QuP: Graceful Degradation in State Propagation for DVEs

Lutz Behnke
Dept. of Computer Science
University of Applied Sciences
Hamburg
lutz.behnke@haw-hamburg.de

Christos Grecos
School of Computing
Univ. of the West of Scottland
Paysley, Great Britain
Christos.Grecos@uws.ac.uk

Kai von Luck
Dept. of Computer Science
University of Applied Sciences
Hamburg
luck@informatik.haw-hamburg.de

ABSTRACT

Large Distributed Virtual Environments (DVEs) combine the interactions of many users, constituting a highly complex distributed system that encompasses of a multitude of heterogeneous computers and communication over varying networks. The size of a DVE is not only measured in the spatial reach of its virtual world, but also the number of concurrent users it can provide a sufficient quality service for. The existence of hardware or software failure of these many parts must be considered the normal state of operations for any sufficiently large DVE.

We propose QuP, an approach to provide high fault tolerance when compared to many P2P approaches. QuP is also able provide the service of a large DVE service to clients connected via asymmetric or lower bandwidth networks.

Categories and Subject Descriptors

C.2 [**COMPUTER-COMMUNICATION NETWORKS**]: Distributed Systems

General Terms

Algorithms, Reliability

Keywords

DVE, MMOG, Cloud, fault tolerance, Spatial Publish Subscribe

1. INTRODUCTION

DVEs provide a platform for a wide range of entertainment and productivity applications. They allow the collaboration of a large number of participants in a game or co-workers to share common set of information in a virtual world. For smaller such environments a single computer may be able to supply the necessary coordination of the participants. But with a growing number of connected users, or more complex interactions, a distributed solution is needed.

Basically two basic architectures for DVEs have been put forward in the past: (1) a Client/Server (C/S) approach and (2) a Peer-to-Peer (P2P) approach. The C/S approach relies on a single central computer to coordinate the interaction of the users, which are then presented to the user, using a client software. A central server may possess large processing and networking resources, but scaling it further will quickly become cost prohibitive, long before it becomes unavailable to procure. A much bigger concern is the fact that the server constitutes a single point of failure that would halt the service when failing. Obviously, modeling a real world C/S implementations as a single machine is a stark oversimplification (e.g. [24]), but compared to the P2P approach, the authoritative elements are controlled by the same organizational entity. In contrast, the P2P approach requires that each participant computer shares the load of operating the DVE, and can in theory scale without limit. Entering participants would also provide more computing and networking resources to the overall system. However, P2P solutions are not without problems, as they make a number of assumptions about the type of participants and their resources. These assumptions might not hold true, especially when operating a DVE for private customers over the public Internet.

One area of DVEs driving research has been the field of Massive Multi-player Online Games (MMOGs) with their very large user base (beyond the order of 10^6 for some). But their users expect a stable service while potentially engaging in disallowed manipulation of game state (a.k.a. "cheating") at the same time. This creates an operation environment that requires very high resistance to any and all unplanned changes, regardless of willfully by attackers or due to equipment faults.

This paper proposes QuP[1], a solution to apply P2P and fault tolerance techniques to a server centric approach to structuring a large DVE from small to mid-size server components, readily available from commercial Cloud providers. QuP is intended to support a large ($> 10k$) number of concurrent users, given sufficient server resources. These users are intended to populate the same, continuous virtual environment, thus allowing them to interact without any re-

MMVE'14 March 19-21 2014, Singapore, Singapore
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2708-4/14/03 ...$15.00
http://dx.doi.org/10.1145/2577387.2577389.

[1]Originally intended just as a placeholder name for the project. As the focus changed, the original meaning of the acronym lost its meaning, but was retained as the name for the project.

strictions visible to the user.

QuP attempts to strike a balance between the requirements of operating a DVE as service in the presence of a hostile environment and maintaining control of the game state while retaining the superior scaling characteristics of a P2P based approach. Our aim is to improve the overall robustness of DVE systems.

