The vMatrix: Equi-Ping Game Server Placement For Pre-Arranged First-Person-Shooter Multiplayer Matches

Amr Awadallah and Mendel Rosenblum
Computer Systems Lab, Stanford University
aaa@cs.stanford.edu, mendel@cs.stanford.edu

Abstract- **Today most multiplayer game servers are pre-located statically, which makes it hard for gamers to find equi-ping hosts for their matches. This is especially important for first person shooter games (FPS), which are a class of interactive games that is very sensitive to difference in ping between the participants and the hosting server. In this paper we present a novel solution, which builds on top of the classic operating systems concept of a virtual machine monitor (VMM). A VMM allows us to encapsulate the state of the game server in a virtual machine file, which could then be activated on any real machine running the VMM software. The main advantage of this solution is mainly backward compatibility, that is we can take any existing FPS game and migrate it to this platform without any code changes to the game client nor the server. Another advantage is the economies of scale for such a network since it can be shared between different games. We describe our vMatrix framework and address how to move the virtual machine game servers across the real-machines to minimize the difference in ping between all participants of a given match. We also demonstrate this solution using Microsoft's popular Halo PC game, we show that this solution does not degrade the game performance and does not require any code changes.**

Keywords: **virtual machine monitor, first person shooter, multiplayer games, ping fairness.**

1 INTRODUCTION

In this paper we describe a practical solution for enabling fair online first person shooter (FPS) gaming. One of the most important aspects for participants of a FPS match is to have equal round-trip time to the server hosting the match (also known as ping time). This is important since it dictates the frequency of state updates from the host to the clients, thus the players with lower pings can indeed see the *future* of *lagging* players and shoot them before they know it! Not only that, the lagging players have incorrect positions for the other players as their clients try to extrapolate the current state based on the last (now stale) state update they got from the server (a common technique known as Dead Reckoning). Professional players are very aware of this limitation, hence the reason why they take zigzag paths while running around to fool the extrapolation done by the gaming clients forcing the other players to shoot at false projected positions instead of their actual positions, which leads the game server to register a miss.

Previous studies in this area demonstrated that players prefer absolute pings to be less than 180ms, some of them further emphasized that relative delays between participants is more important than the absolute value of latency to the server [23] [24][25][26][27].

Today most game servers are pre-located at static nodes around the Internet, this is ok for the casual gamers whom join and leave all the time (they just pick the server closest to them). However, serious players usually form what is called a *clan*, and they purchase or rent a hosting server so that they can host the clan matches on it. Its typical that each clan gets a server local to their country, or side of the country in case of large countries like the US. So for example, in the US an east coast based clan will tend to get a server in the east coast, while a west cost clan will get a server in the west coast. This leads to the unfairness issue when the clans have matches against each other. The east coast clan will argue that the match should happen on their east coast server, and the west coast clan will argue for the reverse. The reason arguments arise is that typical round-trip-times rise from less than 30ms when playing on a server in the same coast as the clan players, to more than 130ms when crossing over to the other coast. This difference of 100ms gives a large advantage to the local clan; they literarily see the positions of other players and shoot at them before they get there. These differences can be much worse than 100ms for cross-country matches.

Its important to re-iterate that it's not the large round-trip-time (also known as ping) it self that irks players the most, it's the difference in ping that leads to the unfairness. Today's deployed solutions to this problem are un-satisfactory. For example, one solution is alternating matches between the home and away team, but this still does not change the fact that during each match the low-ping team will have a significant advantage. Another solution is searching for another clan with a server central to both of the participating clans, however this is very adhoc and most of the time such a server cannot be found since one of the participating clans still needs to have the password for the game server to setup the proper match configuration.

The solution is accessibility to game servers that are equi-ping to all participants. However, its very cost-prohibitive to install copies of each game on servers all over the Internet, rather we need an economical solution that allows us to share

the hardware easily between different games (and even other vMatrix applications that are not gaming related).

The main reason leading to hardship moving game servers in and out of hardware servers is the dependencies that the server code has on operating systems, libraries, third party modules, server hardware, and even people (the game administrators). Simply copying the code of the game server is not possible since the target machines need to have exactly the same environment for the code to run unchanged, which is not practical. The library versions and patches that work with one game server might cause another server to fail.

