

A New Persistence and Communication Architecture for MMOG/DVE ©

Janaka Liyanage, Brian Goldiez

Introduction

Massively multiplayer online games have become very popular in recent times due its social nature that fully immerses the player in a virtual world, as well as the recent advances in the communication infrastructures, which enables users to access and interact with rich content over the wire. However, as more and more players flock into these large virtual worlds, building, deploying and maintaining such crowded worlds has become ever more difficult. In particular, free interaction between large numbers of players operating in a secure and reliable environment is currently not supported. Additionally, if operated in an appropriate environment, games offer the potential for delivering education and training in the public sector, military, public safety, and health care industries, as well as for corporate interactions and commerce.

Literature Review

A decade ago all the online games were hosted in single monolithic game servers. Therefore each game supported only several thousand concurrent players. However, with millions of players connecting to the game worlds, such monolithic boxes are no longer practical. They also have the disadvantage of having a single point of failure, resulting in disconnecting all players from the game and losing the game state.

In order to solve the problem of handling large numbers of concurrent users, game developers introduced the idea of “sharding”, where several individual instances of the game world are maintained by several servers, with each server supporting several thousand users. While being better than the previous monolithic box approach, this approach suffers from the single point of failure problem. In addition since each server is independent of each other (except for sharing user login data, score, credits etc), users in a one server cannot interact with those in another. The popular MMOG World of Warcraft (WoW) implements this strategy for supporting millions of users [12].

A bit more complex approach for solving the scalability issue was implemented in Second Life [13]. Here the game world is partitioned into several geographically defined cells and each cell assigned to a separate server. This method allows the players to interact with all the players who are connected to the game by walking around the world and as a result moving among the servers. While being elegant, this approach introduces several of challenges that need to be solved to assure a satisfactory user experience. Issues arise with respect to supporting:

- Interactions across partitions as users come near edges;

- Moving objects across edges – moving objects from one DB to the other;
- Load balancing / dynamic partitioning based on population density;
- Permitting some non-spatially bounded interactions.

Due to the above problems with geographical partitioning, several new strategies were proposed. Pedro *et al.* introduce the concept of load balancing based on the workload generated by avatars, without considering the regions of the virtual world they are located [7]. However, this approach has serious performance limitations in multi server deployments since to serve requests for even a single player, objects from all servers need to be read. One of the most notable software that implements this concept has been the Project Darkstar, initiated by Sun Microsystems [14]. It implements the publisher-subscriber approach to relay the messages to the players, and the processing workload is broken down into small “tasks”, each of which can be scheduled in any server. Darkstar depends on a fully transactional database backend to store the game objects for synchronization and consistency handling.

While a single server implementation of Darkstar can handle several thousand users, the initial implementation of this ambitious system was found to scale negatively with multiple servers. The developers are currently researching the possibility of introducing caching and grouping of objects depending on access patterns to solve this problem.

There have been several studies to analyze traffic patterns in a typical MMOG [4, 5]. One of the most important results is about the spatial and temporal locality of information exchanged. That is, players in a particular region of the game world tend to become active at the same time. Any partitioning algorithm needs to take these characteristics into consideration in order to provide the best player experience with minimal hardware resources.

Peer to Peer (P2P) paradigms have been introduced to solve the problems of scalability and single point of failure in the MMOG engines [2, 3, 6]. Unlike in the client-server paradigm, users can communicate with each other without going through a server. A hybrid architecture combining the ideas from P2P and zoning is presented in [2]. Even though P2P architectures have been used in small-scale online games, due to limitations in authentication, security and persistence, P2P architectures are rarely used in commercial MMOGs.

Partitioning the MMOG world into number of very small cells and assigning them to individual servers in a server farm is a new method that has become popular recently [1, 11]. Dewan *et al.* introduce the concept of partitioning the world into a large number of hexagonal shaped micro-cells and dynamically assigning these cells to servers depending on the load [1]. Beatrice *et al.* implement the same idea of micro-cells with a restriction that limits the cell migration to neighboring servers [11]. This scheme reduces the communication overhead for users when they are near a cell boundary. While both of these methods introduce a nice method for load balancing, they lack methods for fault tolerance and seamless handling of user migration between cells belonging to different servers. These load balancing algorithms also fail to recognize potential non-spatial relationships between geographically distant cells.