1.1 Characteristics of Proposed Solution

QuP behaves like a client/server system when observed from the outside, but has the capability to limit the impact of any component fault. In the best case only a slight performance degradation may be experienced by a subset of users. In the worst case, a client will have to connect to a different server node of the cluster. But there will be no loss to the global world state. A client will retrieve state changes that have occurred while the client was disconnected.

QuP operates on a number of server components that communicate with each other over a high speed backbone network. The clients are assumed to utilize WAN connections, typically in the 1 to 10 MBit range. The server components are further differentiated into game server nodes and storage nodes. Clients connect to the former, the later hold and persist the game state.

QuP uses Spatial Publish Subscribe (SPS) to apply Area of Interest (AoI) Management to dynamically partition the spatial space of a large DVE. This allows the partition of the computational interaction into zones of potential interactions of the entities representing the user within the game according to the internal rules of the DVE. In the following these representations of the user with in the game are called avatars.

1.2 Modes of Failure

When designing QuP we have considered the following failure modes: a) failure of a server component b) communication errors, c) failure of a storage component.

For a single failing server node, all clients connected to that node will be disconnected from the service. Upon attempting a reconnection, the client will be assigned to another server node by the system management service. Only the state change messages already sent to the failed server node will be lost. The global state will be preserved by the storage nodes. As the clients need to be re-assigned to other nodes, a slight degradation of performance may be experienced by other clients connected to those new server nodes, due to the load created by fetching those elements of the global state that are relevant to the migrated clients. Determining the exact percentage of failed server nodes, that a QuP cluster can recover from is left for a complete implementation and further experimentation.

Failed storage nodes can be recovered from as the complete world state is replicated over a number of nodes. Errors during individual reads or writes are handled by a Quorum. Sloppy Quorum and recovery through vector clocks [12] are intended for a more complete implementation.

2. RELATED WORK

Numerous P2P solutions for many application other than DVEs, like [9] and others have been published in the past. Specific adaption to the stringent latency requirements of interactive applications of DVEs have also been the continuous focus of research for the last years. Most, like [8] form a publish/subscribe infrastructure, that will selectively forward state change information to a subset of the participants each.

Partitioning the global state of the VE has often (like [4]) been done in fixed manner, assigning nodes to fixed regions as in [10]. As discussed in [23] and others, this may lead to overloading certain parts of the service while starving others, as users tend to cluster around elements of the virtual world that interests them. To solve this zone-less partitioning techniques like [1] have been proposed.

Hu deploys SPS in [16] to partition the state of a DVE and identify the required set of recipients for state change messages. It uses AoI to partition the global state to filter the interaction between any given set of two participants. QuP extends the notion of visibility to support any form of potential interaction as presented in [14]. QuP supports multiple forms of interactions concurrently by automatically selecting the one with the greatest range as base for AoI range computation.

In [17] a number of drawbacks of a P2P solution are outlined (some are specific to that SPS approach). First among them is the requirement, that all nodes need to be trusted. Miller points out, that the management of security in a P2P DVE is difficult and may never achieve the level that is required for a commercial offering [21]. Work in this, like Merabti et al. [20] usually require redundant work to be done in some form or other, thus increasing the required bandwidth to send verification information. Miller et al also points out that most private users connect to the Internet via asymmetric connections [22]. Thus in a P2P network of equals, the lower bandwidth of the two directions has to be used as the base of performance calculations.

Waldo et al. describe Darkstar [26], modeling three distinct types of nodes operating in the DVE: a) a client to be operated by the user; b) a logic- or game-server and c) a persistence node, the later two operated by the service provider. QuP extends this model to avoid performance bottlenecks and single points of failure, for instead of a single database host, as in Darkstar, QuP consists of a larger number of storage nodes. The number of required storage nodes must be large enough to support rate of updates generated by the operation of the DVE, including replication. Also does the QuP storage node consist of a in-memory database and an additional persistence back-end, that will read data from disk only the case of failure recovery. The exact number of each is a parameter for optimisation and depended on the server and network hardware used, as well as the design of the game deployed on the platform.

