The vMatrix: A Network of Virtual Machine Monitors for Dynamic Content Distribution

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

Abstract

Today there are many solutions for the caching and distribution of static content (e.g. images, html pages, video files). However, delivering dynamic content and interactive services (e.g. CGI services) remains a challenge due to the many dependencies that such services have on custom libraries, third party modules, operating systems, and server hardware. In this paper we propose a novel solution, which builds on top of the classic operating system concept of a virtual machine monitor (VMM). A VMM allows us to encapsulate the state of the machine in a virtual machine file, which could then be instantiated on any real machine running the VMM software. This eliminates the dependencies problem by allowing us to move the whole machine around, thus reducing the problem of dynamic content distribution to delivering large virtual machine files among real machines running the VMM software. The main challenge is mimicking the original environment surrounding the virtual machine such that it can be transparently transported between real machines without needing to make any software changes. We propose a practical architecture and analyze the performance tradeoffs for two-tier architectures that require connectivity from the mobile front-ends to back-end databases.

Keywords: dynamic content distribution, web caching, virtual machine monitor, mobile interactive services.

1 Introduction

In this paper we propose and analyze a practical solution for the distribution and replication of dynamic content. In a nutshell, dynamic content is the result of a program executed on the web server (e.g. a CGI script), as contrasted with static content, which is simply a file retrieved as is from the hard disk of the web server (e.g. an image).

Today there are many solutions for distributing and replicating static content, ranging from classic web proxy caches [5][8][10][11] to the more recent content distribution networks (CDNs) [6][7][9]. The main benefits of distribution and replication are: improving end user response time, higher availability, absorbing flash crowds, and network bandwidth savings.

Recent Internet measurements indicate that non-cached dynamic content constitutes about 40% of web requests [27][38]. Dynamic content is indeed becoming more prevalent on the web as more applications are being ported from the desktop to become web services, which allows quicker bug fixing, upgrading, and collaboration, among other benefits.

However, the problem of distribution and replication of dynamic content continues to elude us due to the many dependencies that such services have on custom libraries, third party modules, operating systems, and server hardware. Simply copying the code of the dynamic service is not possible since the target machines need to have exactly the same environment for the code to run unchanged, which is not very practical. Previous solutions in the area of dynamic content distribution are not backward compatible and require changes to the existing applications.

We propose a backward compatible solution that builds on top of the classic operating system concept of a *Virtual Machine Monitor* (VMM) [35]. 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 mimics exactly what a real machine would look like. This allows us to encapsulate the state of the entire machine in a VM file, which could then be instantiated on any RM running the VMM software. This solves the dependencies problem since the whole machine is transferred with the code, modules, libraries, and operating system that the dynamic service depends on. Hence the problem of dynamic content distribution is reduced to delivering large VM files within a network of RMs running the VMM software, we call this network *the vMatrix*[*]

[*] The name "*The vMatrix*" comes from the analogy to the popular sci-fi movie "*The Matrix*". In the movie, machines controlled humans by virtualizing all their external senses; we

We do not attempt to build a VMM, but rather we reference existing software for the x86 architecture from VMware, Inc. [18]. The main challenge that we focus on is how to *mimic* the environment surrounding the virtual machine such that it can be moved transparently between real machines without making significant architecture or software changes. In this paper we demonstrate that through the use of off-the-shelf technologies it is indeed possible to mimic the surrounding environment. Through network address translation (NAT) it is possible to provide transparent mobility of VMs between RMs, such that the transported machine can resume normal operation once it is re-instantiated at a new location without needing to reconfigure its software (e.g. networking stack). For multi-tier architectures, virtual private networks (VPNs) allow us to provide secure connectivity between the transported front-end VMs and any other back-end machines they depend on.

We claim that the distinguishing advantages of this approach are: very low switching cost of converting an existing web service to run within such a framework, ability to move front-end servers to the edge, and on-demand replication of servers to absorb sudden surges of requests. The main disadvantage is that the mobile VM files can be pretty large, on the order of gigabytes, however, it is similar to large multimedia video files for which many solutions exist today to deliver efficiently on the internet [28].

In section 2 we present example scenarios to motivate the solution that we are proposing. In section 3 we enumerate the main functional and performance challenges. In section 4 we focus on how to mimic the environment surrounding a VM to allow transparent mobility and two-tier secure connectivity. In section 5 we address the performance implications of porting two-tier architectures into our proposed framework. Finally in sections 6, 7, and 8, we cover future work and related work then conclude.

2 Motivation

Consider a map service where the users fill a web form with an address, and in return they receive a street map. Notice that caching the returned map image is useless since while many users could be using this service at the same time, they are most likely interested in different destination addresses. However, the underlying code that renders this page and the geographical database it depends on are very static in nature. Typically this web service is built out of a single-tier architecture where the necessary code and geographic data are self-contained within a single server. A load-balancer is used in front of a farm of identical map servers to scale to a large number of users. Also the server cluster is replicated in the east and west coast primarily for reliability reasons. A typical system is illustrated in Figure 1.

It is cost prohibitive for the service providers to operate server clusters in more than a couple of data centers; indeed, this is one of the main value-adds of CDNs for static content distribution. Static mirroring does not provide real-time distribution and replication. However, if we install the servers inside transportable virtual machines (which is no different than installing them inside real machines) and allow them to be moved in real time to demand hot spots, then we can achieve all benefits of content distribution as shown in Figure 2.

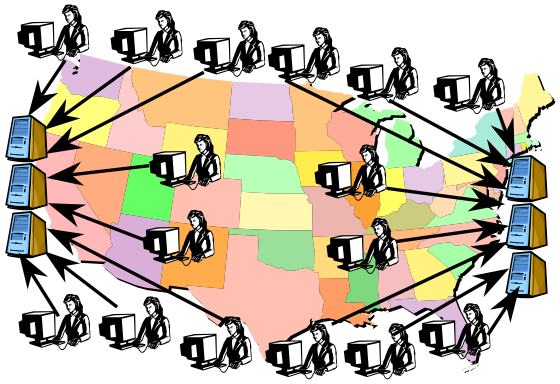

Figure 1: Single-tier architecture with static mirroring.