The problem of seamless migration of users between cells has been studied extensively. The idea of dual boundary cells, where two different borders, CI (check-in) and CO (check-out) are used to minimize the number of server changes due to frequent user movement across cell boundaries, has been popular [8, 10]. An interest-driven cell crossing method for hexagonal cells has been studied in [9]. While these methods mitigate the problem of degraded user experience near cell boundaries, none of them eliminate the problem entirely.

Our solution is influenced by both Darkstar and other conventional geographic partitioning systems. We aim to strike a balance between the two approaches by maintaining the good qualities of both methods while trying to eliminate or mitigate the bad ones.

Proposed Architecture

Scenarios to be Supported (Requirements):

- New server added;
- New users logged in or new objects created;
- Server crash – need for high availability;
- Need for load balancing – no static assignments due to the dynamic nature of the simulation or game;
- Very large number of users, very large world;
- Free user movement across the world – new interactions – if partitioned world, how to handle user migration/ near boundary situation.

Characteristics of an HPC (e.g., UCF STOKES):

- Huge RAM - in-memory databases or caching;
- InfiniBand connect between servers – low-latency, high bandwidth communication between servers via RDMA.

Partitioning and Zoning Architecture

The basic concept behind our architecture is the division of the world into a number of small overlapping cells (zones), square or hexagonal in shape. While the zoning is currently spatially oriented, we anticipate extending the architecture to non-spatial partitions, such as social or group hierarchies.

Overlapping cells perform two important functions in our architecture. They provide a method for redundancy and fail safely against server crash. Secondly, they provide a method for seamless migration of users across cell boundaries. Thirdly, they provide a method for load balancing the reads and writes among the servers.

Other shapes for cells, such as triangles and circles, were also considered. However, overlapping circles do not create a homogenous pattern, and the number of triangles required to cover all the boundaries is much larger than squares or hexagons.

The use of overlapping cells means that each object in the game world is maintained in multiple servers. Only one of these copies is writable (master copy) and other copies are only readable (slave copies). The server holding the master copy of an object is the master server for that object. A graphical depiction of a square overlapping pattern is depicted in Figure 1, below.

Requirements for cell overlapping and allocation:

- 1) The master copy and all slave copies cannot be held by the same server (to maintain fault tolerance). In a spatial sense this means that all the cells covering a particular region cannot be held by the same server.
- 2) The world needs to be broken into cells of small size in order to perform effective load balancing among servers.
- 3) Cells that are too small would increase the cell management and user migration overhead.
- 4) If two overlapping regions are held in the same server, the server will maintain only a single copy.

Selecting a correct cell size is an important decision in this architecture. The decision should depend on the average/maximum flocking density of the client software (e.g., game) as well as the server capabilities. Finding an optimal cell size is an open research problem, even though various approximate sizes can be derived using the above mentioned criteria. As a general guideline we like to select a cell size that has a maximum capacity of around 200 players. Clearly, cell size is also dependent upon the number of blade servers in the system.

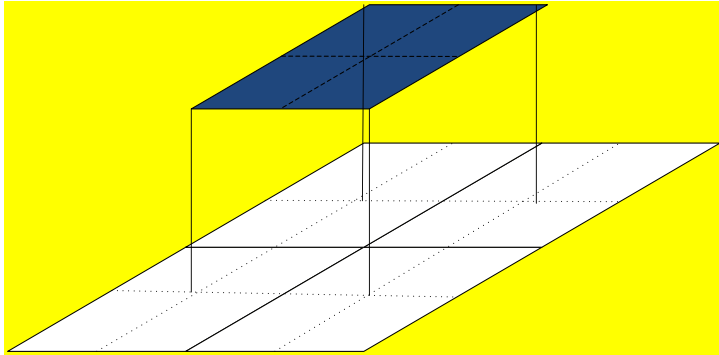


Figure 1. General Depiction of Overlapping Cell Arrangement

Cell Overlapping Patterns

Overlapping of cells can be either loose or tight. When the cells are loosely overlapped, each region in the world is replicated on two servers. When they are tightly overlapped, each region is replicated on three or four servers depending on the shape of the cells. The tightly overlapping pattern is preferred over the loosely overlapping pattern because all of the cell boundaries are covered seamlessly by an overlapping cell. However, tight overlapping tends to increase the degree of replication and therefore increases the resource usage. Some techniques for mitigating this problem are discussed in the “Allocation of Cells” section below. The overlapping patterns are depicted in Figure 2 below.