Large distributed storage systems, like Dynamo [11] and Riak [6] have been state of the art for a number of years now. They support proven resistance to component failure. QuP builds upon their design, but was adapted to the specific requirements of a DVE, which allow the identification of clients with similar sets of requested data, grouping their connection onto the same server node.

QuPs design is capable of connecting external code as described in Horn et al. [15], which they state as one vital design benefit of their solution. QuP only manages the state of the game world, not the rules on how to manipulate that state. In our base design, game logic would be executed on the same node as the QuP server, but is architectural separate and not part of this paper. But as the change requests and notification of failure or success are send as messages throughout the system, they could easily be forwarded to an external program via a publicly defined protocol, like HTTP/JSON or similar (possibly after some filtering to enforce access permissions or similar restrictions).

The architecture of QuP discussed in the following does not exclude the exploitation of communication off-loading as discussed in [2]. We consider these extensions to an AoI-centric system design, which can quickly identify candidates for a direct client to client communication, especially if both clients regularly send, potentially aggregated, notifications to the server for merging with the global state.

3. ARCHITECTURE

As shown in figure 1, game server nodes not only execute the code to implement the game rules, but also act as cache to game state. For each active object (avatars, representing the connected clients, as well as objects that act as autonomous agents, like non-player characters, magic effects, etc.) the AoI management identifies the set of objects that could possibly be interacted with. The attributes of these objects are retrieved from the storage nodes. Regardless of actions by a connected clients, the cached state is continuously updated by change messages arriving from the storage nodes. A system management service allows the dynamic assignment of clients to server nodes when establishing connections between the two. By assigning the clients based on the proximity of their avatars, caching can be optimised and most interactions become local operations. This results in only changes to be written to the storage backend, and all state to have been previously been cached (during the establishment of the first client). For the worst case it degenerates into a system similar to RING [13], as each state change messages is passed from the client to the first server on to the second server and then forwarded again to the receiving client.

The global state is structured as a set of objects with an arbitrary key/value attribute set and a unique object identifier (OId). Position information as well as the interaction nimbus of each object are transformed into octree node Ids upon change to the pertaining attributes. For passive objects the node Id for both would be the same. The global state is persisted in the storage nodes. It is partitioned using consistent hashing [19] to assign a subset of the OId name space to a storage node and replicated over a number of additional storage nodes.

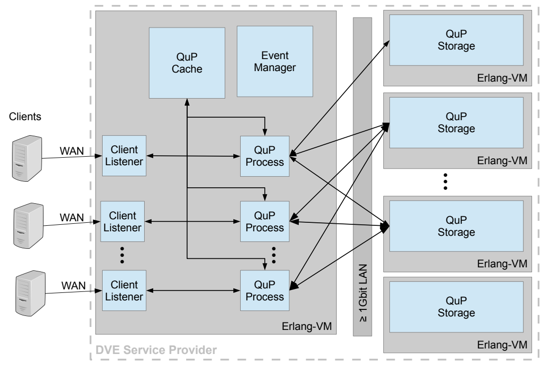

Figure 1: Components of Architecture

As state partitioning is independent from the spatial organisation of the VE, node overload through a high avatar density (and thus most likely a high rate of state change) should be governed by the network bandwidth of the server nodes. The number of storage nodes can be scaled independently to meet demand. Increasing the number of servers nodes that replicate a given section of the VE can scale the available CPU and RAM resources as long as the local changes can be transmitted fast enough.

In QuP, the server holds the authoritative copy of the state, to be queried by the clients. While clients may cache partial world state, to improve user experience, they don't strictly need to do so.

QuP does not provide a synchronous primitive for reading a value from storage to the server node. Rather, all state within the influence nimbus of the avatar is subscribed to by the server when the client connects to the system, and kept current during the duration of the connection. The state is cached by the server node. Subscriptions to state are handled on a per server node basis, for the sum of state required by the connected clients. A process (e.g. a client listener) on the server node may request object data from this cache. In the case of a cache miss, it is transparently retrieved from the storage nodes.

