Resource Allocation across Multiple Cloud Data Centres

Barnaby Malet
Imperial College London
180 Queen's Gate
London SW7 2AZ, UK
bwm05@doc.ic.ac.uk

Peter Pietzuch
Imperial College London
180 Queen's Gate
London SW7 2AZ, UK
prp@doc.ic.ac.uk

ABSTRACT

Web applications with rich AJAX-driven user interfaces make asynchronous server-side calls to switch application state. To provide the best user experience, the response time of these calls must be as low as possible. Since response time is bounded by network delay, it can be minimised by placing application components closest to the network location of the majority of anticipated users. However, with a limited budget for hosting applications, developers need to select data centre locations strategically. In practice, the best choice is difficult to achieve manually due to dynamic client workloads and effects such as flash crowds.

In this paper, we propose a cloud management middleware that automatically adjusts the placement of web application components across multiple cloud data centres. Based on observations and predictions of client request rates, it migrates application components between data centres. Our evaluation with two data centres and globally distributed clients on PlanetLab shows that our approach can decrease median client response times by 21% for a realistic multi-tier web application.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications

General Terms

Management, Performance, Experimentation, Design

Keywords

Resource allocation, Management middleware, Web applications, Cloud computing

1. INTRODUCTION

The benefits of hosting web applications in the cloud are becoming increasingly attractive to both individuals and businesses. By using resources on demand in cloud data centres, developers can significantly reduce management and deployment overheads. *Infrastructure-as-a-Service* (IaaS) solutions, such as Amazon Web Services [3] and Rackspace Cloud [16], operate multiple, geographically-distributed data centre locations with resources for computation, storage and communication. Locations are priced differently based on bandwidth and computational usage and provide different performance to clients based on their network location.

Many of today's web applications expose rich *user interfaces* (UIs) written in JavaScript, which are designed to give a user experience similar to that of native desktop applications. They commonly employ a method called *Asynchronous JavaScript And XML* (AJAX) [12] to fetch further content after the application has loaded. AJAX enables the web browser to make background calls to the remote web server. This can be used to change UI elements without reloading the page and to provide responsive behaviour. The quality of the user experience is strongly correlated with the response time of the AJAX calls—high response times lead to sluggishness of the UI.

Therefore it is an important decision in which cloud data centre to host an application given a fixed budget: placing application components close to a large user base results in better quality-of-service due to lower latency and increased throughput; however, it may also incur a higher cost if resources at that location are more expensive. This problem is complicated by the fact that resource allocation decisions have to take changes of the workload into account.

Resources are available as *virtual machines* (VMs) that appear as traditional servers. Techniques for migrating VMs across LANs [8] are widely available but the ability to migrate VMs across WANs remains more challenging [23]. We argue that these mechanisms should be used to strategically place application VMs across disparate geographic locations in order to optimise user-perceived latency and cost.

To achieve this goal, we propose a *cloud management middleware* that strategically migrates VMs of multi-tiered web applications between data centre sites in response to workload changes. Our middleware monitors client-server workload at the data centre gateway. This information is used by a control plane to initiate VM migrations between data centres in order to move application components closer to clients. The web clients are aware of the migration mechanism and redirect their traffic to the newly migrated VM.

This approach achieves lower network round trip times, improving client response times in the web application, without requiring additional VMs. Our evaluation results on the global PlanetLab test-bed [6] together with two data centres show that this approach can improve the median response time of a web application by 21% and the 80th percentile by 7% over a single data centre deployment.

In summary, the main contributions of this paper are:

1. an approach for reorganising two-tiered web applications by allocating resources across data centres in response to workload changes;

2. a design of a cloud management middleware that dynamically migrates VMs based on client workload to provide maximum quality of service to clients; and

3. a prototype middleware implementation that optimises the execution of a web application, and its evaluation with two data centres and many clients on PlanetLab [6].

In the next section, we motivate our work. Section 3 describes the design of the cloud management middleware focusing on the VM migration and placement algorithms. We give results from an Internet deployment in Section 4. The paper finishes with related work in Section 5 and conclusions in Section 6.