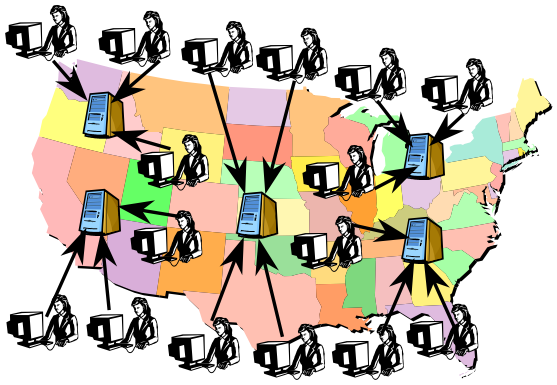

Figure 2: Single-tier architecture with mobile virtual machine servers to enable dynamic content distribution and replication.

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 benefits of distribution and replication are as follows:

1. *Improving the response time perceived by end users*: By moving the content closer to the network edge where the users are directly connected to their Internet service providers (ISPs), it is possible to significantly reduce the overall response time for retrieving content since packets travel over a smaller number of hops. Research measuring the benefits of CDNs shows that they in fact provide a valuable service by avoiding the slowest servers [1].

2. *Improving overall availability and uptime*: Distributing the content over a number of geographically disbursed locations provides better overall availability since if one of the servers is down, or if the network connectivity to it is lost; then another close by server could always be found. It also offers stronger protection again DoS (Denial of Service) attacks, because it is harder to attack all of the geographically dispersed servers. Recent simulation results [49], based on actual connectivity traces, show that distributing code inside the network can result in an order of magnitude improvement of availability.

3. *On-Demand replication to absorb flash crowd requests:* By allowing servers to be cloned (by making a copy of the VM file) and replicated on demand in different locations, flash crowds can be dynamically absorbed. A flash crowd is unpredicted increase of web requests, such as an unforeseen surge of stock market activity.

4. *Reducing the overall bandwidth consumed in the network:* By placing the content closer to the end users, the content packets cross a smaller number of links and hence consume less aggregate bandwidth. This is validated by CDN hit ratio analytic modeling [2] and by proxy cache measured results [3][4].

However, the scenario is slightly more complicated if the front-end web servers need to query a back-end advertising database to display ads on the results web page. In this case connectivity is needed between the transported front-end machines and the back-end ad databases as illustrated in Figure 3. In section 5 we discuss in detail the trade-offs of porting a two-tier architecture to this framework.

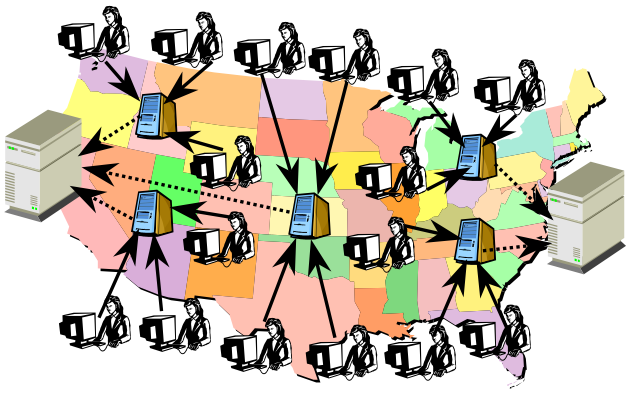

Figure 3: Two-Tier architecture with virtual private networks from the front-end servers to back-end databases.

There is also a whole new class of interactive dynamic services that require optimal placement within the network. For example, a multi-player online gaming server works best if it can be positioned at an equal latency point from all players currently connected to it.

In conclusion, by operating a network of RMs running the VMM software, if there is a sudden surge in demand at any place in the world then the web server hosting the service, or even the entire cluster, can be replicated and transported in real-time (order of minutes) to be closer to the demand.

3 Proposed Solution

Our proposed solution for dynamic content distribution is a network for delivering VM servers between RM hosts running the VMM software. As contrasted to previous work (which we cover in section 7), we claim that this solution presents the smallest switching cost for porting an existing web service into such a dynamic content distribution network. In fact, single tier architectures could be ported to this network with no code changes; the developer would simply need to install the service inside of a VM, same way he installs it inside a RM today. Once that is done the VM is ready to be instantiated anywhere in the world. Similarly two-tier architectures could be ported with no code changes, but might require some code changes for performance reasons as discussed in section 5.

In this paper we do not tackle the problem of *where* to send dynamic content, but rather we tackle the problem of *how* to distribute and replicate dynamic content with minimal code changes. The problem of where to send dynamic content is very similar to the caching and distribution of static content, and the basic premise is to detect Internet demand hot spots and cache content in close proximity to these hot spots. There are a number of solutions and algorithms for determining the best places

to replicate content and mirror servers [6][21][22][23][31].

Achieving backward compatibility and fluid distribution does not come without challenges. The main challenges are twofold:

3.1 Functional Challenges

The main functional challenge is to *mimic* everything around the operating system and web service application such that it runs just as it would run in its native environment. There are two main components:

3.1.1 Machine Mimicking

For the purpose of machine mimicking we reference virtual machine monitor software from VMware, Inc. A VMM is a thin layer of software that runs on top of a real machine and exports an abstraction of the real machine [35]. This abstraction is a virtualized (mimicked) view of all hardware in the machine (e.g. CPU, Memory, IO) as shown in Figure 4. VMMs allow multiple guest virtual machines with a full operating system and applications to run in separate isolated virtual machine spaces, such that they cannot affect each other. Note that unlike a Java Virtual Machine [37], 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.

