QuakeD: A Distributed Quake Game

John Jackman, Xia Liu, Dan Swanton, Brian Wu
Massachusetts Institute of Technology
{jdj3, xialiu, swanton, brianwu}@mit.edu
May 12, 2005

Abstract

This paper describes the motivation, design, implementation and performance of QuakeD, a distributed Quake game. The goals of QuakeD are to allow thousands of people to play in a single game and to minimize degradation of gameplay. The first goal is achieved by dynamically adding and removing servers as the total load increases and decreases. Careful server selection and redundant servers achieve the second goal.

1 Introduction

Quake 2 was a popular first person shooter computer game developed by id Software. By default, Quake 2 allows a maximum of 32 clients to play together in a single game by connecting to the same dedicated host server. Clients communicate their actions through UDP to the host server, and the host server is responsible for keeping track of the world state and communicating this state to clients. In December of 2001, id Software released Quake 2's source code under the GPL[1].

Our first goal was to make a Quake game that scaled very well for hundreds or thousands of players. Playing Quake with such a large number of users leads to an entirely new experience, especially for coordinated team play. For example, consider the popular Quake 2 variant "Capture the Flag." In this game-type, the players are split evenly into two teams: red and blue. Each team has a home base that contains their flag. Teams score by stealing the other team's flag, and then safely returning to their own base with the opponent's flag in their possession. In a typical 10 person game of capture the flag, each team will have about 5 players, and about half of these will be attempting to retrieve the enemy's flag at any given time. Although these players would ideally work together to achieve their goal, it is often simpler given the small server population to simply grab the enemy flag and make a quick solo sprint back to base. This strategy is effective because there are only about 5 enemy players to run past. In contrast, imagine how such a game would change if there were 500 red players and 500 blue players in a huge world with just one flag of each type. Players would be forced to work together and travel in groups. A large, distributed Quake game would have interesting team-play implications.

Our second goal was to minimize the degradation of Quake's gameplay. First person shooters have the following challenging design constraints:

Latency: Players will notice latencies as low as 10ms and will not tolerate additional "lag."

Familiarity: Players have grown attached and accustomed to certain favorite maps.

Density: Players expect maps to be neither overcrowded nor sparse.

Our design takes these constraints into consideration in creating a massive, distributed world. Although Quake 2 was used for our prototype, we expect that a similar solution could be applied to other games where having a single centralized server is either infeasible or undesirable.

Section 2 presents a high-level overview of our system design. In Section 3 we detail of our architecture. In Section 4 we will describe our prototype implementation. Section 5 presents the results of this prototype. Section 6 reviews other work related to distributed

computer games. Section 7 concludes.

2 Design Overview

We modify Quake by using a dynamic grid of servers that consists of the players' host machines. The world is partitioned into *zones*. Each zone is run by one server, and consists of one normal quake map. Zones contain between 0 and 32 players, although there are usually around 15 players in a given zone. In the special case where there is only one zone server in the grid, the experience is exactly the same as the traditional single server online game, with the one exception that one of the player's machines is hosting the server, instead of a dedicated server.

The zones are arranged in a grid with connections between adjacent zones.

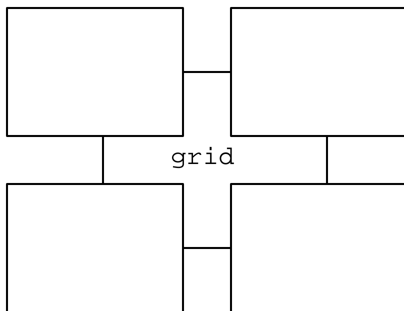

Figure 1: Zones and Connections

In Figure 1 above, there are 4 fully-connected zones. Each zone is controlled by a different server. These servers are chosen from among the player's machines, but there is no special relationship between a player and the zone he is serving. In other words, it is perfectly legitimate for a player to not be present in the zone he is serving.

The grid is overseen by one central entry server. The entry server knows the IPs of the client's acting as zone servers, as well as the zone connections in the grid. The entry server is responsible for inserting new players into the grid and maintaining the grid's integrity.