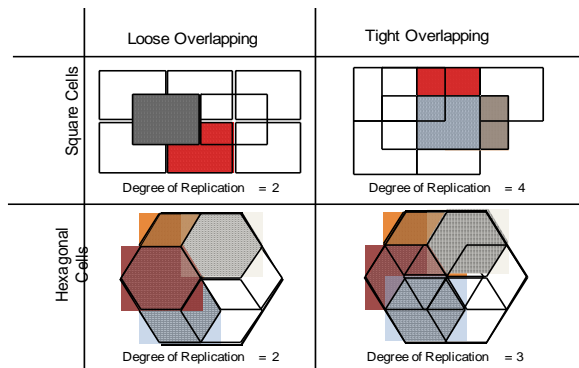


Figure 2: Specific Cell Overlapping Strategies

Replication

Each object represented in the system can be updated (written) by only one server at a time. This is the master for that object. Other server(s), which have the state of the object replicated, are the slaves for this object. After every update of the object in the master, the state is replicated to each of the slaves. Slave servers can send write requests to the master server to update the state of the object. Depending on the amounts of read/write requests made by each server, the master is selected dynamically, out of the servers that maintain a copy of that object.

By assigning a single unique master for each object, object locking, which requires large amounts of inter-server communication is prevented. However, since the slave copies can be considered as caches of the object on remote servers, cache coherency issues can arise if the object is not quickly updated on the remote servers. Using fast interconnects found in HPC's should mitigate this problem.

When a user is inside an overlapping region between two cells (of two servers), the decision as to which server the user connects is determined by considering the current load of the two servers. Note that the server to which a user connects is responsible for performing all the game logic, authentication and integrity checks for that user. Also note that moving the user from one server to another in this situation does not require moving any code or data. This is because the user object is already replicated on all servers maintaining the overlapping region and that each server maintains an instance of the client (e.g.,

game). However, in the rare occasion that a user teleports between two spatially distant regions, moving the user object accordingly is necessary. The latency should not hinder the user experience because users naturally expect a small delay when teleporting.

User Migration across Cells

Overlapping of cells allows users to move from cell to cell in a seamless manner as depicted in the figure 3 below. Cell overlap also avoids thrashing when a user is on the border between two cells and moving between cell boundaries.