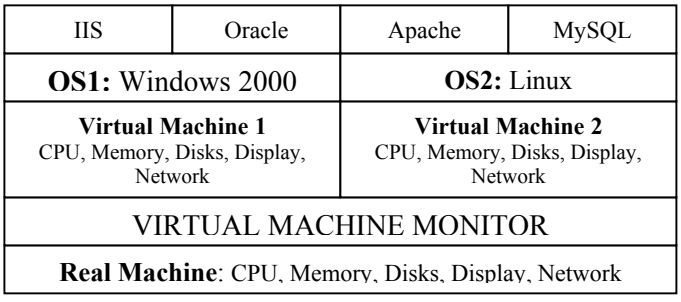

Figure 4: Virtual Machine Monitor

VMMs were introduced in the 1970s by IBM [36] to arbitrate access to hardware of an expensive mainframe machine between a number of client operating systems. VMMs died in the 1980s, as the PC became mainstream and computer hardware prices dropped, but were resurrected recently for the x86 architecture by VMware, Inc. [18]. VMMs have many newly discovered applications, including, but not limited to: running Microsoft Windows and Linux at the same time, isolation of applications, training inside undoable virtual machines, OS kernel development, and web hosting for multiple clients in isolated virtual machines.

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.

The VMware VMM software also enables the suspension of VMs in a live state, such that all CPU registers, IO buffers and memory are dumped to disk, then the machine could be resumed later at the same point it was suspended at (this is similar to suspending a laptop). This feature has the added advantage that the suspended VMs can be activated in a very short time, typically around 10 seconds, instead of booting up the machine from idle state which tends to take more time and consume more resources.

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.

3.1.2 Network Environment Mimicking

Once the machine is mimicked, we then need to mimic its interactions with the outside world. We can always view any web server as a black box with the main connectivity to the outside world achieved through its networking card. The trick then is to maintain the outside view from the machine regardless of where it is located. In section 4 we cover in detail our approach for network environment mimicking, in summary it builds on off the shelf technologies like network address translation (NAT) and configurable virtual private networks (VPNs).

3.2 Performance Challenges

One of our goals is to allow services to be ported into this framework with minimal application changes. However, certain changes might be required to improve performance of the transported service. For example, if there is a lot of communication between the front-end mobile VM and a back-end small database, then it might be best to move the back-end database with the front-end server which could lead to significant code changes. Performance challenges are discussed in detail in section 5.

4 Network Mimicking

Recall that the vMatrix is an interconnected mesh of RMs ready to host VMs. The RMs exist within entities that we refer to as a *hosting network (HN)*. HNs are Internet service providers that operate the RMs and make them available for hosting VMs. We propose a two-level global-local architecture. Global assignment decides to which HN a given VM should be sent, and then local assignment is responsible for picking a suitable RM within a given HN. Figure 5 illustrates the main components, which are:

1. *CPA:* The CPA is the content placement agent that decides to which HN should the content VM server be sent. In its most basic form its functionality is very similar to that of the distribution and replication of static content. We do not focus on the details of the CPA in this paper, indeed, the CPA can be a human operator observing a world map were he/she can see all locations with available real servers then adjust a knob to increase the number of instances of any internet service at any location with a demand hot spot.

2. *VMA:* The VMA is the Virtual Machine Agent and it runs locally in the HN. As new RMs are added to the HN and loaded with the VMM software, they are subscribed with the local VMA. Hence, the VMA is aware of all available RMs within its network. The VMA is responsible for the local matching of VMs to RMs. The VMA is also responsible for configuring all the networking components (DNS, VPN, NAT/Load Balancer, Firewall) to maintain the mimicked network environment around the VM. Below we describe the functions of the VMA at a very high level. Similarly, the functions of the VMA could be either automated or performed by a human.

At a high level, once the CPA decides that a new server VM needs to be instantiated within a given HN, it contacts the VMA within that network. The VMA then replies back whether there are available RMs to host the new VM. If so then the CPA sends the VM file to the VMA, and consequently the VMA transfers the VM file to a suitable RM, configures all the network components, and starts the VM server.

4.1 Transparent Mobility

By transparent mobility we mean the ability to move the VM server and have it resume normal operation at a new location without the need to reconfigure its software (e.g. networking stack). We propose a solution based on network address translation techniques (NAT)[43]. The IP address for the mobile VM is pre-picked from a pool of private intranet IP addresses [42], e.g. the 192.168.x.x range. Machines belonging to the same domain (e.g. yahoo.com) should not use the same IP address more than once, to avoid conflicts between VMs instantiated in the same network. However, VMs belonging to different domains (e.g. yahoo.com and msn.com) can share the same private IP address since they will be mapped to different VLANs as described in the next section.