3 Design Description

3.1 Zone Connections

Zones are connected by zone-teleporters. Zones are aware of their connections to neighboring zones, as well as the IPs of the servers for those zones. We try to minimize the time required for players to transition from zone to zone.

Teleporters are existing Quake 2 entities that instantaneously move players from one point on a map to another. Teleporters can be identified by the yellow sparks that they emit. Zone-teleporters look similar, except that they are distinguished by the bright green sparks that they emit. When a player steps into one of the zone-teleporters, their screen temporarily goes black, and they reappear at a predetermined location in an adjacent zone. One nice feature of the zone-teleporter connection is that they will minimize zone-to-zone communication. The only objects that will cross zone-teleporter connections are players.

We ensure a quick transition between zones by preloading the next zone as a player approaches the zone-teleporter. When a player gets close to a zone-teleporter, he automatically starts loading the map and entity contents of the zone led to by this zone-teleporter. This information is stored in a temporary location so that it can be quickly accessed if the player enters the zone-teleporter. Furthermore, this information is replaced *lazily*. That is, it is loaded until completion, and then only replaced if the player enters a different preloading area. This ensures that players who continually enter and exit the same preloading area do not suffer drastic performance consequences.

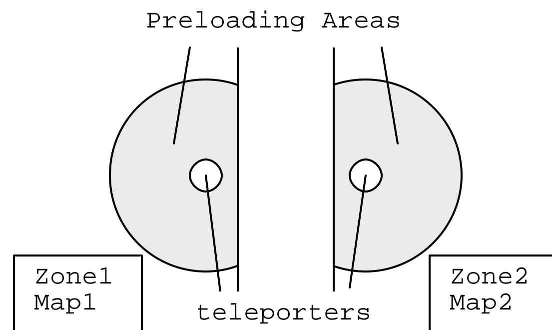

Figure 2: Preloading Areas

In the example above, if a player from

Zone 1 enters the preloading area for Zone 2, he will begin to load Map2 and the player models in Zone2. Therefore, when he steps in the zone-teleporter, Zone2 will already be loaded and can be immediately used. All that remains is for the player to begin to receive updates from Zone2's server. If a player steps into a preloading area, but does not use the zone-teleporter, and eventually leaves the preloading area, he will retain Zone 2's information until he enters a preloading area for a different zone. We have strategically placed the zone connections so that no two preloading areas are in close proximity of each other.

We considered tunnels as another possibility for connecting zones. Tunnels would be enclosed physical spaces with bends to minimize the area in which a player from one server can interact with a player from another server.

tunnel
Zone1
Zone2

Figure 3: Example Tunnel

Tunnels as zone-connections would provide the attractive feature that transitions between zones could be literally seamless. Unfortunately, the 3D geometry of the standard Quake 2 maps makes this prohibitively difficult. In order to use tunnels to connect maps on our grid, each map edge would need to match up exactly. Any map that was intended to be used in our grid would need to have its geometry modified so that all possible tunnel zone-connections were at the same height, and in the same relative place on the map. Alternatively, a large and tile-able map could be created, which would fit nicely with tunnel zone-connections. However, one of our initial design goals was to unobtrusively support the traditional maps provided with Quake 2. Zone-teleporters allow us to use these maps in our distributed grid with minor modification: instead of changing the geometry of the maps, we simply introduce additional zone-teleporter entities.

3.2 Zone Servers

Zones are hosted by the players. The initial zone server is the first player to join. The entry server creates new zones as necessary and chooses a player to act as the new zone server. The entry server chooses the player in its current zone with the lowest ping who is not already a server. It also takes into account the player's past reliability as a server, as described in Section 3.7. When the player serving a zone leaves the game, a new best server is selected in the same way. Changes in who is serving a zone are communicated to the entry server and to all connected zones. Ultimately, there will be *n* zone servers hosted by the players with approximately the *n* best pings.