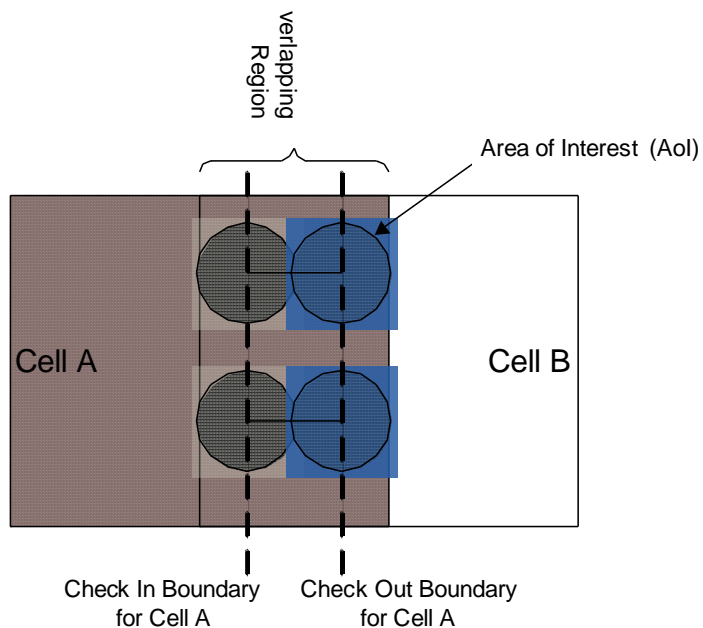


Figure 3. Cell Overlap and Area of Interest (AoI) for Transition Between Cells

As seen in the figure above, users are moved to the new cell as soon as its Area of Interest (AoI) is no longer fully contained inside the current cell. As mentioned earlier this migration does not involve moving any code or data, therefore resulting in better user experience. By selecting the sizes of the cells and the radius of AoI appropriately (i.e. the AoI size should be smaller than the cell size), implicit check-in (CI) and check-out (CO) boundaries are formed. This prevents the frequent user migration between servers due to user movement around cell edges, which can degrade the user experience. Nyquist sampling theory can be used as a consideration when selecting AoI size with respect to cell size.

As previously shown, in the tight overlapping case with square cells, there are eight different cells overlapping with each cell. We can number these cells 1-8. In the case of hexagonal cells, there are only six cells overlapping with each cell. These are depicted in Figure 4 below.

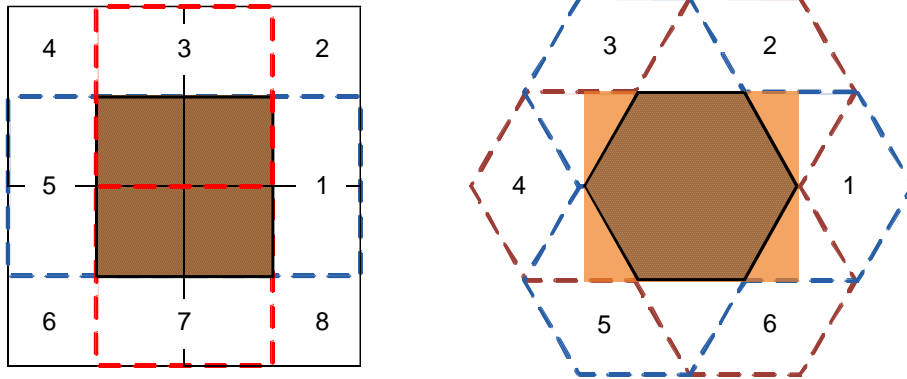


Figure 4. Coverage Using the Tight Overlapping Cell Paradigm

When a user moves near a cell border and his AoI starts crossing the border, a new (overlapping) cell to migrate the user must be determined. This can be done by partitioning the cell into various regions and determining the new cell depending on the region the user is located. One method for cell partitioning is shown in the figure 5 below. The topography of the adjacent cells could be considered in boundary topology. For example if an adjacent region is mountainous, the cell boundary abutting this region could be made small or a wide angle because avatars would not be able to venture in this area and other regions may benefit from a finer migration strategy.

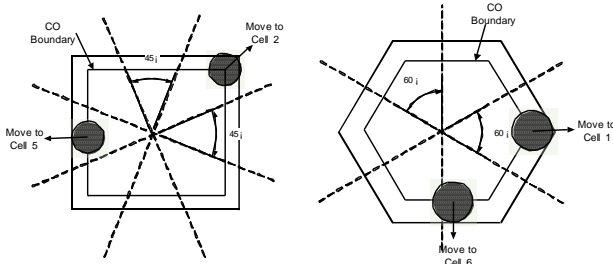


Figure 5. An Example of Partitioning for Entity Migration Between Cells

Allocation of Cells

Each server is responsible for an area made up of multiple cells (possibly neighboring). When allocating cells, the system should try to allocate spatially neighboring cells to the same server to reduce the overhead of user migration. Note: Neighboring cells are defined to be the cells sharing a common border, which is different from overlapping cells.

To address the redundancy requirements, certain restrictions are necessary to ensure variation of servers in overlapping cells. In the case of loose overlap, the system must ensure that cells are allocated to servers such that no two overlapping cells are stored in the same server. A graph can be built such that a vertex in the graph represents each cell and each edge represents an overlapping between the two cells. Graphs for each cell shape currently being considered and overlapping degree are shown in Figure 6 below. Figure 7 shows an expanded lattice structure.