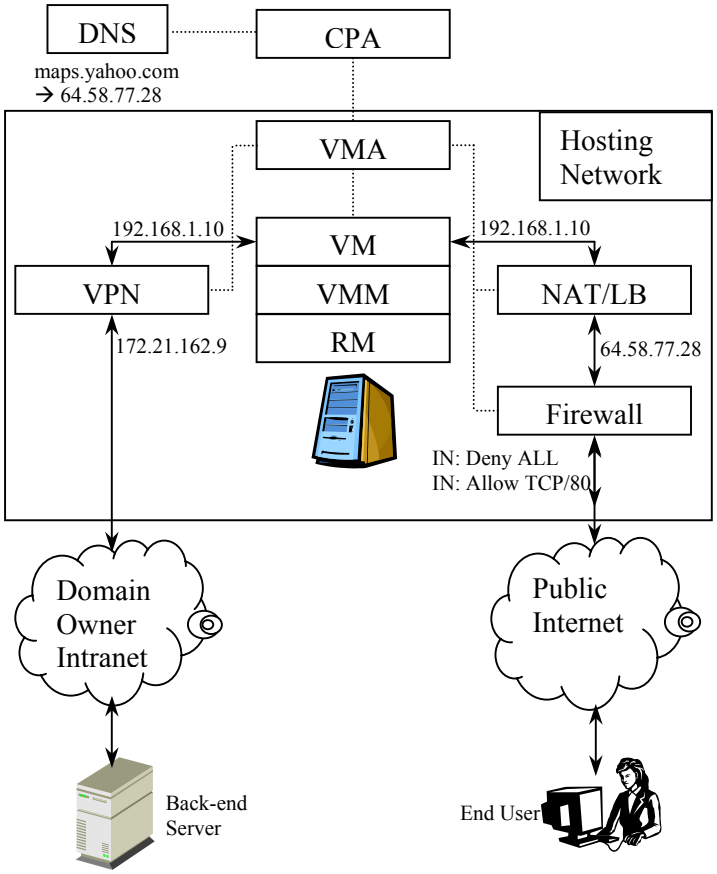

Figure 5: The Content Placement Agent (CPA) instructs the VM Agent (VMA) to instantiate a VM on a RM. The VMA then configures the necessary networking components (NAT/Load Balancer, DNS, Firewall, VPN)

Referring to Figure 5, the IP address picked for the VM is 192.168.1.10. Once the VMA instantiates the VM on the RM, it configures the NAT to map the internal private address 192.168.1.10 to a public IP address of 64.58.77.28. The VMA then contacts the CPA and instructs it to configure the primary DNS (Domain Name Server) to include the new IP address of 65.58.77.28 as a possible target for maps.yahoo.com. When a given user browser queries the network for maps.yahoo.com, the request is relayed to the primary DNS server for this domain, which uses the source IP address of the user's

machine to translate maps.yahoo.com into the server IP which is closest to the user. Such a DNS response usually has a very short time-to-live attached to it, so that intermediate DNS servers relay the requests back to the primary DNS server. One known short-come of this approach is that some times intermediate DNS servers are configured to enforce a minimum time-to-live, over-riding the time-to-live specified by the primary DNS server. Similarly some web browsers on the user side might cache the DNS response rather than resubmit the DNS query.

The main benefit of this approach for transparent mobility is that the VMs can be moved between networks without needing to reconfigure them, but rather the VMA configures the networking elements surrounding them, the VMA need not know any of the details of what is running inside the VM. This approach also allows cloning and replicating VM servers in different networks, since local private IP addresses are re-usable in distinct networks. So if there is a full cluster in the east coast, and a spike of demand is detected in the mid-west, then the whole cluster could be cloned (essentially copies of the VM files) and replicated as is in the mid-west.

Note that the NAT box might also be a load balancer (LB). In this case multiple VMs with private IPs sitting behind the LB can map to the same public IP address. The VMA simply needs to add the private internal IP address for the new VM server to the rotation of private IPs serving the public IP address of 64.58.77.28. The load balancer then spreads incoming requests across all private IP addresses currently in rotation including the new VM server that was just instantiated.

Another value add is absorbing flash-crowds. In that case a variable number of suspended VMs could be resumed on demand, to absorb this short demand period. Once the flood is over then the VMs can be suspended back to dormant state, and the host RMs can be used for some other service. If a VM file is already present in the network with the surge of demand, then starting a new VM from a suspended state takes on the order of 10 seconds, as compared to booting a server from scratch which can take a few minutes.

Another issue that needs to be considered is moving or suspending servers with active web connections. The graceful solution is to take such servers out of load-balancer rotation, or remove their DNS and NAT mappings, then after detecting that all existing connections are closed, it can be safely suspended, moved to a new RM, and then resumed.

Even though DHCP [47] is a possible alternative to NAT, it is not transparent to the software inside the VM and requires that the IP lease be renewed once the machine is moved to a new network. This dictates that the VMA needs to be aware of which operating system is installed inside the VM and issue the right command sequence to invoke a renewal of the IP lease, versus the NAT approach which is completely external to the VM. Another complication is that some server applications (e.g. ftp) within the VM server might cache the IP address and changing the IP address using DHCP will require restarting all such applications to get the new IP address.

Finally, for security reasons, the VMA also configures a firewall to secure the public address. So as shown in Figure 5, only port 80 will be allowed incoming access to the 64.58.77.28 public IP address.

4.2 Secure Connectivity

Two-tier architectures require that the mobile front-end VMs maintain connectivity to back-end servers that they rely on. We propose a solution based on configurable virtual private networks (VPNs) [44]. In detail, once the VMA instantiates the VM on the RM, a VPN is configured providing a secure IP tunnel (IPSec) [45] from the front-end VMs in the HN to the back-end servers in the Intranet of the owner domain. Note that the VPN setup is done transparently below the VM, thus it does not require any software configuration inside the VM. This is illustrated in Figure 5, where the 192.168.1.10 private address on one end of the secure tunnel is mapped to the 172.21.162.9 private intranet address (in the intranet of the domain owner) on the other end of the secure tunnel. The meta information attached with each VM file is the server local IP address and an authentication certificate for establishing the VPN tunnel.

Another security issue that needs to be considered is VM servers belonging to different domains but connected to the same network switch (e.g. yahoo.com and msn.com). In this case it is possible for the VMs to snoop on each other's packets or even attack each other. This problem could be solved by allowing the VMA to configure VLANs (virtual LANs) on the network switch that the VMs are connected to, such that different domains belong to different VLANs, this provides perfect network isolation between domains, while also providing secure connectivity between VMs belonging to the same domain. Even in the case where two different domain VMs end up in the same RM, the VMware VMM can provide separate VLANs for the

hosted VMs preventing them from snooping or attacking each other.

5 Two-Tier Architectures

As discussed earlier in section 2, single tier architectures fit very nicely within the vMatrix, however two-tier (and consequently n-tier) architectures have the added complexity of requiring communication from the front-end mobile VM servers to back-end servers.