The above design specifies that zone servers elect new players to serve based on the latency between the player and the zone server. One alternative we considered was to have each player ping each other, and then select the player with the lowest average latency to all players in the zone. This design would have the advantage of selecting zone servers that are, on average, best for the other players in the zone. However, this would also introduce a high communication overhead between players. Ultimately, we decided that the ping between a player and a zone server is a close enough approximation to the efficacy of server that it is not necessary for each player to ping all other players during server selection.

Another design that we considered was to have the zone servers hosted not by the clients, but instead by a trusted cluster of servers. The big advantage of this design is that no modifications would be necessary to the clients in order to play in a distributed game. Instead, all modifications for distributed play would be made on the server side. This would mean that any owner of Quake 2 would be able to simply connect to one of our distributed servers. Another advantage of using trusted servers is that security and zone server failures become less of a concern since they can be

maintained by a system administrator. A disadvantage of this approach is that the world could only grow in size proportional to the number of available servers. In the end, we decided that these advantages were sufficiently strong to justify implementing a server-side-only version. The type of distributed game to start is now a command line option that can be specified upon entry server launch. Throughout the rest of this paper, however, we will refer only to the peer-to-peer zone server framework. Although it is true that using a trusted server cluster for hosting zone servers has advantages, we are also interested in providing a workable peer-to-peer solution.

3.3 Load Balancing

The primary load balancing mechanism is the redistribution of dead players. Death is a frequent occurrence in Quake, and even more frequent when there are large numbers of players near each other.

When a player dies, a one-step greedy algorithm will attempt to relocate the player to a more optimal zone. We estimate that the ideal number of players in a map is around ten. The spawning algorithm looks at the zone where the player died, and also at all neighboring zones that are connected to it. The algorithm chooses to spawn the player in the zone with the number of players closest to the ideal. For example, if the zone the player died in has seven players, and the two neighboring zones have eleven and fourteen, the player gets spawned in the zone with eleven. This algorithm alleviates overcrowding because when there are many people in a room, there are frequent deaths, and these dead players are repopulated in less crowded areas. Another advantage of this algorithm is that does not spawn players in nearly empty zones. This is important not only for the players to maintain a high level of action, but as we will see later, it is also important for dynamic zone removal.

An alternate design that we considered was to have the entry server keep a rough estimate of the population of each zone, and then use the information from all zones to decide where to spawn the player. The problem with this approach is that every death would require a decision from the entry server, which limits the scalability of our system. Another alternative that we considered was a multi-step version of the given algorithm. Instead of stopping the algorithm after at most one hop, the algorithm could be repeated until a local optimum was found. We decided not to do this, however, because player deaths and subsequent rebalancing is so frequent that this is not necessary. Furthermore, when the player did travel multiple hops, it would delay the player spawn and consume server resources.

If the spawning algorithm finds itself in a situation where all considered zones have twice ideal the number of players, it communicates to the entry server that a new zone should be created. The entry server is responsible for using the zone connection information to create this new zone on the edge of the grid. The entry server also decides on the new zone-connections, and communicates all changes to the neighboring zones. This new zone is soon filled by the spawning algorithm, because this new zone has a population of zero, which is closer to ideal population than the overcrowded nearby zones that have at least twice the ideal number of players.

If a zone is empty, the zone server communicates to the entry server that it should be considered for removal. The entry server removes the zone as long as removal does not result in a disjunction in the grid. Because the entry server knows the zone connections between all zones in the grid, it can easily make decisions necessary to maintain grid integrity. When there are on average fewer than ten players per zone across the grid, the spawning algorithm will cluster players into zones of ten, and the excess zones will be discarded.

3.4 Entry Server

The entry server is responsible for coordinating the entire QuakeD system. It needs to assign players to become zone servers, and keep track of their statistics (see section 3.6). Zone servers are chosen based on their pings

and their past history as a zone server.