We propose a novel backward-compatible solution that builds on top of the classic OS concept of a *Virtual Machine Monitor (VMM)* [2] (refer to Appendix A for a brief review of Virtual Machines). The observation we make is that a VMM virtualizes the *real machine (RM)* at the hardware layer (CPU, Memory, IO), and exports a *virtual machine (VM)*, which exactly *mimics* what a real machine would look like. This allows us to encapsulate the state of the entire game server in a VM file, which could then be instantiated on any RM running the VMM software. This solves the software dependencies problem since the whole service is transferred with the OS, libraries, code, modules, and code that the service depends on. It solves the hardware dependencies problem since the VMM can *lie* to the overlying OS about the hardware resources available to it (e.g. memory size), hence mimicking the same hardware environment for that service regardless of the real hardware of the hosting real machine (though there might be performance degradation). It also solves the people problem since the VM capsule can include the passwords and access rights that the game administrators need to manage the server and setup the proper matches.

Hence the problem is reduced to equi-ping placement of large VM files within a network of RMs running the VMM software; we call this network of virtual machines *the vMatrix*[1].

We do not attempt to build a VMM, but rather we reference existing software for the x86 architecture from VMware, Inc. [1]. Note that similar VMM software is also available from Microsoft [5], and an open-source version is available as Xen [28]. We choose VMware due to their close relationship with Stanford University, however we think other VMMs will be just as useful.

In this paper we present our framework in detail and briefly address how to place the virtual machine game servers services across the real-machines to minimize ping difference between the participating players without degrading the server performance (in terms of latency, throughput, and availability). Another challenge that we touch on is how to avoid making any significant architecture or software changes to existing game servers so that this solution is backward compatible.
We claim that the distinguishing advantages of our approach are the combination of:

1. Equi-ping placement of game servers to minimize round-trip-time difference between participants;
2. Backward compatibility leading to zero cost for converting existing game servers to run within our framework;
3. Economies of scale by leveraging the fact that this network can be shared among many different types of games rather then being fully dedicated to one game.

In two previous papers [8][9] we covered additional advantages of the vMatrix platform, which include:

1. Leveraging the migration aspects on an Internet-scale to achieve Dynamic Content Distribution;
2. Enabling server-multiplexing to reduce total cost of ownership;
3. Providing quick re-activation of servers to reduce mean recovery times in cases of software crashes;
4. Absorbing flash crowds by on-demand replication of servers.

In section 2 we illustrate a motivating example. In section 3 we present the vMatrix framework. In section 4 we discuss the vMatrix implementation details. In section 5 we discuss our experiences with migrating the Halo PC game server to the vMatrix platform. Finally in sections 6 and 7 we cover related work then conclude.

2 MOTIVATION

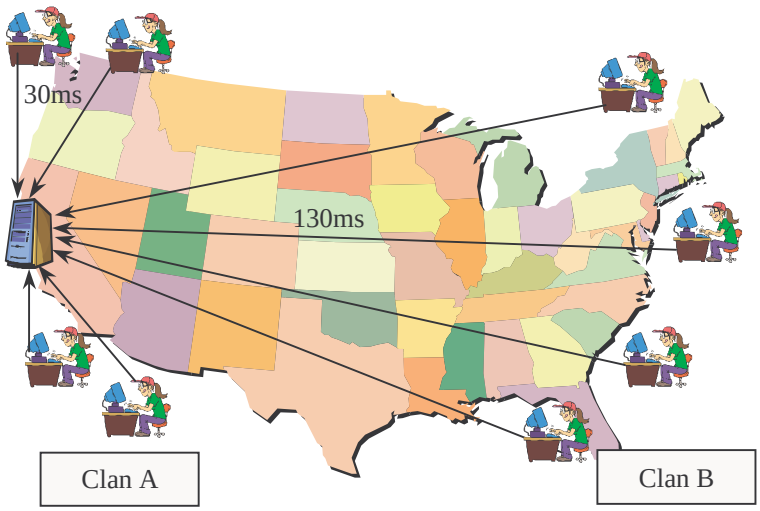


Figure 1: Today's static server placement creates unfairness for First-Person-Shooter Clan matches.

[1] The name "*The vMatrix*" comes from the analogy to the 1999 sci-fi movie "*The Matrix*". In the movie, machines controlled humans by virtualizing all their external senses; we propose doing the same back to the machines! It is a virtual matrix of real machine hosts running VMM software, which are ready to be *possessed* by guest VMs (*ghosts*) encapsulating Internet services.

The scenario illustrated in Figure 1 shows what happens in today's world of statically placed first-person-shooter game servers. This occasion arises frequently when clans (teams) from different countries are to play each other, it can also happen within large countries like the US as illustrated. In this scenario clan A will typically get low pings to the server, on the order of 30ms round-trip-time, while clan B will get pings on the order of 130ms or more. This leads to a ping differential of 100ms, which allows clan A players to see the future of clan B players, hence they can start shooting at them before clan B players detect their presence.

In such situations it is not uncommon for clan B to call off the match as most gaming organizing comities (e.g. Cyber Athlete League [29]) stipulate that ping differences of more than 90ms can lead to voiding the match. The clans struggle to find an equi-ping server, which is not always possible since they need to have administration access to that server to setup the proper match configuration.