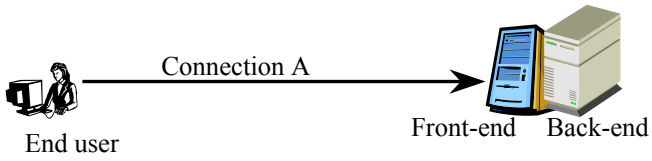

Figure 6: Two-tier architecture with front-end and back-end in same local area network.

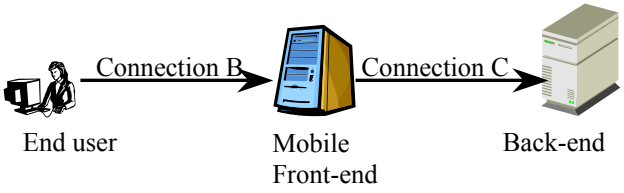

Figure 7: Two-tier architecture with front-end mobile to any place on the Internet.

Figure 6 illustrates a standard two-tier architecture with both the front-end and back-end servers located in the same network, while Figure 7 illustrates a two-tier architecture with mobile front-ends. The request from the end user is sent over connection B (which can be a number of Internet hops) to the mobile front-end machine, which then queries some information from the back-end database over connection C (again possibly a number of hops away), then uses the returned results to generate a web page for the end user.

Referring to section 2, the main motivating goals are faster response time, improved availability, on-demand replication and saving bandwidth. The first three goals are clearly achieved in the single tier case, and the fourth goal is achieved if the total number of bytes served by the VM server is bigger than its size. However, in the two-tier case the benefits may or may not be achieved depending on the traffic patterns from the end user to the front-ends and from the front-ends to the back-ends. Lets consider the impact on the four goals.

5.1 Response Time

Here we present a simplistic back of the envelope analysis to demonstrate that response time benefits can be achieved for the two-tier case under certain practical conditions.

In Figure 6 the response time perceived by the end user is primarily dominated by connection A. The communication overhead between the front-end and the back-end is relatively negligible since it usually occurs over a well-provisioned 100Mbps switch or even a 1Gbps switch. However, in Figure 7 the response time is a function of both connection B and connection C. Lets assume that TCP is used for communication over connections A, B, and C. According to [46], a simplified formula to derive the response time (R) to serve a request is:

$$R \geq k \cdot N \cdot RTT \cdot \sqrt{p}$$

Where k is a constant, N is the number of packets (derived as the request size divided by the maximum segment size allowed over the connection), RTT is the round trip time, and p is the probability of packet loss.

For distribution to be worthwhile we require that the total response time in the distributed case be less than the response time in the non-distributed case, thus,

$$N_B \cdot RTT_B \cdot \sqrt{p_B} + N_C \cdot RTT_C \cdot \sqrt{p_C} < N_A \cdot RTT_A \cdot \sqrt{p_A}$$

But N_B is equal to N_A, so the generalized requirement is:

$$N_C / N_B < (RTT_A \cdot \sqrt{p_A} - RTT_B \cdot \sqrt{p_B}) / (RTT_C \cdot \sqrt{p_C})$$

To further simplify this formula, assume that the front-end is moved as close to the end user as possible such that (RTT_B, $\sqrt{p_B}$) will be relatively negligible, and (RTT_C, $\sqrt{p_C}$) will be almost equal to (RTT_A, $\sqrt{p_A}$), hence the requirement reduces to:

$$N_C < N_B$$

Even though this analysis is based on a TCP connectivity assumption, the final result is rather intuitive. For there to be benefit in moving the front-ends as close as possible to the end users, the number of packets exchanged between the front-ends and the back-ends should be a smaller than the number of packets exchanged between the end user and the front-ends. However, some proprietary back-end protocols might make the assumption of communication over a highly responsive low-loss LAN, and thus might by very

sensitive to loss. When such protocols are used over a WAN they might stall in the event of repetitive packet loss or long round trip times. Such protocols are not compatible with our proposed framework and will need to be rewritten under the assumption of WAN communication, i.e. they need to be resilient in case of frequent packet loss and large delays. It should be noted though that proprietary protocols are used in a very small number of websites, as compared with TCP.

We also note that even in the case where $N_C > N_B$, the end user might *perceive* better response time by recoding the front-ends such that the web page is partially rendered and a progress meter is displayed while the request between the front-end and back-end is fulfilled. The user perceives such an experience as more responsive than a very long delay before any parts of the web page are rendered due to a far away front-end.

Another solution to address cases where $N_C > N_B$ is to move the back-end database with the front-end machines. For example, if the back-end database is a relatively small read-only static geographical maps database, then we can package it in a VM and ship it in conjunction with the front-ends. This dictates the expanded notion of moving a cluster of VMs together, rather than individually. However, this solution is not practical for huge back-end databases that are hosted in a RAID system (on the order of 100GB), and it is also not practical when the end user is submitting information that needs to be written to the back-end servers, e.g. an auctions listing.

5.2 Availability

For the single-tier case, availability is certainly improved since the servers are geographically dispersed in more than one location. Hence, if connectivity is lost to one of the servers, or if that specific server is down, then it is always possible to find an alternative server. Simulation results by [49], confirm that distributing code inside the network results in an order of magnitude improvement of availability.

However, in the two-tier case, while availability between the end users and the front-ends is improved, availability between the front-ends and the back-ends is compromised. In Figure 6 the front-ends and back-ends are collocated on the same network switch, which is usually a reliable system constituting of a primary switch with a hot standby to take over in case of failure. However, in Figure 7, the front-ends are connected to the back-ends over an unreliable WAN network and connectivity could be lost at any time. Most web services will display an error message to the end user when connectivity is lost to the back-end. So at least in the distributed case, the end user will get a message informing them of the service interruption, versus if connectivity is lost to the front-ends then the user gets no indication of status, which is a very bad user experience.