The entry server is also required to keep state on the connections between zones (see section 3.1). Whenever a zone server crashes or becomes unreachable, the entry server needs to confirm the backup zone server as the new zone server (see section 3.5.1). If both the primary and backup zone servers fail simultaneously, the entry server must notify the neighboring zone servers, and modify the zone connections so that all of the zones are still connected.

The entry server is also responsible for spawning new players. When a player wishes to join QuakeD, he first contacts the entry server, who then contacts an appropriate zone server. Then, that chosen zone server contacts the new player, and the player enters that zone. The entry server is really at the heart of our design, and it helps coordinate everything. If it goes offline for any reason, there is a backup entry server, as described in section 3.5.2.

3.5 Fault-tolerance

3.5.1 Backup Zone Servers

QuakeD relies on many potentially unreliable players to host the zone servers. We considered implementing a backup zone server for each zone server, which would attempt to keep an exact replica of the state kept on the primary zone server. As soon as a zone server is selected by the entry server, that zone server would elect its own backup zone server. From then on, the zone server would simply forward the backup zone server all of the relevant packets the clients in that zone and from the entry server. If the backup zone server does not receive a packet from the zone server for a specified timeout period, then the backup zone server assumes that the zone server has crashed or lost its connection. The backup zone server then contacts the entry server, and becomes the primary zone server. If the backup zone server is unable to contact the entry server, then it assumes that there has been a network partition, and continues trying to contact the entry server. After being confirmed as the new primary zone server, it elects another backup, and contacts all the clients currently in the zone.

The entry server does not have any knowledge of the backup servers, since the zone servers are the intermediaries between the entry server and their own zone servers. When a zone server crashes, the backup tells the entry server the primary's IP, and then receives confirmation to become the primary zone server including the IPs of the adjacent zones. A second or two of play could get "undone" from the players' perspectives, or there might be some unusual lag, but the players would be able to continue playing without significant problems on the new zone server.

If both the primary and backup zone servers crash at the same time, then all of the clients timeout, and contact the entry server. The entry server modifies its representation of the zones to account for the crashed zone server, and respawns the clients as if they had all died. The entry server then selects a new zone server (if necessary) to take the place of the crashed zone server. The new zone server then selects its backup as normal.

In a normal Quake game, players die very often, sometimes multiple times a minute. The problem of reliability could be circumvented by simply allowing the clients to timeout, and subsequently contact the entry server to be respawned.

3.5.2 Backup Entry Server

In a normal Quake game, there is a single dedicated server which can handle up to 32 simultaneous players. However, if that server goes down, the game stops completely. In QuakeD, hundreds or even thousands of players can participate in the same game, but the entry server is still a single point of failure. In order to achieve reliability, we decided to take a similar approach to the entry server as we did to the zone servers. There is a backup entry server, which maintains the same state as the primary entry server. The primary entry server simply forwards copies of all the packets it gets to the backup. If the backup entry server does not receive a packet from the primary entry server

for a specified timeout period, then the backup entry server tries to contact all the zone servers. If a majority of the zone servers report also noticing that the entry server is down, then the backup entry server promotes itself to be the primary entry server. When the crashed entry server is restored, the current primary entry server sends all of its state to the restored server as fast as is possible without disrupting gameplay. The effects on the game of the entry server crash are similar to that of a zone server crashing; there is a long pause which can be attributed to lag. Also, players that need to be spawned need to contact new primary, and therefore experience a longer than normal pause before spawning.

3.6 Security

Security could be a major issue in QuakeD, since our design relies on players in the game to host zone servers. We could try to complicate our system by attempting to prevent players from cheating, but our efforts would be easily circumvented since we need to release our source under the GPL. Instead, we chose to allow players to complain about zone servers that they suspect are cheating.

The entry server keeps state for each zone server IP. The persistent state consists of the total amount of time that the IP has hosted a zone server, the total number of complaints, and the IPs of the clients that complained. The entry server does not accept multiple complaints from the same IP for the same zone server IP; this prevents a single person from maliciously ruining a zone server's rating. The players should realize that it is in their best interests to use this system properly; if all of the fastest zone servers get poor ratings, then slower hosts will become zone servers and the quality of gameplay decreases. When choosing zone servers, the entry server takes this state into account as well as the ping times. Zone servers with the highest pings, fewest complaints and most time spent hosting are the most likely to be chosen as zone servers in the future.