Its important to note again that unfairness is not due to the large absolute value of the ping, but rather the large average ping differential between both clans.

The game servers are usually statically separated like this because it's very hard for one clan to own more than one server in different places spread across the Internet. There are a number of new renting services that allow clans to rent a server for a few days, but these are still not very economical since they require at least one day of rent, as opposed to a few hours for a single match. Also these renting services do not currently reveal much about the server location and whether it will be fair for all the participants.

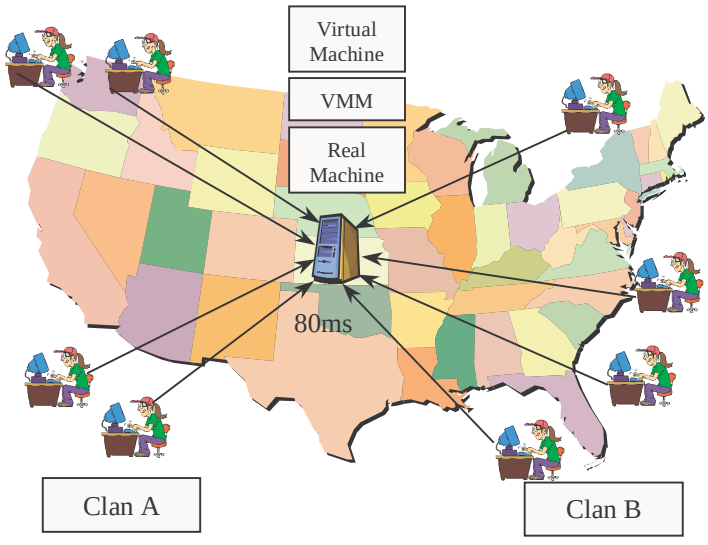


Figure 2: Dynamic game server placement using the vMatrix infrastructure lets us place the server at an equi-ping node with respect to all participants.

Within the vMatrix we add a VMM separation layer between the VMs (carrying the OS and code for the services) and RMs (which are the shared resource). Now with the vMatrix renting service, an equi-ping RM can be allocated then the proper VM carrying the needed game server can be copied over to it and instantiated. In contrast with the same snapshot we represented in Figure 1, which lead to a ping differential of 100ms, Figure 2 illustrates how we can have an equi-ping of 80ms to all participants, bringing the ping differential down to zero.

3 THE VMATRIX FRAMEWORK

The vMatrix is a network of real machines (RMs) running virtual machine monitor software (VMM) such that virtual machine files (VMs) encapsulating a machine for a given game server can be activated on any RM very quickly (on the order of seconds to minutes depending on the underlying storage and network infrastructure, e.g. local hard-disks versus a fiber-optic Storage Area Network).

3.1 Main Components

The basic framework for the vMatrix is illustrated in Figure 3. There are 3 main clusters:

1. The *Production Cluster*: this is where the VMs are instantiated on dedicated RMs dispersed over the Internet. The Oracle, as described below, picks the optimally placed RM to achieve equi-ping.

2. The *Loading Chambers:* this is where the VMs are instantiated for maintenance purposes. While in this state the clan administrators can connect to the server, install the game server software, configure the server properly, etc. Note that since the servers are not getting operational load yet, we can have more than one VM sharing the same RM.

3. The *Hibernation Nest*: this is simply the backend storage for keeping all the VM files in dormant suspended state until needed. The VMs are not accessible in this mode (neither for administration, nor for operational load).

The *Oracle* is the program responsible for maintaining the state of all VMs and RMs and it supervises the vMatrix network. As new RMs are added to the network and loaded with the VMM software, they are subscribed with the Oracle. Similarly, whenever a new VM is created it is registered with the Oracle. The Oracle is also responsible for the matching of VMs to RMs and copying the VM file to that specific RM then activating it.

In our simple prototype, the Oracle is a Perl script that reads configuration files listing all available RMs and VMs. The Oracle communicates with the RMs to copy VM files from the storage to them (using `scp`), and communicates with the VMM server software on each RM to boot or suspend VMs (this is done using the VMware Perl API).