To further improve service availability in the distributed case, the front-ends can be re-designed to cache recent requests to the back-ends and use such cached results when connectivity is lost. Indeed most web services are designed with this in mind since it provides a graceful degradation path. This solution only works for services that are read-only, or write-once read-many. For example, if the front-ends simply need to connect to a back-end ad server, then by caching these ads on the front-ends, the ad can be re-used and the user will not feel the disconnection that took place. Similarly, user profiles for personalized web pages are typically write-once read-many, and hence if the front-ends cache the user profiles then loss of connectivity will not affect the experience of the end user. The only downside of that is that the user might get a slightly stale profile, versus not getting any service if connectivity is lost to the front-ends.

As discussed above, in some scenarios it might be reasonable to package the back-end database in a VM and ship it in conjunction with the front-ends. Hence eliminating the availability problem between the front-ends and back-ends.

5.3 On-Demand Replication

For the single-tier case, multiple front-ends can be *cloned* (i.e. same VM file copied more than once) and replicated on-demand in different networks to absorb flash crowd requests. For the two-tier case, if the back-end databases allow any front-end to connect to them through some know port (e.g. Oracle's TNS listener) then we can certainly instantiate more front-ends as demand dictates. However, if the connectivity protocol between the front-ends and back-ends is a proprietary protocol that makes assumptions about the IP addresses or some logical name of the front-ends then architecture changes will be required.

5.4 Bandwidth Savings

It is worth noting that the size of VM files will typically be on the order of gigabytes, even in compressed form. A 4GB file takes around 15 minutes to be transmitted over a typical T3 link, or 1 minute over an OC12 link, which is reasonable to achieve real-time distribution of

dynamic content. This problem is very similar to the problem of transferring large multimedia video files for which many solutions exist today providing efficient and reliable delivery [28].

However, we always need to weigh the 4GB that will be consumed while moving the server, against the bandwidth savings that will be achieved by moving the server closer to end users. If the total number of request bytes served by the web server, during its lifetime at the new location, is larger than the size of the VM file then obviously bandwidth savings are achieved. It is also possible to pre-push the VM file of the server to all major networks smoothly over night when bandwidth is available rather than abruptly during daytime. Also if the server is being replicated in a number of networks then proper reliable multicast techniques [28][29] can be used to further reduce the consumed bandwidth.

6 Future Work

It is the goal of this work to show that it is possible to encapsulate legacy Internet services via VMMs, to achieve a standardized solution for improving the scalability, interactivity, availability, and efficiency of web services without requiring cost prohibitive changes to existing system architectures. Our approach to illustrate that this is a practical solution is to build out a *vMatrix* prototype, and port into it a number of existing Internet services in collaboration with Yahoo, Inc. These services represent the spectrum of practical web service architectures (e.g. single tier, two-tier, write-once/read-many, write-many/read-many, single-user, one-user to many-users). We seek to illustrate that the migration cost is minimal for both the developers of the service and the system administrators, i.e. quick migration, short learning curve, and support for traditional system administration tasks like troubleshooting, rebooting, monitoring, code updates, and log collection.

Other interesting problems and applications that we might tackle are:

6.1.1 Server Multiplexing

Internet portals run a large number of services, on the order of 100s, which use thousands of servers. Typically a number of servers are fully dedicated to a given service and are provisioned at the peak load that this service is expected to need. For example, a stock quote service might need 100 dedicated servers during weekdays when there is high-demand, but on weekends it could only use around 5 servers while the remaining 95 servers stay idle. Alternatively, a sports service uses only 10 servers during weekdays and uses 100 servers on the weekend. Using the *vMatrix* we can multiplex both services on the same 100 servers instead of requiring 100 fully dedicated servers for each. This is very similar to circuit switching versus packet switching, instead of dedicated peak provisioned servers (circuit) for a given service; we multiplex the VMs (packets) across a shared pool of servers. The key is how to provision the total number of servers to minimize the probability of server loss (i.e. a requested VM with no RM to be instantiated on). We conjecture that this approach could lead to significant reduction in the number of required servers without imposing high switching costs or administration overheads.

6.1.2 VM Scheduling

Due to the distribution of the VM front-end servers over a potentially large number of geographically dispersed RMs, it is very possible that the number of user requests to any given VM is at such a low rate that it only uses a fraction of the resources (CPU, memory, hard-disk, network bandwidth) of the hosting RM. In this case it is desirable to instantiate more than one VM per RM to maximize utilization of the RM resources. The key for achieving this is to balance maximizing the utilization of the host RM, without compromising the performance of the guest VMs.

6.1.3 The CPA optimization problem

In section 4 we assumed the presence of a content placement agent (CPA) that decides where dynamic content should be delivered. There are a number of solutions and algorithms for determining the best places to replicate content and mirror servers. However, due to the fluidity and small time scale that our solution introduces, it is possible to move servers around much quicker, and hence static mirror placement algorithms might not recommend the optimal placement [21][22][23]. Some previous solutions tried to address the problem of dynamic server mobility [31]. We propose a grander optimization problem which takes into account all of the following components:

1. Location of RMs where VMs can be instantiated.
2. Location of users sending requests to the servers
3. Location of back-end servers relative to RMs
4. Size of VM files relative to size of server requests
5. Cost of bandwidth during different times of day

Then the output of such an algorithm is the optimal time and location to instantiate a new server. Opus [53] presents one way to address this optimization problem.

6.1.4 Compute Utility

A special case of the framework we presented is to provide a compute utility service where the location of the VM servers is not important, rather the VMs simply need to consume CPU cycles any place in the world to perform simulations or other CPU intensive tasks.

