of the architecture is to maintain a distributed data storage (that represents the state of the simulated environment) keeping the latency as small as possible.

We presented an infrastructure that implements a DMSE and tests to assess the performances of such DMSE. We

 ISBN: 978-1-61208-102-1 

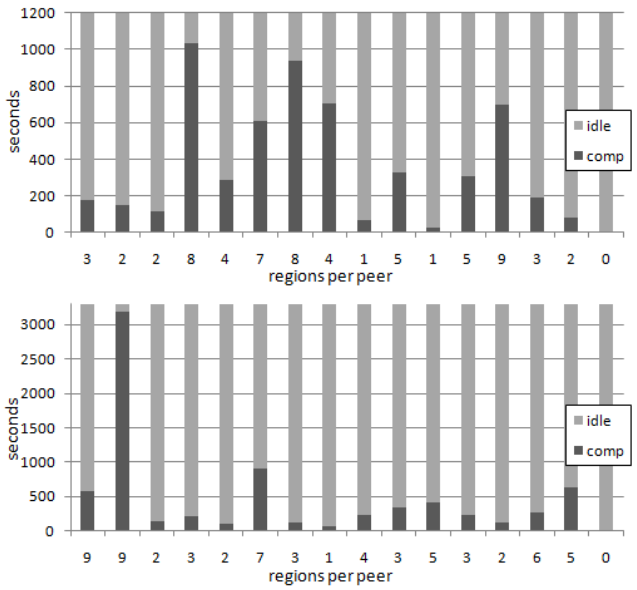


Figure 3. The figures depict how the simulation time is spent by each peer involved in the computation. The total simulation time is split into: computation time, time spent by the peer simulating a region of the map; and idle time (shown shaded) which comprises both the communication and synchronization time. Two cases are showed which corresponds to: (bottom) the worst case (maximum unbalancing); (top) best case occurred during all the tests. It is reported the number of region associated with each peer.

addressed and solved, in a totally distributed manner, the four main aspects of such architecture: world partitioning, world state propagation, synchronization and load balancing. The tests revealed that the architecture presents a quite good scalability, the communication overhead due to the peers interaction is dominated by the computational power provided by the peers. Unfortunately, such scalability is hard to achieve because of variability of the load balancing. A reasonable next step is to address the problem of load balancing by employing more sophisticated techniques, the challenge is to obtain such balancing in a distributed manner, without relying on a centralized load balancer. Preliminary results show that, in the most desirable situation of a balanced workload, the system obtains significant improvements in the measured performances.

## REFERENCES

[1] D. Pittman and C. GauthierDickey, "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 '07)*. New York, NY, USA: ACM, 2007, pp. 25–30.

[2] J. Waldo, "Scaling in games and virtual worlds," *Commun. ACM*, vol. 51, no. 8, pp. 38–44, 2008.

[3] C. W. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model," *Computer Graphics*, vol. 21, no. 4, pp. 25–34, 1987.

[4] A. Boccardo, R. De Chiara, and V. Scarano, "Massive Battle: Coordinated Movement of Autonomous Agents," in *Proc. of the Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*, 2009.

[5] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," in *IEEE/ACM Transactions on Networking (TON), Volume 11, No. 1*, Feb. 2003, pp. 17–32.

[6] S. P. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. of ACM Special Interest Group on Data Communication (ACM SIGCOMM '01), San Diego, CA, US*, Aug. 2001, pp. 161–172.

[7] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt, "P-grid: a self-organizing structured p2p system," *SIGMOD Rec.*, vol. 32, no. 3, pp. 29–33, 2003.

[8] P. Druschel and A. Rowstron, "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 '01)*, Nov. 2001, pp. 329–350.

[9] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," in *Tech. Report No. UCB/CSD-01-1141, Computer Science Division (EECS), University of California at Berkeley*, Apr. 2001.

[10] G. Cordasco, R. De Chiara, U. Erra, and V. Scarano, "Some Considerations on the Design of a P2P Infrastructure for Massive Simulations," in *Proceedings of International Conference on Ultra Modern Telecommunications (ICUMT '09), October 2009, St.-Petersburg, Russia*, 2009.

[11] D. Karger, E. Lehman, F. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "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 '97)*, 1997, pp. 654–663.

[12] E. Buyukkaya, M. Abdallah, and R. Cavagna, "VoroGame: A Hybrid P2P Architecture for Massively Multiplayer Games," in *Proc. of the 6th IEEE Consumer Communications and Networking Conference (CCNC '09)*, Jan. 2009, pp. 1–5.

[13] H.-Y. Kang, B.-J. Lim, and K.-J. Li, "P2P Spatial Query Processing by Delaunay Triangulation," in *Proc. of the 4th InternationalWorkshop on Web and Wireless Geographical Information Systems (W2GIS 2004)*, 2004, pp. 136–150.

[14] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman, "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 '03)*, 2003.

[15] B. Knutsson, H. Lu, W. Xu, and B. Hopkins, "Peer-to-Peer Support for Massively Multiplayer Games," in *Proc. of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '04)*, 2004, p. 107.

[16] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 20, pp. 100–110, 2002.

[17] A. Nandan, M. G. Parker, G. Pau, and P. Salomoni, "On index load balancing in scalable p2p media distribution," *Multimedia Tools Appl.*, vol. 29, no. 3, pp. 325–339, 2006.

 ISBN: 978-1-61208-102-1 