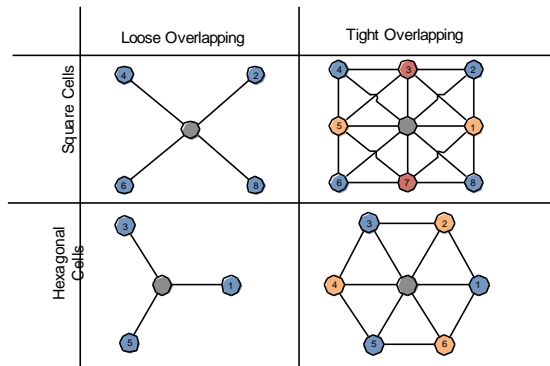


Figure 6. Basic Graph Coloring Representation of Cell Overlaps

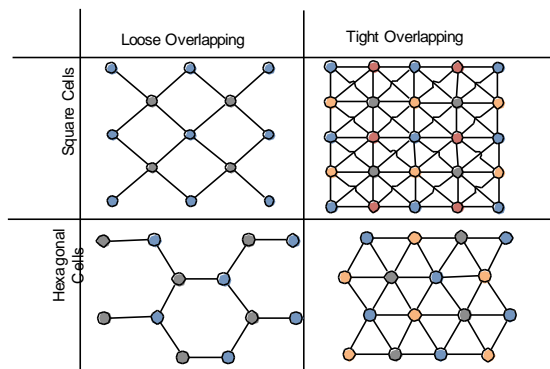


Figure 7. Sample Cell Lattice Structures

Different colors represent the different servers where each cell is allocated. These graphs create lattices as shown above. We call these lattices Cell Allocation Graphs (CAG). For the loose overlap restriction, the problem of allocation is a graph coloring problem. Though graph coloring is NP-hard in the general case, the regularity of these lattices makes the lower bounds for such colors quite clear. From analyzing these CAGs, it is evident that in the loose overlapping case we need at least two different servers to

ensure that no overlapping cells are allocated to the same server. According to the above scheme we need a minimum of 4 and 3 servers respectively to handle tight overlapping cases for square and hexagonal shapes. The use of hexagonal cells is superior to square cells since the replication overhead and resource usage are lesser. However, the use of square cells is more straightforward to implement from both software and geometry perspectives.

While the above scheme satisfies requirement (1) of cell allocation, it is a bit too restrictive in the tightly overlapping case. This is because overlapping cells can be allocated to the same server as long as the overlapping portion is also replicated in at least one different server. This modification will increase server utilization and reduce replication overhead since the overlapping area is now present in fewer servers, and according to requirement (4) a lesser number of copies are maintained.

Using this new scheme of allocation, at one extreme the whole world would be replicated on just two servers, each maintaining a copy of the world (in case the world is very sparsely populated). At the other extreme, each cell in the world would be allocated in a different server (in case the world is very densely populated). However, the weaker restriction afforded by tight cell overlap is not a traditional graph coloring problem—consequently, a more general view is required.

Dynamic Cell Allocation Algorithm (DCAA)

Cell allocation can be initialized to one of the extreme CAGs or CAGs shown in the above figure.

Whenever a server becomes overloaded, the DCAA algorithm is invoked to balance the load. It achieves this by moving one of the cells in the overloaded server to a less loaded server. Two decisions need to be made in this regard:

1. Determine which cell to move;
2. Determine where to move it (to which server).

Which Cell to Move?

- The cell that is moved should have a significant impact on the load of the current overloaded server.
- It should mitigate thrashing after moved to a new server.
- The moved cell should have little interaction with other cells in the current server.

Where to be Moved?

- The server should be under-loaded or an idle server.
- The movement should satisfy requirement (1).
- The moving cell should have high interaction with other cells in the selected server.

Interaction between cells can happen due to few reasons:

1. Overlap between Cells – The objects written in one cell need to be updated in the other overlapping cells.

2. Neighboring Cells (cells that share a border) – When a user comes closer to a border (in the loose overlapping case), his AoI might intersect with the neighboring cell, requiring the access of objects in that cell.
3. Non-Spatial Communications – There could be objects in the client (e.g., game) world that require communication between distant cells (e.g. teleconferencing object).