7 Related Work

The problem of *static* content distribution and replication is an inherently easy one since the fundamental object being cached is a nicely encapsulated file that does not have many dependencies. Static content changes at most once every couple of days (e.g. images, fixed HTML pages, download files, video files). The common distribution and replication solutions are standard proxy cache hierarchies with expiration times associated with the cached objects [5][8][10][11], or CDNs that push such objects to the edge of the network [6][7][9].

It is important to distinguish the difference between *frequently changing* content as opposed to dynamic content. Frequently-changing content gets stale very quickly as time passes by and needs to be delivered to all edge nodes in a real time fashion (e.g. news, stock quotes, sports scores). It is sometimes also referred to as dynamic content. The common solutions for frequently-changing content distribution are partial/delta web page composition at edge nodes by pulling content pieces from back-end servers [17][25][32], origin servers that track content dependencies and proactively push content to edge nodes [12][13][14][15][16][29][34], or a combination push-pull approach [33].

The main focus of this paper is the distribution and replication of *dynamic* content. We define dynamic content as the result of a program executing on the server at the time of the request. Unlike dynamic content caching research that focuses primarily on saving server CPU cycles [14][26][27], our focus is to achieve the benefits covered in section 2.

Previous work in the area of dynamic content caching suffers from a common disadvantage, which is requiring the web service developers to recode their applications within a new framework or adhere to a set of strict guidelines. In other words, they are not backward compatible. This represents a huge impediment for developers, since it requires them not only to learn how to use a new framework, but also to port all their existing code to this new framework. This is not cost effective since the salary of system programmers, is typically much higher than any of the network or server costs.

The Internet Content Adaptation Protocol (ICAP) [30] is a light weight protocol, which allows remote procedure calls through HTTP, so that servers can instruct proxy caches to do adaptation processing on the content (e.g. shrink an image). Active Cache [20] is a solution that allows servers to supply *cache applets* with the documents such that the proxy caches must run these applets on the documents before serving them (e.g. add a banner ad). The main complexity is how to write the code such that it is executable under different proxies from different vendors without compromising performance or security. Active cache uses a sand-boxed Java environment. Both Xenoservers [50] and Denali [52] require developers to write their code under a specialized operating system optimized for encapsulation and migration.

Application Servers [39][40][41] provide a strict API for system services, and hence it is feasible to move the application between different servers running the same application server. However, programmers do not strictly adhere to these APIs, which prevents application mobility. Java servlets [24] can be moved between servers running the Java virtual machine, but this approach suffers from performance degradation, which might not be justifiable for a web service. Also it requires that existing applications be rewritten within the Java environment, which presents a high switching cost. The Portable Channel Representation is an XML/RDF data model that encapsulates operating system and library dependencies to facilitate the copying of a service [19] across different systems. Again it requires the programmers to learn a new framework and port their existing work in to it, but it also does not provide isolation between servers belonging to different domains.

Finally, operating system virtualization, e.g. Ejasent [48] and Ensim [51], traps all operating system calls, hence allowing applications to be moved across virtual operating systems. The downside of this solution is that it is operating system dependent and imposes strict guidelines on what the applications can and cant do.

8 Conclusion

In this paper we proposed a novel solution for the distribution and replication of dynamic content. The solution is a network of real machines running virtual machine monitor software, hence allowing server virtual machines to be moved among the real machines. This

reduces the problem of dynamic content distribution to be similar to the problem of static content distribution, where the content being distributed is large virtual machine files.

We illustrated that it is possible to use off the shelf technologies to *mimic* the network environment surrounding the virtual machine such that it can be transparently moved without the need to reconfigure its software. We also discussed the performance challenges to fit existing two-tier services within such a framework.

References

[1] "The Measured Performance of Content Distribution Networks", 5th International Web Caching and Content Delivery Workshop, May 2000. Kirk L. Johnson, John F. Carr, Mark S. Day, M. Frans Kaashoek.

[2] "Web Caching and Content Distribution: A View from the Interior", 5th International Web Caching and Content Delivery Workshop, May 2000. Syam Gadde, Jeff Chase, and Michael Rabinovich.

[3] "Traffic Server Audited Performance Benchmark", Inktomi Corporation, 1998. *http://www.inktomi.com/new/press/1998/inkbench98.htm*.

[4] "A distributed test bed for national information provisioning", Duane Wessels, Traice Monk, k claffy, and Hans-Werner Braun.

[5] "Inktomi Traffic Server", Inktomi Corporation. http://www.inktomi.com/products/cns/products/tseclass.html

[6] "Akamai Content Distribution Network", Akamai, Inc. http://www.akamai.com/html/en/sv/content_delivery.html

[7] "Digital Island Content Distribution Network", Cable & Wireless. http://www.digisle.net/services/cd/.

[8] "A hierarchical Internet object cache", Proceedings of the USENIX 1996. Anawat Chankhunthod, Peter Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell.

[9] "Mirror Image InstaDelivery Internet services", Xcelera, Inc. http://www.mirror-image.com/

[10] "Squid Internet Object Cache". Duane Wessels et al. http://squid.nlanr.net

[11] "Cache Flow Edge Accelerators", Cache Flow, Inc. http://www.cacheflow.com/products/index.cfm/

[12] "A Publishing System for Efficiently Creating Dynamic Web Content", INFOCOM 2000. Jim Challenger, Arun Iyengar, Karen Witting, Cameron Ferstat, Paul Reed.

[13] "A Scalable System for Consistently Caching Dynamic Web Data", 18th Annual Joint Conference of the IEEE Computer and communications Societies, 1999. Jim Challenger, Paul Dantzig, Arun Iyengar.

[14] "Improving Web Server Performance by Caching Dynamic Data", 1997 USENIX Symposium on Internet Technologies and Systems. Arun Iyengar, Jim Challenger.