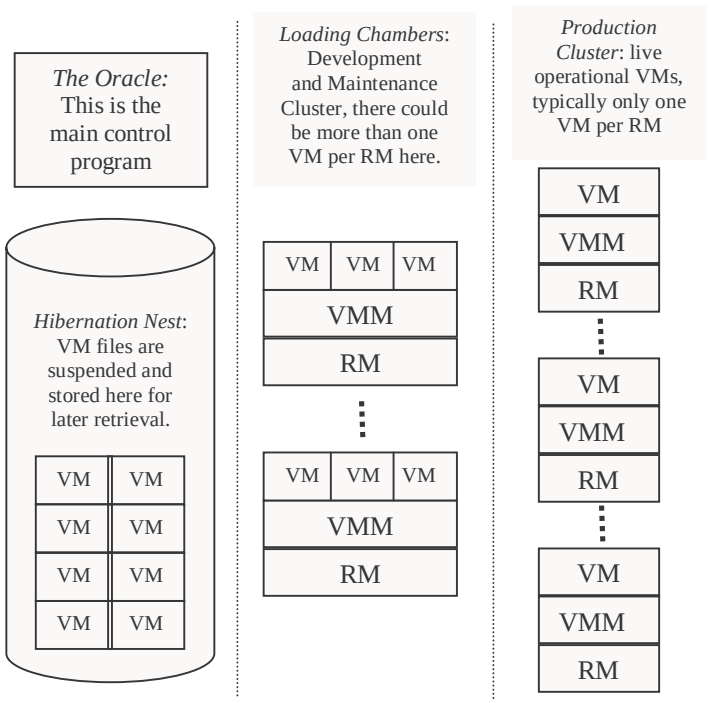
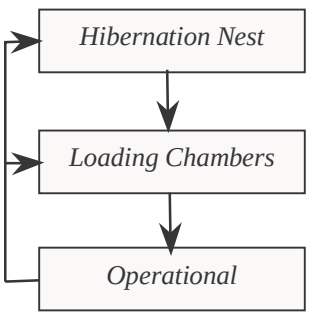


Figure 3: The vMatrix Framework

3.2 VM Server Lifecycle

The simple state diagram shown in Figure 4 describes the lifecycle of a VM Server:



1. A large number of virtual server files are stored as dormant files in a SAN or NFS server. They can be in frozen pre-booted state for fast re-activation.

2. A number of virtual servers are activated in shared RMs (i.e. more than one VM per RM) so game administrators can maintain them.

3. A virtual server is activated on a dedicated RM in preparation for a match to begin.

Figure 4: Lifecycle of a Virtual Server

4 IMPLEMENTATION DETAILS

As contrasted to previous work (which we cover in section 6), we claim that this solution presents the smallest switching cost for porting an existing game server into such a dynamic allocation network (i.e. backward compatibility) and at the same time it achieves the advantages of equi-ping allocation and economies of scale. In section 5 we illustrate this ease of conversion through a real-life example.

4.1 Equi-Ping Game Server Placement

The game server VMs need to be instantiated on an RM that is equi-ping to all participants, however participants are usually not known until match time. The straightforward approach to get latency information is to build ping profiles from each participant to all available RM servers just before the match begins. This solution does not scale very well if we have a large pool of servers to choose from. A ping topography map for the Internet is needed; such database should be able to return the ping difference given two IP addresses. Akamai's EdgeScape IP database [30] can return the connection speed for a given IP, but will not provide the latency between two given IPs. A more practical solution is Meridian [32] from Cornell University; it can efficiently return the latency between a set of player nodes and a set of server nodes by using routing measurements and gossip protocols. King [35] is another tool that can be used to efficiently estimate latency between two nodes using their immediate downstream DNS servers as proxies.

So the first step in the placement algorithm is to build a cross matrix with the pings between each of the player IPs and the available RM IPs. The algorithm then proceeds as follows:

Let S be the set of RM servers available to host the match, and let m be the number of available servers.

Let P be the set of players participating in this match, and let n be the number of players.

Let $RTT_{s,p}$ be the round-trip-time (ping) from server s to player p, where s belongs to S, and p belongs to P.

Next we disregard all servers that have $RTT_{s,p}$ larger than 180ms, which is the maximum ping that FPS players can reasonably tolerate. This leads to a smaller set S' that only contains servers that satisfy the 180ms constraint.

Now, for each server s in S' we compute a closeness factor C_S representing the average differential ping between the players if server s is picked:

$$C_s = \frac{\sum_{p \in P} \sum_{i \in P} \left| RTT_{s,p} - RTT_{s,i} \right|}{2n}$$

Lastly we search for the smallest C_S among all S' servers and that is our target server to host the match.

The order of complexity of this algorithm is $O(n^2 m)$. Notice that this is not as expensive as it seems since n is usually small on the order of 16 to 64 players max, so this is really a linear algorithm with number of servers.

The placement decision can be cached in case the same set of players play again. The Oracle then proceeds to `scp` the VM image for the needed game server to that location. Once the VM is transferred the Oracle instructs the VMM software to activate that image making the server available to the players.