Whenever an object in a particular cell accesses or updates an object in a different cell, the interaction metric of both cells is updated. This metric implicitly includes the interaction happening between overlapping cells. This metric is then used in making both of the decisions: which cell to move and where to move it.

Load Balancing

According to the discussion above, load balancing in this architecture is achieved by following methods:

- Distribute reads/writes among servers;
- Dynamically change the number of cells assigned to servers.

Server/Cell Management

One server in the cluster is selected (by a voting scheme) as the Meta Server (MS), which is responsible for the following:

- Determining server crash (by heartbeat or polling) and recreate crashed servers content from replicas;
- Dynamically allocating cells to servers:
 - 1) Implement the Dynamic Cell Allocation Algorithm (DCAA),
 - 2) Load balancing by adding cells to idle servers or removing cell from overloaded ones (explained below);
- Initial allocation of cells to available servers;
- Adding or removing servers and maintaining a list of available set of servers;
- Act as a proxy for external management or querying tasks (e.g. data mining).

Every time cell allocation changes, the Meta Server notifies the affected servers about their new neighboring servers. This way, when a player comes near an edge, the zone server knows who to contact, without asking the Meta server (a bottleneck at Meta server is prevented).

Each server maintains a heartbeat with the MS (for presence-monitoring in case of server crash), which is also used to communicate its current load to the MS (for load balancing). In case the MS fails, each server is therefore able to determine this failure immediately and start voting to select a new MS.

Voting Scheme for Selecting a New MS:

- 1) Each server sends its recent load (percentage) to every other server via broadcasting in IB or multicasting in IP.
- 2) After receiving the information from all the servers, the server with the least load declares himself as the new MS. Thresholds are established, to avoid thrashing of MS shifts.
- 3) All the other servers send their cell allocation information to the new MS. The MS then builds a map of current cell allocation.

A general server crash or a MS crash will not cause any interruptions to any users except for those who were directly connected to the crashed server. Even those users will only experience a slight delay until they are reconnected to new servers. The simulation state will not be affected and the content will be recreated by the MS ASAP.

Implementation Details at UCF on STOKES

An IBM Blade server on STOKES supercomputer is assigned to each of the servers in the above discussion. Each server is made up of three components each performing different tasks.

(One Game server + One MemCache + One DB server) = A server (a blade) responsible for a cell(s)

A server should have hostname that can be used to uniquely identify that server for communication purposes. A server can contain multiple CPU and processing cores. Therefore it is necessary to provide multithreaded implementations for the above three components to exploit the full hardware capabilities.

The overall architecture is shown in Figure 8 below and each component is explained in the sections that follows the figure.