This design recognizes that complete security in an open-source distributed game is an unrealistic goal. However, it leverages the fact that the Entry Server *can* be trusted to store information about which zone servers have been reliable hosts in the past. This system allows the players to police themselves, which should prevent at least the egregious and obstructive forms of cheating.

4 Implementation

To test this design, we implemented a prototype by modifying the Microsoft Windows version of the Quake 2 source code.[1] Our prototype showcases the basic distributed grid design, allowing players to use zone-teleporters to transition between different zone servers. Building our implementation on top of the existing Quake 2 code base introduced a number of design constraints and limitations.

For example, in order to add zone-teleporters to existing Quake maps, it was necessary to implement an entity loading system. Traditionally, data for map entities such as the location of the rocket launcher and health packs is stored directly in the ".bsp" map file. However, we needed to modify this listing to insert zone-teleporters. Our solution was to support the loading of custom entity files. When a Quake map is loaded, QuakeD tries to find a valid custom entity ".ent" file of the same name. If a custom entity file cannot be found, QuakeD creates one based on data read from the map. If a ".ent" file is found, it is used by the zone server to specify the location and default properties of all entities in a map. Under this system, inserting a new zone-teleporter into a traditional map is accomplished by editing a text file. For example, here is the definition for the north zone-teleporter in the map "The Frag Pipe":

```
...
{
"origin" "340 344 -16"
"classname"
"info_player_deathmatch"
"angle" "270"
}
{
"origin" "340 344 -16"
"targetname" "north"
"classname" "misc_teleporter_dest"
"angle" "270"
}
{
```

```
"origin" "288 352 -16"
"target" "north"
"classname" "misc_teleporter"
}
...
```

The first snippet is an original player spawn location; it is located at x=340, y=344, z=-16 and rotated 270 degrees around the z axis. We first create a zone-teleporter destination in the exact same location. This is where players who teleport *into* this zone will appear. Next, we define a zone-teleporter right next to this destination. This zone-teleporter has "north" as its target because it will be sending players one square north in the zone server grid. We provide similar custom zone-teleporter entity files for four popular Quake 2 maps: q2dm1, q2dm2, q2dm3, and q2dm7. Most of our zone-teleporters are inserted next to player spawn locations, as these are conveniently located in corners of the map.

If a player joins the game and the entry server determines that a new zone is needed, that player automatically becomes the zone server for the new zone. This was not originally part of our design (we wanted to choose based on ping times and history as a zone server), but it was simpler. When a player dies, the current zone server checks with its neighbors, and spawns the player in the zone with the population closest to the ideal ten players. If the current zone had four neighbors, then the player is spawned in one of five zones, the current zone and its four neighbors.

Our prototype achieves the basic goals of our design. A player connecting to our game is placed in a zone server. If he enters a zone-teleporter, the server will transfer him to the next zone. If he dies, the zone server will decide where in the vicinity he should be spawned. However, our implementation does have a number of limitations. While using the Quake 2 source code provided an excellent starting point, it imposed a difficult learning curve. Ultimately, half a semester of time was not enough to understand the Quake 2 code base and to implement all aspects of QuakeD's design. In particular, the backup zone server and server reliability list are not included in this prototype. Another feature we were attempting to include was relaying a player's state when a player enters a zone-teleporter. Instead, a player's default health and weapon are restored after teleporting between zones. Nevertheless, the prototype is functional, and does satisfy our basic design goals.

5 Results

Our prototype achieves the design goals of minimal additional latency, allowing players to use familiar maps, and ensuring that servers are neither overcrowded nor sparse.