Note that most VMM software allows for the suspension of VMs in a live state, such that all CPU registers, memory, and IO buffers are dumped to disk, then the machine could be resumed later at the same checkpoint that it was suspended at (this is similar to suspending/hibernating a laptop). Hence, by pre-booting the game server VMs, before suspending and storing them in the Hibernation Nest, when a match is about to start we just need to copy the suspended VM to the RM then resume it fairly quickly and there is no need to wait for a full boot to take place. Once the match is over then the VMs can be suspended back to dormant state, and moved to the Loading Chambers (for maintenance) or Hibernation Nest (for storage), and the RM is now freed for a different game server.

It has to be noted that VMware now offers VMotion [7] technology that can move VMs while maintaining the active connections, this would actually allow us to further optimize the location of the game server while the match is progressing. VMotion can migrate live servers in less than 2 seconds by doing clever memory deltas using bitmaps, unfortunately this 2 second lag will certainly be noticed by the players and might have adverse effects if it happens during a critical moment in the match. Furthermore, this solution requires that the source and target RMs mount the same disk volume from a SAN, and that they have CPUs from the same processor family (e.g. PIII and P4 wont work). Recently published work also show that hot migration is possible based on the Xen VMM [12], in that paper they illustrated mobility for an active first-person-shooter game server (Quake III) which manifested it self as a 60ms temporary jitter for the participating players. However, this solution also requires speedy access to the VM files via an iSCSI gigabit interface. So both VMotion and Xen mobility are not suitable between data centers, but they can certainly be used to improve availability within a data center.

4.2 Backward Compatibility

Most game servers could be ported to this framework with minimal to no code or infrastructure changes; the game administrators would simply need to install the OS and game server software inside of a VM, same way they install it inside a RM today. Once that is done the VM is ready to be instantiated on any RM running the VMM software. The VM preparation and software installation is done in the Loading Chambers, where the RMs main purpose is to host many idle VMs so that administrators can prepare them for operational deployment. The VMs are not exposed to any operational load while waiting in the Loading Chambers other than allowing the administrators to test their configurations.

Note that the VMM software allows for more than one VM to share the same RM, however they are fully isolated and each one can have its own IP address. As far as administrators are concerned when they connect remotely to a given VM in the Loading Chambers, they truly believe it's their own fully assigned isolated real machine. However, if these machines are exposed to heavy load, like decompressing a large tar-ball, then neighboring administrators might sense a sudden slow down and can start to realize that they are sharing the machine with somebody else. It must be noted though that server-class VMM software alleviates this issue by providing a resources quota system that prevents VMs from cannibalizing all of the RM resources (i.e. CPU, Memory, Disk space, IO, Network, etc).

5 EXPERIENCES

It is the goal of this work to show that it is possible to encapsulate legacy game servers via VMMs, to achieve a standardized solution for equi-ping placement without requiring cost prohibitive changes to existing system architectures. We illustrate that this is a practical solution by building out a vMatrix prototype, and porting into it Microsoft's popular Halo PC game server, which is a currently a widely deployed game server.

Our experience confirms that the migration cost is negligible, i.e. no code changes, quick migration, and short learning curve.

5.1 The Experimental Setup

The lab in which we performed the experiments consists of three Pentium III servers at 550MHz, 640MB ram and 9GB hard disks each. The first machine serves as the Production Cluster, the second machine serves as the Loading Chambers, and the third machine serves as the Hibernation Nest and also runs the Oracle software. We used the VMware ESX server, which is a server-class virtual machine monitor. The ESX server consumes about 3.5GB of disk space and 184MB of memory. The CPU overhead is typically less than 5%.

Microsoft's Halo PC is one of the most popular first-person-shooters. We used the Oracle command line interface to create a VM in the Loading Chambers. We then installed on it the software components illustrated in Figure 5. The time it took us to do this is not significantly more than it would have taken to install on a normal real machine. We did not change a single line of source code for the game server (in fact we did not even have access to the source code, just the executables), and it became fully supported within the VMatrix framework as is.

Halo PC Game Server (278MB)
Operating System: Windows XP (1.8GB)
Virtual Machine exposes a PIII-550MHz with 512MB RAM and 5.5GB hard disk.
VMware ESX VMM Server (consumes 184MB RAM, 3.5GB hard disk and 5% CPU)
Real Machine (PIII-550MHz, 640MB RAM, 9GB hard disk)

Figure 5: VM for Microsoft' Halo PC Game Server

Once we configured the VM for Halo PC in the Loading Chambers, we next instructed the Oracle to activate the VM, which caused the VM to be stopped then copied over to the operational cluster then restarted. When a VM is stopped, only two files need to be copied, the first is the configuration file for the VM describing its memory size, Ethernet address, etc, and the second file represents the hard disk of the VM. Once the two files are copied over to a dedicated RM in the operational cluster, it is restarted and becomes ready to accept a live match. Note that the restart operation, though a bit time consuming (as opposed to a suspend/resume operation), has the side benefit of forcing the game server to reregister it self with the gaming directory service (typically GameSpy Arcade Host Directory [31]). This allows the game server name to be mapped correctly to the IP address at the new location, thus providing transparent mobility.