[15] "TIBCO Rendezvous Messaging System", TIBCO Software Inc. http://www.tibco.com/products/rv/index.html

[16] "Vignette Advanced Deployment Server", Vignette Inc. http://www.vignette.com/CDA/Site/0,2097,1-1-1329-2067-1345-2198,00.html

[17] "Edge Side Includes" http://www.edge-delivery.org/

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

[19] "Enabling Full Service Surrogates Using the Portable Channel Representation", 10th Intl. WWW Conference. Beck, Moore, Abrahamsson, Achouiantz and Johansson.

[20] "Active Cache: Caching Dynamic Contents on the Web", Proceedings of Middleware 1998. Pei Cao, Jin Zhang, and Kevin Beach.

[21] "On the Placement of Internet Instrumentation", IEEE Infocom 2000, Sugih Jamin, Cheng Jin, Yixin Jin.

[22] "Constrained Mirror Placement on the Internet", IEEE Infocom 2001. Sugih Jamin, Cheng Jin, Anthony R. Kurc, Danny Raz, Yuval Shavitt.

[23] "On the Placement of Web Server Replicas", IEEE Infocom 2001. L. Qiu, V. N. Padmanabhan, and G. Voelker.

[24] "Java Servlet Specification v2.3", Javasoft, Sun Microsystems, 2000.

[25] "Optimistic Deltas for WWW Latency Reduction", USENIX 1997. Gaurav Banga, Fred Douglis, and Michael Rabinovich.

[26] "Exploiting result equivalence in caching dynamic web content", Oct 1999, USENIX Symposium on Internet Technology and Systems. Ben Smith, Anurag Acharya, Tao Yang, and Huican Zhu

[27] "Cooperative Caching of Dynamic Content on a Distributed Web Server", 1998, 7th IEEE Intnl Symposium on High Performance Distributed Computing. V. Holmedahl, B. Smith, and T. Yang.

[28] "A Digital Fountain Approach to Reliable Distribution of Bulk Data", ACM SIGCOMM 1998, John W. Byers, Michael Luby, Michael Mitzenmacher, Ashutosh Rege.

[29] "Scalable Web Caching of Frequently Updated Objects using Reliable Multicast", 1999, USENIX Symposium on Internet Technologies and Systems. Dan Li and David R. Cheriton.
[30] "ICAP: Internet Content Adaptation Protocol" http://www.i-cap.org/icap/media/draft-elson-opes-icap-01.txt
[31] "A Dynamic Object Replication and Migration Protocol for Internet Hosting Service", 19th International Conference on Distributed Computing Systems, Austin, Texas, June 1999, IEEE. M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal.
[32] "HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching". USENIX Symposium on Internet Technologies and Systems, Monterey, California, USA , December 1997. Fred Douglis, Antonio Haro, and Michael Rabinovich.
[33] "Adaptive Push-Pull: Disseminating Dynamic Web Data", Tenth International World Wide Web Conference, Hong Kong , May 2001. Pavan Deolasee, Amol Katkar, Ankur Panchbudhe, Krithi Ramamritham, and Prashant Shenoy.
[34] "Class-Based Cache Management for Dynamic Web Content", IEEE Infocom 2001. Huican Zhu and Tao Yang.
[35] "Survey of Virtual Machine Research", IEEE Computer, June 1974. R. P. Goldberg.
[36] IBM Virtual Machine 370, 1972. IBM Corporation http://www-1.ibm.com/ibm/history/history/year_1970.html
[37] "Java Virtual Machine Specification, The, Second Edition", 1999, Addison-Wesley. Tim Lindholm and Frank Yellin. ISBN: 0201432943.
[38] "Organization-Based Analysis of Web-Object Sharing and Caching", 1999 USENIX Symposium on Internet Technologies and Systems. Alec Wolman, Geoffrey M. Voelker, Nitin Sharma, Neal Cardwell, Molly Brown, Tashana Landray, Denise Pinnel, Anna R. Karlin and Henry M. Levy.
[39] ATG Dynamo Application Server, ATG. http://www.atg.com/en/products/das.jhtml
[40] IBM WebSphere Application Server. IBM Corporation. http://www.ibm.com/software/webservers/appserv
[41] BEA Systems WebLogic Application Server , BEA Systems. http://www.beasys.com/products/weblogic/server/index.shtml
[42] "RFC1918: Address Allocation for Private Intranets"
[43] "RFC1631: The IP Network Address Translator"
[44] "RFC2764: A Framework For IP Based Virtual Private Networks"
[45] "RFC2709: Security Model with Tunnel-mode IPSec for NAT Domains"
[46] "The macroscopic behavior of the TCP Congestion Avoidance algorithm", Computer Communications Review, July 1997. M. Mathis, J. Semke, J. Mahdavi, T. Ott.
[47] "RFC2131: Dynamic Host Configuration Protocol"
[48] "Ejasent: Making the Net Compute", Ejasent Inc. http://www.ejasent.com
[49] "End-to-end WAN Service Availability", Third Usenix Symposium on Internet Technologies and Systems, March 2001. B. Chandra, M. Dahlin, L. Gao, A. Nayate.
[50] "Xenoservers; Accounted execution of untrusted code", IEEE Hot Topics in Operating Systems (HotOS) VII, March 1999. Dickon Reed, Ian Pratt, Paul Menage, Stephen Early, Neil Stratford
[51] "Ensim: Hosting Automation Solutions", Ensim Inc. http://www.ensim.com
[52] "Denali: A Scalable Isolation Kernel", Proceedings of the Tenth ACM SIGOPS European Workshop, France, Sept. 2002. Andrew Whitaker, Marianne Shaw, and Steven D. Gribble.
[53] "Opus: an Overlay Peer Utility Service", Proceedings of the 5th International Conference on Open Architectures and Network Programming (OPENARCH), June 2002. Rebecca Braynard, Dejan Kostic, Adolfo Rodriguez, Jeffrey Chase, and Amin Vahdat.