In order to change the state, the game process would send a request to update a value to the QuP process running on the same server node. This request will be forwarded to the storage servers, which are indicated by the consistent hashing (QuP uses SHA1 as does Riak). Provided the request does not collide with another update, the changed value is stored, and update messages are sent to all subscribers of the object (including the writing process, wich will be auto-subscribed, if need be.) There it will update the local caches on the game server nodes and forward the message to the game process that initiated it. Also all other game processes that are subscribed to that object and run on the same machine will be notified. They in turn may optionally forward the change to the client.

It has been reported, that the majority of all state information within the DVE is static or experiences little change. (Horn et al. showed that in SecondLive only 8% of all objects are mobile [15]). QuP exploits this by pre-loading all state information within the visibility envelope of an avatar. Thus all state required for updates in reaction to a command message sent by a client is already available to the server node that the client is connected to. This state change is then propagated concurrently to all other server nodes that share a spatial proximity.

QuP utilizes an octree as the method to structure the space of the virtual world, as it has been well documented for a number of years. It was chosen for its simplicity. In QuP the spatial positions are all normalized to be within the cube with length of 1, thus turning all operations within the octree into simple bitfield operations encoding the position in the tree. Other than the Voronoi diagrams used e.g. in VAST [18] or BSP-Trees as proposed in [25] , the octree is static, regardless of change in the set of connected clients, the position of the avatars or even the change of the world it models.

The subset of world state loaded onto a server node is dynamically determined by the position and size of the visibility envelope of the avatar (see 2). In an unloaded server with no initial client connections, the subtree of octree that models the world space will need to be fetched from the storage servers once a client connects. As additional clients connect, only the difference between the subtree of client A and client B need to be read from the server. This allows fast handling of connection of new clients to a server node, provided that the avatar of the client is positioned in a similar spatial region as most of the other avatars connected to the server. But if (re-)connection is cheap, clients may easily be moved to the server most optimal for the partitioning of clients among the game world.

As avatars move over time, the visibility envelopes of the avatars may move apart, increasing the amount of loaded state per server node. Should the server be unable to handle the load, it needs to shed client connections through reconnection to other server nodes.

QuP does not take into account view frustum nor culling nor any other directed AoI approaches. We assume that by rotating the avatar may quickly change its set of visible objects. As QuP is intended as an overestimating heuristic, objects are pre-fetched to a certain envelope as described in [1], as only QuP filters the intra-server traffic. Further reduction of the data sent to the clients may be done by the actual game code that runs on the game server.

3.1 Structure

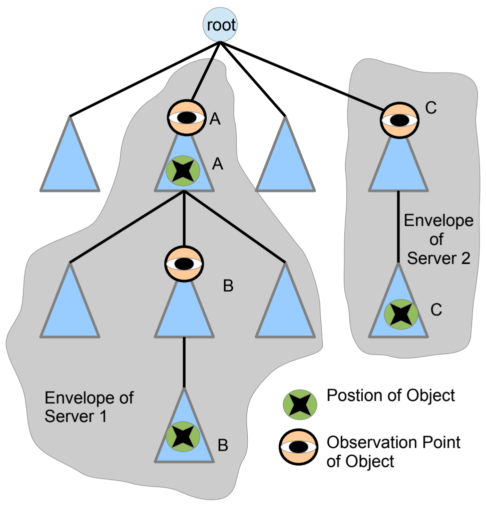

Figure 2: Example of fetch envelopes. Clients A and B are connected to server 1, client C is connected to server 2

QuP maintains two sets of objects: a) The game state objects, form the content of the game state. b) the octree nodes. Each object contains a set of key-value pairs. The keys are named via global identifiers, the values are arbitrary data. Each object is identified by a sequence of byte. The game state objects hold arbitrary state data. The octree nodes hold two attributes: the list of objects contained in this node and the list of objects that can observer the contents of the subtree with this node as the root. The system maintains these lists automatically. Objects of both types can be subscribed to and will will be copied to a server node initially as well as when the objects change. These transfers are done automatically, without need for intervention by the game software author.