2. MOTIVATION

Modern Internet applications are typically designed using a multi-tier architecture, in which each tier is responsible for a subset of the application's computation and/or storage. Typical examples are the RUBiS [2] and TCP-W [5] web applications, which consist of a web server tier, an application tier and a database tier.

Due to recent advances in virtualisation technology [4, 20] and its associated benefits in data centre management, applications are no longer hosted on dedicated machines but instead placed within VMs. This shift has permitted researchers to develop autonomic management solutions that can resolve server hot-spots and performance bottlenecks using VM migration and provisioning [19, 24]. These techniques have been used by cloud providers to increase performance and ultimately provide better service level agreements (SLAs).

The widespread use of content delivery networks (CDNs) demonstrates the advantages of placing web servers close to clients [14]. However, the entry cost for using a CDN remains high and often beyond what small businesses can afford. A further disadvantage of CDNs is that they focus on content distribution and thus do not cater for custom computation and persistence. These short-comings coupled with the geographic diversity of cloud compute sites provide a strong motivation for a cloud management middleware that optimises the placement of VMs.

To illustrate our idea, consider a scenario in which a web start-up deploys a multi-tiered web application in the cloud. It has a limited budget and thus can only afford a small number of VM instances. The choice of data centre locations is large and growing steadily, and the start-up must decide how to distribute its resources among the sites in order to provide the best service to its users.

This problem is non-trivial: different peak times across geographic regions mean that it is difficult to estimate where the majority of users will be located at any given time. In addition, the application may face unpredictable traffic of varying intensity and duration, such as flash crowds [10]. The

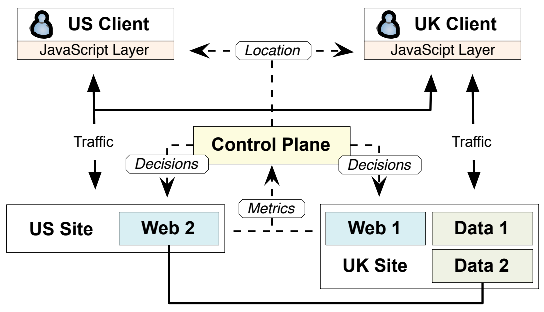

Figure 1: Cloud management middleware with two deployed web applications. Each application consists of an application tier (Web 1 or Web 2) and a storage tier (Data 1 or Data 2).

decision to move or create a VM close to the clients may provide a benefit (i.e., decreased latency or increased throughput) but may also incur a cost (i.e., moving data and/or increased hosting fees). Since client workload changes can occur rapidly and unexpectedly, such decisions have to be periodically re-evaluated. We argue that this should be done by a cloud management middleware that makes these decisions dynamically at a fine granularity.

3. CLOUD MIDDLEWARE DESIGN

Our solution is to augment a traditional IaaS offering such as AWS [3] with a cloud management middleware. Our design aims to reduce the client response times of two-tiered web applications consisting of an *application tier* and a *storage tier*. It achieves this by dynamically moving application tiers and their associated data closer to clients based on the current and previous workloads.

Our middleware consists of the components shown in Figure 1. Clients bootstrap by downloading application code (such as HTML/JavaScript) from a web server whose IP address is resolved via DNS. The application code is static and can be distributed via traditional means (such as CDNs).

Clients track the movement of the application tiers using a *JavaScript layer* through which all server-side calls are passed. This means that application code can transparently refer to remote application components without having to know their current IP address. (This imposes the constraint that anchor tags must refer to a JavaScript function rather than a URL—the JavaScript layer then forms the URL based on the current server IP address.)

A *control plane* uses the client workload observed at the data centres to decide where best to place VMs. It coordinates VM migrations and traffic routing: When a migration decision has been made, a new VM image is instantiated at the target location and the application state is transferred. The clients then re-direct their traffic to the new site.

In terms of strategy (cf. Section 3.3), the control plane migrates application tier VMs freely between sites—we assume that they are stateless and have a low migration cost. In contrast, VMs supporting the storage tier are placed at the location that has historically seen the highest access. This is because data migration is considered to consume more resources.