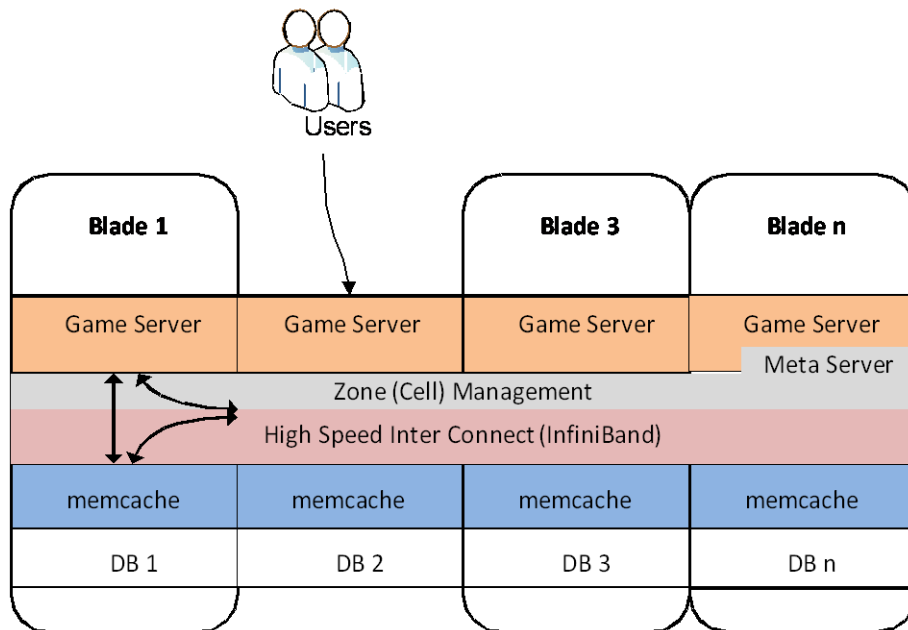


Figure 8. Implementation of Distributed Architecture on HPC (UCF STOKES)

Game Server

This contains a copy of the game logic. Game server is responsible for the following:

- Maintaining IP connections with the users and communication with users (user identification and authentication can be offloaded to a separate authentication server – such as the User Nodes of STOKES);
- Performing the Game logic operations in response for user events;
- Determining the reads and writes needed to fulfill user events and communicating these with the Zoning layer below.

Zoning Layer

This layer contains the Meta Server (active only in one server) and the logic required to communicate with overlapping cell servers and MS, and the MS voting scheme. This layer is the main focus in this document. This layer uses the star topology high speed IB interconnects to communicate among servers.

MemCache or Caching layer

Memcached servers are deployed on each of the Blades in between the Zoning layer and the DB. The large memories available on Blades are thus exploited to reduce the disk read/write frequency. This would allow the full utilization of the available BW of IB without disk access becoming a bottleneck.

Note: *memcached* servers are deployed independently of each other – the only communication between the *memcached* servers is via the Zoning Layer

Database Servers

The database servers can run low overhead, embedded databases since SQL support is not necessary when the objects are accessed directly by a key. However, administrative queries can be simplified if a relational database is used. In either case, like the *memcached* servers explained above, the DB servers deployed on different servers do not communicate with each other directly nor implement any replication functionality. The DB tables are partitioned horizontally among the servers. Specific to STOKES, the DB is stored in a non-shared portion of the GPFS file system.

Current Status of Research

DARKSTAR was hosted and evaluated on UCF STOKES. The architecture described in this paper is currently being coded.

References

- [1] A Microcell Oriented Load Balancing Model for Collaborative Virtual Environments, Dewan Tanvir Ahmed, Shervin Shirmohammadi
- [2] Zone Based Messaging in Collaborative Virtual Environments, Dewan Tanvir Ahmed, Shervin Shirmohammadi, Ihab Kazem
- [3] HYMS: A Hybrid MMOG Server Architecture, Kyoung-chul KIM, Ikjun YEOM, and Joonwon LEE
- [4] Traffic Characteristics of a Massively Multi-player Online Role Playing Game, Jaecheol Kim, Jaeyoung Choi, Dukhyun Chang, Taekyoung Kwon, Yanghee Choi, Eungsu Yuk
- [5] Game Traffic Analysis: An MMORPG Perspective, Kuan-Ta Chen, Polly Huang, Chun-Ying Huang, Chin-Laung Lei
- [6] <http://scalamo.sourceforge.net/> ScalaMo: A scalable MMOG architecture
- [7] Improving the Performance of Distributed Virtual Environment Systems, P. Morillo, J. M. Orduña, M. Fernández, and J. Duato
- [8] VELVET: An Adaptive Hybrid Architecture for VErY Large Virtual EnvironmenTs, Jauvane C. de Oliveira, Nicolas D. Georganas
- [9] Improving Gaming Experience in Zonal MMOGs, Dewan Tanvir Ahmed, Shervin Shirmohammadi, Jauvane C. de Oliveira
- [10] RTF: A Real-Time Framework for Developing Scalable Multiplayer Online Games, Frank Glinka, Alexander Ploß, Jens Müller-Iden, and Sergei Gorlatch
- [11] A Multi-Server Architecture for Distributed Virtual Walkthrough, Beatrice Ng, Antonio Si, Rynson W.H. Lau, Frederick W.B. Li
- [12] <http://www.worldofwarcraft.com/index.xml>
- [13] <http://secondlife.com/>
- [14] <http://www.projectdarkstar.com/>