An object is considered contained in an octree node, when there is no smaller node (i.e. further down the tree) which will fully encompass the spatial location and size of the object. The visibility envelope forms a type fo pseudo-objects, thus is registered in the containing octree node just as well. All positions and sizes are normalized to be between 0 and 1.0 in order to make the implementation independent from the actual size and resolution of the game world. The maximum tree depth is fixed for the lifetime of the game, to make processing the bit fields, identifying the octree nodes simpler. The octree nodes are only created on demand, to save vast amount of unused storage space

Objects and octree nodes alike are stored in a number of copies in the storage nodes. QuP uses the same assignment of fixed length chunks of multiple virtual storage nodes.

3.2 Consistency Model

When perceived from the outside, QuP provides serializability [7], as it forms a single entity. The order of messages arriving at the server is not necessary the order in which the messages are executed (i.e. the resulting state changes are performed). For conflicting changes, the one that reaches the quorum of storage nodes first will succeed. The others will return with a failure. Since change messages are provided with the old value as known by the writing process, a conflicting change can be detected. A number of non-conflicting change requests (`add-to-set`, `remove-from-set`, `increment-counter`) is provided by the API in order to reduce the number of failures.

3.3 High Load

Given that sufficient server nodes are available, a QuP based system will migrate client connections from nodes to those under lesser load, given that the interest envelopes of the connected clients on the new node are sufficiently close to those required by the migrated clients. Repeatedly adding server nodes will eventually fill up the available network resources. But as each server node acts as a replicator of state change messages forwarded to the clients, this still will aid in scalability.

3.4 Access Controls

In addition to the need to protect the state of DVE from unauthorized manipulation, a DVE may also contain content, that is indented to be perceived only by a subset of users. This may either be a certain content, which not all users have been licensed to consume or simply be a "fog of war" type functionality, common to many multi player games, where exploration is a central part of the game appeal. As outlined in [5], "explorer" type players may feel cheated, if technical means will get other players the same information. As QuP implements an authoritative server, it may filter any content disclosed to users, based on business or game rules.

4. EXPERIMENT

In order to verify our approach, we have implemented a prototype in Erlang [3]. Erlang was chosen to reduce the development effort to send messages over the network, and in order to reduce the complexity of writing software that handles a large number of threads of execution in a concurrent manner.

The main focus of the experiment is to show that the communication overhead between server nodes and storage nodes grows slower than the n^2 expected for a fully connected network between clients. The latency incurred by WAN connections between game client and game server was *not* covered by the test.

The experiment was run five standard desktop PCs (twin core, 2 to 4 Gig RAM each). The use of Erlang allows us to start a large number of virtual machines, to show how the message latency for state updates increased with added connected clients, distributed over a growing number of server cluster components.

As shown in figure 3, each physical machine runs a number of server nodes, as well as a number of storage nodes, each in separate virtual machines. On each server node a number of load generator simulate connected clients by generating a movement update every $500ms$. As consistent hashing will ensure that the objects are evenly distributed among the storage nodes, on average 80% of all message will have to be transmitted over the LAN, instead of being received locally on the physical machine.

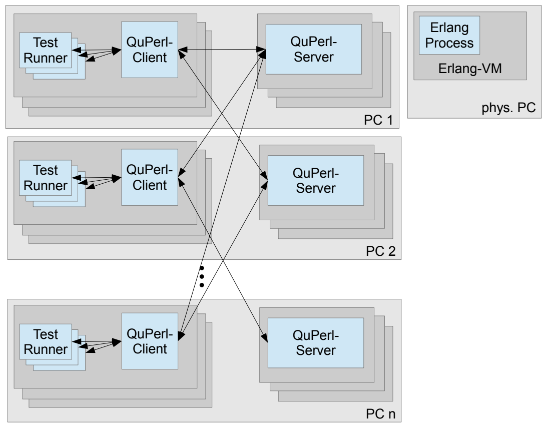

Figure 3: Experiment Setup

A fixed number of 100 load generators (clients) was started on an increasing number of virtual machines, primarily in order to establish the overhead of setting up the octree and copying the object to the individual VM. Each load generator would perform 50 steps and then terminate.