QuakeD has minimal additional latency costs. While a player is in a zone, he experiences the same ping times on average that he would if the host were running a normal server. The only additional costs are during zone transitions and spawns. We found that zone transitions averaged 2.52 seconds. This was average was obtained by timing zone transitions on a Acer Ferrari 3200 laptop with a 2800+ MHz AMD processor that was connected to the zone server through MIT's 10 Mb/s Ethernet. Spawn delays are comparable to zone transition delays, as they involve at most one zone transition, and the player state does not need to be transferred.

Traditional maps are playable in QuakeD due to the custom entity loading system. The only caveat is that the entity text file must be modified to specify the location of zone-teleporters. We considered attempting to automatically decide the location of zone-teleporters based on player spawn positions, but ultimately we decided that it would be better to let someone familiar with the map place them by hand.

The spawning algorithm ensures that each zone server has close to the ideal number of players at all times. Unfortunately, this has proven to be difficult to test and to quantify. The problem is that a single zone server can handle up to 32 players. In order to give a reasonable test to the spawning algorithm, we need enough players that at least 3 zone servers are required. This translates to needing about 100 clients. Unfortunately, the client code is Windows only

and requires a graphically intensive output, so that only one client can reasonably be run per Windows computer. We attempted to recruit friends to act as testers, but we were unable to recruit enough players to fill even one zone server. One alternative we pursued to human testing was to use client-side bots for stress testing. Unfortunately, while we were able to connect a large number of bots to our system, these bots did not understand the concept of zone-teleporters. Instead, the bots remained inside the zone server where they were initially placed. Because the bots never willingly transitioned between zones, they proved ineffective for testing QuakeD under stress. As a result, the only type of testing that was easily achievable was to test the system's performance with small numbers of clients and zone servers. In this scenario, QuakeD appears to function well: playing our distributed version of Quake 2 incurs no additional latency costs except during zone transitions and spawning, and these delays average only 2.52 seconds.

6 Related Work

MMORPGs such as Everquest and World of Warcraft are existing games where thousands of users play together. The server load is distributed among a large number of host servers.

A key feature of MMORPG gameplay is that latency is not a large concern; MMORPG players can tolerate up to a second worth of lag. For example [3] showcases a distributed peer-to-peer MMORPG system with an average message latency of 150ms. This latency requirement for MMORPG's is loose enough to make the server distribution problem fairly straightforward.

First person shooter games, however, place a high priority on latency, with typical lag times of < 100ms. A fraction of a second can be a significant delay in a Quake game, so having multiple servers interacting becomes a challenging problem.

The only example of a distributed first person shooter that we are aware of is IBM's GameGrid project. In this project IBM tested its Grid technology by implementing a distributed Quake 2 server. "GameGrid dynamically partitions areas of the game map, including players and objects, onto different servers. If a player or object, such as a rocket, moves from one server to another, the first server sends the player's state—the player's name, vector, velocity, and statistics—from one server to the next." Basically, IBM had a cluster of about 30 servers that worked together to server one large game map.

A key similarity between MMORPGs and IBM's GameGrid implementation is that they use a predefined number of static servers provided by the host [2].

7 Conclusions

We have described the architecture, implementation, and performance of the QuakeD distributed Quake game. We have attempted to achieve scalability by dynamically starting up zone servers as the number of players increases. When implemented, the backup servers for the zone servers and the entry server will improve reliability.

Acknowledgements

We would like to thank Robert Morris and Athicha Muthitacharoen for their feedback on earlier drafts and for their helpful suggestions during project conferences.

8 References

[1] "id Software's Technology Licensing Program," [Online Document], Available at HTTP: http://www.idsoftware.com/business/technology/techlicense.php

[2] M. Hachman, "IBM Tests Grid with Games." *eWeek* (2003 Aug 20), Available at HTTP: http://www.eweek.com/article2/0,1759,1501261,00.asp

[3] Bjorn Knutsson, Honghui Lu, Wei Xu, and Bryan Hopkins. Peer-to-Peer Support for Massively Multiplayer

Games. Department of Computer and Information Science, University of Pennsylvania. Available at HTTP: http://pdos.csail.mit.edu/6.824/papers/p2p-mmg.pdf