Another trick that we did in this setup was to lie to the underlying VM as to how much physical memory is really present. Even though the real machine only had around 456MB of available free physical memory, we used the VMM virtualization functions to virtualize the remaining 568MB on disk. The result was that the Windows XP VM really thought it had 1024MB of physical memory available. Depending on the active working set while the game server is running, this might make the machine a bit slower, so it's not an optimal situation, but it demonstrates how the services can be moved even between heterogeneous hardware servers.

The resulting VM file size was about 2GB (1.8GB for Windows XP, 10MB for Halo PC Server executables, and 268MB for the environment maps). This translates to 1GB gzip compressed, which takes less then 2 minutes to transfer over 100Mbps Ethernet, or about 4 minutes over a T3 line. The transfer time can be further reduced by using smart differential compression techniques (e.g. chain coding [6]), though this might add some decompression overhead in pulling the VM files back from storage. So the activation time to add servers can be very reasonable and on the order of a few minutes.

The experiment we have described here is limited in that we performed it in a single lab at Stanford, so mostly low LAN pings. We think it's relatively straightforward to generalize the experiment to do cross-Atlantic matches, but we did not have access to an Internet wide VMM cluster.

Note that performance analysis of VMware virtualization overheads is not the goal of these experiments (in fact VMware has strict guidelines against publishing explicit benchmarking metrics); rather it's the illustration of the ease of converting an existing game server into this framework without requiring any coding or architectural changes. However, we attempted to give rough estimates in Figure 5, which illustrate that the VMM CPU performance overhead is usually less then 5%, the memory overhead is about 184MB, and the hard disk overhead is about 3.5GB.

6 Related Work

To our knowledge no body has tackled the problem of optimal server placement for the purpose of reducing ping differential for first-person-shooter game servers. However, there has been many previous research addressing (1) other important aspects of having a distributed large scale multiplayer network of servers, (2) artificially inflating pings to achieve fairness, and (3) modeling game server traffic patterns so that ISPs can properly pre-provision network bandwidth for gaming services. In this section we do a quick overview of these solutions.

Reference [34] is the most related work among the papers we surveyed, it tackles the problem of multi-player server selection (rather than placement). It presents an architecture to pick a game server minimizing the lag differential between a group of players. It assumes a dense pre-deployed population of game servers such that an optimal server can be selected; however, this is only true for the most popular FPS games.

Reference [33] proposes changing the game server to artificially inflate the lag of all players to make them equal. They present a very sound analysis of how the game server can track the error perceived at each player due to stale state, then how to compute the proper delay to hold back the state updates such that all players observe the same error. Though this technique certainly improves fairness, it penalizes the players with good connections (they do propose a budget scheme to try and mitigate this effect).

Reference [10] from IBM TJ Watson Research Center proposes an on-demand service platform for online games, which builds on top of standard grid technologies and protocols. The main issues they tried to address are: reducing latency, improving scalability, and achieving economies of

scale by sharing the platform between multiple game servers. Though they also tackled the problem of first-person-shooters, they did not attempt to directly address the issue of minimizing ping differential between players to achieve fairness. Their solution is not fully backward compatible; it requires game developer awareness of the service platform (i.e. requires code changes), though they tried to minimize that as much as possible.

Another paper from a separate group in IBM TJ Watson and Columbia University [11] proposes a zoom-in-zoom-out algorithm for adaptive server selection for large-scale interactive games. Their focus is to minimize resource utilization while still providing small latency to participants. Their algorithm is primarily targeted for MMORPGs (Massively Multiplayer Online Role Playing Games); which typically have relaxed delay differential requirements as compared to FPS games (can tolerate up to seconds of delay differential versus a max of 180ms for first-person-shooters). While MMORPGs do not exhibit significant cross-coast delay differential unfairness, they could still exhibit cross-continent unfairness, which is why game companies typically localize MMORPGs on a per-continent basis.

There are many solutions [13][14][15][16][17] similar to the two papers listed above, that either focus on the MMORPG problem and/or propose a new platform that requires significant code rewriting for the game servers.

Another branch of interesting research in this space focuses on modeling the traffic generated by game servers [18][19][20] so that ISPs and game server providers can properly provision their networks. Though this type of research helps reduce overall latency, it does not address the unfairness problem due to ping differentials.