As shown in figure 4, the average latency, while increasing, due to the fixed hardware resources, increases dramatically less than the n^2 for a full distribution. The amount of latency, especially for the $90th$ percentile is still extremely high. We attribute this to the lack of fine tuning that the prototype still has.

We consider QuP to warrant further investigation to improve the worst case behavior to the levels required for interactive game play. At the same time we are already improving the test platform to include fault injection.

We plan a number of future experiments to quantify the robustness of the approach by determining two vital limits: a) the maximum number of failed nodes that will retain a global service, even if individual clients experience loss of connection and have to reconnect to working nodes. b) the number of nodes that need to fail before the system cannot recover a unified global state due to the dynamic connectiveness of the state of individual objects (objects in a container, objects in spatial locale, etc.). This object connection is different from the state commonly used to test data stores like Dynamo or Riak and thus warrants additional work. Very early indications suggest that QuP haves in a very robust nature in these situations.

Further optimisation of the architecture e.g. through the use of IP level multicast to send out messages to a large number of subscribers, or the use of message collation will be an option should the local network become a bottleneck.

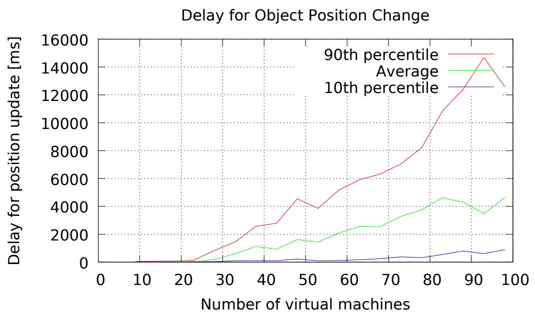

Figure 4: Update Delays

5. CONCLUSIONS

We have shown that our approach can maintain a similar level of scalability as documented for most P2P solutions in this area. We have argued that it provides higher levels of fault tolerance, as shown for approaches that our system is derived from.

QuP achieves this at the cost of increased requirements for central server infrastructure and network resources, that have to be borne by a service provider. In order to optimize the tasked resources for a changing a number of users, more research is needed to improve client migration, while maintaining the level of service for a client that is currently connected. This will allow the number of server nodes to be adapted for current users levels.

6. FUTURE WORK

The next steps in validating QuP will focus on a number of areas: (1) Improve the measurements during normal operation to measure end-to-end latency in detail and message bandwidth between server nodes; (2) Establishing the exact amount of failed nodes the architecture can support while maintaining a given number of clients for a set maximum latency; (3) Test the maximum avatar density for a given spatial area of the VE for a maximum.

For those areas as well as an general improvement of the test and validation quality, we are currently constructing the necessary test platform. It will encompass increased hardware resources and automation and measurements to allow a wide range of test parameters. The testbed will be operated as a private cloud, allowing the system management to to control system resources. This enables not only the automated testing of node failure, but also the addition and removal of resources to dynamically adapt server resources to demand by client connections.

A further area of study is the establishment of limits for the visual range of objects in relation to the local avatar density. Objects with a very high view range (e.g. an "All Seeing Eye") can quickly overwhelm the network connection of the server node it is connected to, as all state change would need to be copied to this node. Modelling the maximum range that can be supported and verifying this experimentally will be a further focus of future work.

7. ACKNOWLEDGMENTS

We would like to thank Qi Wang for his aid in writing this paper.

8. REFERENCES

[1] D. T. Ahmed and S. Shirmohammadi. A Dynamic Area of Interest Management and Collaboration Model for P2P MMOGs. *2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, pages 27–34, Oct. 2008.

[2] D. T. Ahmed and S. Shirmohammadi. Improving online gaming experience using location awareness and interaction details. *Multimedia Tools and Applications*, pages 1–18, 2011.

[3] J. Armstrong. *Making reliable distributed systems in the presence of software errors*. Phd thesis, The Royal Institute of Technology, 2003.

[4] M. Assiotis. A distributed architecture for MMORPG. *of 5th ACM SIGCOMM workshop on Network*, page 4, 2006.

[5] R. Bartle. *Designing Virtual Worlds*, volume p of *New Riders Games Series*. New Riders Games, 2004.

[6] Basho Inc. Riak, 2013.

[7] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1987.

[8] A. Bharambe, S. Rao, and S. Seshan. Mercury: a scalable publish-subscribe system for internet games. In *1st Workshop on Network and Systems Support for Games (NetGames '02)*, pages 3–9. ACM, 2002.

[9] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE : A large-scale and decentralized application-level multicast infrastructure. *Selected Areas in Communications, IEEE Journal on*, 20(8):1489 – 1499, 2002.

[10] J. Chen, S. Grottke, J. Sablatnig, R. Seiler, and A. Wolisz. Scalability of a distributed virtual environment based on a structured peer-to-peer architecture. In *Communication Systems and Networks (COMSNETS), 2011 Third International Conference on*, pages 1–8, 2011.

[11] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review*, 41(6):205–220, 2007.

[12] C. J. Fidge. Timestamps in Message-Passing Systems That Preserve the Partial Ordering. In K. Raymond, editor, *11th Australian Computer Science Conference (ACSC'88)*, volume 10, pages 56–66, 1988.

[13] T. A. Funkhouser. RING : A Client-Server System for Multi-User Virtual Environments 2 Previous work. In *ACM Symposium on Interactive 3D Graphics*, pages 85–92, 1995.

[14] N. Gupta, A. Demers, J. Gehrke, P. Unterbrunner, and W. White. Scalability for Virtual Worlds. *2009 IEEE 25th International Conference on Data Engineering*, pages 1311–1314, Mar. 2009.

[15] D. Horn, E. Cheslack-Postava, and B. Mistree. To infinity and not beyond: Scaling communication in virtual worlds with Meru, 2010.

[16] S.-Y. Hu. Spatial publish subscribe. In *IEEE Virtual Reality (IEEE VR) Workshop MMVE*, 2009.

[17] S.-Y. Hu and G.-M. Liao. Scalable peer-to-peer networked virtual environment. In *ACM SIGCOMM 2004 workshops on NetGames '04*, pages 129–133. ACM, 2004.

[18] S.-Y. Hu, C. Wu, E. Buyukkaya, C.-h. Chien, T.-H. Lin, M. Abdallah, J.-R. Jiang, and K.-T. Chen. A spatial publish subscribe overlay for massively multiuser virtual environments. In *2010 International Conference on Electronics and Information Engineering*, volume 2, pages V2–314–V2–318. IEEE, Aug. 2010.

[19] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing - STOC '97*, pages 654–663, New York, New York, USA, 1997. ACM Press.

[20] A. E. R. Madjid Merabti. Peer-to-peer architecture and protocol for a massively multiplayer online game. In *IEEE Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004.*, pages 519–528. IEEE, 2004.

[21] J. L. Miller. Distributed virtual environment scalability and security. Technical Report UCAM-CL-TR-809, University of Cambridge, Computer Laboratory, Oct. 2011.

[22] J. L. Miller and J. Crowcroft. The near-term feasibility of P2P MMOG's. In *2010 9th Annual Workshop on Network and Systems Support for Games*, pages 1–6. IEEE, Nov. 2010.

[23] P. Morillo, J. M. Orduna, and J. Duato. A scalable synchronization technique for distributed virtual environments based on networked-server architectures. In *Proceedings of the International Conference on Parallel Processing Workshops*, pages 74–81. RAND Europe/Ofcom, 2006.

[24] P. Quax, B. Cornelissen, J. Dierckx, G. Vansichem, and W. Lamotte. ALVIC-NG: state management and immersive communication for massively multiplayer online games and communities. *Multimedia Tools and Applications*, 45(1-3):109–131, May 2009.

[25] A. E. Rhalibi, M. Merabti, and L. John. Interest Management and Scalability Issues in P2P MMOG. In *IEEE CCNC 2006 Proceedings*, pages 1188–1192, 2006.

[26] J. Waldo. Scaling in Games & Virtual Worlds. *Queue*, 6(7):10, Nov. 2008.