It is worth noting that Halo 2 on the Xbox Live [21] platform uses a very interesting method to pick the host for the matches. Instead of having dedicated hosts dispersed on the Internet, the Halo 2 matching servers picks one of the participants to be the hosting server (in addition to being a client). The participant is picked based on historical information that they collect about the throughput and availability of that player when picked as host before. The obvious downside of this solution is that the hosting player is always going to have the smallest ping, which gives them a significant advantage. This approach also has many security issues, as the players attempt to hack their local host code to cheat.

Finally, we refer the reader to the related-work section in our previous paper [9], which contrasts the VMM approach with other solutions like Application Servers, Java servlets, Packaging Tools, OS Virtualization, and Disk Imaging (also know as ghosting).

7 Conclusion

In this paper we presented a novel solution for dynamic first-person-shooter game server placement that reduces overall ping to all participating players to achieve a fair match. The solution is a network of Internet dispersed real machines running virtual machine monitor software, hence allowing game server virtual machines to be moved to an equi-ping real machine. We described our approach in detail and provided a real-life example based on Microsoft's popular Halo PC game. The main advantages of our approach are backward compatibility and the economies of scale that such a virtual machine network provides (since it can work with any game server without any code rewriting).

8 Appendix A

This brief section is provided for the benefit of our readers who are not very familiar with VMM technology. A VMM is a thin layer of software that runs on top of a real machine and exports an abstraction of the real machine [2]. This abstraction is a virtualized (mimicked) view of all hardware in the machine (e.g. CPU, Memory, IO) as shown in Figure 6. VMMs allow multiple guest virtual machines with a full OS and applications to run in separate isolated virtual machine spaces, such that they cannot affect each other. Note that unlike a Java Virtual Machine [4], binary code translation, and machine emulation, the instructions in the VM run natively on the processor of the host RM with almost no change, and hence the performance of code running inside of a VM is almost as fast as the code running directly in a RM.

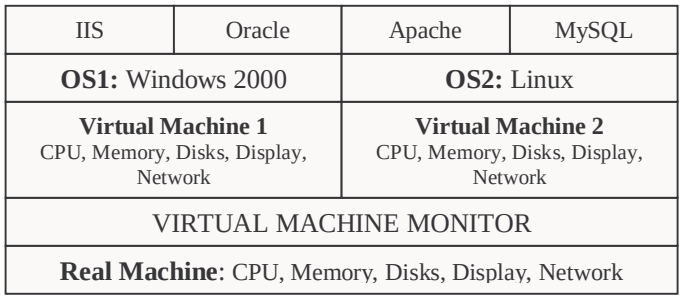


Figure 6: Virtual Machine Monitor

VMMs were introduced in the 1970s by IBM [3] to arbitrate access to hardware of an expensive mainframe machine between a number of client operating systems, and to provide their customers with a forward migration path to newer mainframes. VMMs faded in the 1980s, as the PC became mainstream and computer hardware prices dropped, but were resurrected recently for the x86 architecture by VMware, Inc. [1]. In a well-designed VMM, the code is entirely fooled into believing its mimicked environment such that it cannot detect whether it is running inside a virtual machine or a real machine.

VMware VMM software also provides remote control over the keyboard, monitor, mouse, floppy-drive and CDROM drive of the virtualized machine. This allows owners of the VM to remotely install new software or power cycle the VM without worrying where the machine is physically instantiated, in a sense replacing the popular keyboard/video/mouse (KVM) remote switches (also known as boot boxes).

9 REFERENCES

[1] "VMware Secure Transportable Virtual Machines", VMware Inc. http://www.vmware.com

[2] R. P. Goldberg. "Survey of Virtual Machine Research", IEEE Computer, June 1974.

[3] IBM Virtual Machine 370, 1972. IBM Corporation http://www-1.ibm.com/ibm/history/history/year_1970.html

[4] Tim Lindholm and Frank Yellin. "Java Virtual Machine Specification, 2nd Edition", 1999, Addison-Wesley.

[5] Microsoft Virtual Server. http://www.microsoft.com/windowsserversystem/virtualserver/default.mspx

[6] Constantine P. Sapuntzakis, Ramesh Chandra, Ben Pfaff, Jim Chow, Monica S. Lam, and Mendel Rosenblum. "Optimizing the Migration of Virtual Computers", USENIX OSDI 2002.

[7] "Building Virtual Infrastructure with VMware VirtualCenter", white paper, VMware Inc. http://www.vmware.com/pdf/vc_wp.pdf

[8] Amr Awadallah and Mendel Rosenblum. "The vMatrix: A network of virtual machine monitors for dynamic content distribution", Seventh International Workshop on Web Content Caching and Distribution, August 2002.

[9] Amr Awadallah and Mendel Rosenblum. "The vMatrix: Server Switching". IEEE 10th International Workshop on Future Trends in Distributed Computing Systems, Suzhou, China, May 2004

[10] Debanjan Saha, Sambit Sahu, and Anees Shaikh. "A Service Platform for On-Line Games". Proceedings of the 2nd workshop on Network and system support for games, May 2003.

[11] KangWon Lee, BongJun Ko, and Seraphin Calo. "Adaptive Server Selection for Large Scale Interactive Online Games". ACM NOSSDAV'04, Cork, Ireland June, 2004.

[12] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. "Live Migration of Virtual Machines". USENIX NSDI 2005.

[13] D. Bauer, S. Rooney, and P. Scotton. "Network infrastructure for massively distributed games". Workshop on Network and System Support for Games (NETGAMES), Germany, April 2002.

[14] A. R. Bharambe, S. Rao, and S. Seshan. "Mercury: A scalable publish-subscribe system for Internet games". Workshop on Network and System Support for Games (NETGAMES), Germany, April 2002.

[15] S. Fiedler, M. Wallner, and M. Weber. "A communication architecture for massive multiplayer games". Workshop on Network and System Support for Games (NETGAMES), Germany, April 2002.

[16] B. Knutsson, H. Lu, W. Xu, B. Hopkins. "Peer-to-Peer Support for Massively Multiplayer Games". INFOCOM, Mar. 2004.

[17] Butterfly.net, Inc. The Butterfly Grid, 2003. http://www.butterfly.net/platform

[18] Wu-chang Feng, Francis Chang, Wu-chi Feng, and Jonathan Walpole. "Provisioning On-line Games: A Traffic Analysis of a Busy Counter-Strike Server," SIGCOMM Internet Measurement Workshop, November 2002.

[19] Tristan Henderson, and Saleem Bhatti. "Modelling user behavior in networked games". ACM Multimedia 2001, June 2001.

[20] Johanness Faerber. "Traffic Modelling for Fast Action Network Games". ACM Multimedia Tools and Applications, May 2004.

[21] Microsoft Corporation, "Xbox Live". http://www.xbox.com/live

[22] Wu-chang Feng, Wu-chi Feng. "On the geographic distribution of on-line game servers and players". Workshop on Network and System Support for Games (NETGAMES), 2003.

[23] Grenville Armitage. "An experimental estimation of latency sensitivity in multiplayer Quake 3". 11th IEEE International Conference on Networks (ICON 2003), Australia, September 2003.

[24] Grenville Armitage, Lawrence Stewart. "Limitations of using Real-World, Public Servers to Estimate Jitter Tolerance Of First Person Shooter Games", ACM SIGCHI ACE2004 conference, Singapore, June 2004

[25] Tom Beigbeder, Rory Couglan, Corey Lusher, John Plunkett, Emmanuel Agu, and Mark Claypool. "The Effects of Loss and Latency on User Performance in Unreal Tournament 2003". ACM SIGCOMM Workshop Network and System Support for Games (NETGAMES), 2004.

[26] Peter Quax, Patrick Monsieurs, Wim Lamotte, Danny De Vleeschauwer, and Natalie Degrande. "Objective and Subjective Evaluation of the Influence of Small Amounts of Delay and Jitter on a Recent First Person Shooter Game". ACM SIGCOMM Workshop Network and System Support for Games (NETGAMES), 2004.

[27] Tristan Henderson. "Latency and User Behaviour on a Multiplayer Game Server". Third International Workshop on Networked Group Communication (NGC2001), November 2001, London, UK.

[28] Xen: An open source Virtual Machine Monitor for x86. http://www.cl.cam.ac.uk/Research/SRG/netos/xen/index.html

[29] The Cyber Athlete Professional League. http://www.thecpl.com/league/

[30] Akamai EdgeScape: Internet IP knowledge base. http://www.akamai.com/en/html/services/edge_how_it_works.html

[31] Game Spy Arcade Game Server Directory. http://www.gamespyarcade.com/

[32] Bernard Wong, Aleksandrs Slivkins, and Emin G¨un Sirer. "Meridian: A Lightweight Network Location Service without Virtual Coordinates". ACM SIGCOMM 2005, Pennsylvania, USA, August 2005.

[33] Sudhir Aggarwal, Hemant Banavar, Sarit Mukherjee, and Sampath Rangarajan. "Fairness in Dead Reckoning based Distributed MultiPlayer Games". ACM NETGAMES 2005, New York, USA, October 2005.

[34] Steven Gargolinski, Christopher St. Pierre, and Mark Claypool. "Game Server Selection for Multiple Players". ACM NETGAMES 2005, New York, USA, October 2005.

[35] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. "King: Estimating Latency between Arbitrary Internet End Hosts". SIGCOMM Internet Measurement Workshop 2002, France, November 2002.