In the example in Figure 1, both tiers of application 1

(Web 1 and Data 1) are placed in the UK, close to the majority of clients. Application 2 is reacting to a sudden increase in usage from the US by moving its application tier (Web 2) there, but leaving its database tier (Data 2) in place.

3.1 Web application architecture

Modern web applications are frequently built using the *representational state transfer* (REST) [11] architecture. It enforces a strict separation between the client, which manages application state, and the web service, which is invoked by the client upon a change in application state. At any point in time, a client can either be in transition between application states or “at rest”. When a client is at rest, it exerts no load on the server; commonly, a RESTful architecture relies on the HTTP protocol for communication. As opposed to a traditional web application such as RUBiS where the line between client and server responsibility is blurred because the server maintains application state, a RESTful web service is stateless.

Building a RESTful web application allows developers to draw clear lines of responsibility between components and thus simplifies their implementation. Because RESTful applications are stateless, components can easily be duplicated to adapt to increases in load. Furthermore intermediaries such as proxies, gateways and firewalls can be introduced without changing the interface between components. These advantages make this type of architecture popular in modern web applications and we assume it throughout the rest of the paper.

Modern web applications make heavy use of JavaScript on the client side to provide a rich user experience similar to desktop applications. In our work, we exploit this to get around the problem of maintaining VM addresses during a WAN migration: we extend client code with an intermediate JavaScript layer that acts as a proxy for all client-issued RESTful calls. This intermediary is aware of the current network location of the VM in terms of its IP address. It maintains a persistent HTTP connection to the control plane, which is used to notify the client of a VM migration; when this occurs, the JavaScript layer redirects RESTful calls accordingly. This technique requires a browser that supports the W3C Cross-Origin Resource Sharing recommendation [21] that permits JavaScript calls to web servers other than those from which the application code originated.

This approach allows for fine-grained control of load-balancing for this type of web application, which other techniques, such as round-robin DNS and HTTP redirecting, do not support. By providing a JavaScript client library, web applications can be extended to handle VM mobility with only minor changes to their code base.

3.2 Virtual machine migration

Mechanisms for migrating VMs across WANs remain elusive. Migrating a live VM requires high speed networks, access to a shared block device and control over the network address space [8]—all of which are not available when moving VMs between data centres over the Internet. Since resolving these issues requires changes to hypervisors and networks [23], we instead assume that a consistent image of each VM is available at every data centre. The middleware can then use a start-stop mechanism for migration, while coordinating with the clients to redirect traffic, as described previously. Next we explain this approach in more detail.

3.2.1 Application tier VMs

Since we assume that applications are built using a RESTful architecture, they are stateless and can be migrated by starting a new instance at the target location and stopping the old instance. In practice, an application tier may maintain an in-memory cache to reduce calls to the data tier; in these cases, the newly-instantiated application would lazily re-populate the cache using the data tier.

3.2.2 Storage tier VMs

We assume that the storage tier contains the application state and its VMs must be migrated differently from the application tier VMs. Web applications often maintain a shared database of user-generated content. For RUBiS and TCP-W, it is in the range of a few hundred megabytes to a few gigabytes [2]. This data is challenging to replicate at multiple remote sites because it must remain consistent.

For simplicity, we opt to maintain a single active copy of the data at any point in time. Migration is therefore implemented as part of the application: to migrate a storage tier VM, an identical VM with an empty database is instantiated at the target location. The data is then iteratively copied to the target. Any concurrent database changes to the source data are also mirrored at the target. After the data has been copied, the target becomes the active copy and the source is switched off.

This approach is application-specific, and requires that the storage system is able to maintain a synchronised copy of the data during transfer. It incurs a considerable network cost for transferring the entire state but the middleware tries to migrate storage tiers infrequently. As future work, storage tier migration could be improved using a distributed file system that handles consistency between replicas [18].

3.3 Virtual machine placement

The middleware uses the following simple algorithms to translate the workload observations into migration commands for VMs hosted at data centres.

3.3.1 Placement goals

There are many potential goals when optimising the placement of VMs: reducing monetary costs [15], increasing application throughput or decreasing latency. The specific goal depends on the nature of the underlying application. For example, a large map-reduce job [9] that is not time critical may strive to reduce monetary costs by choosing sites that offer the least expensive compute and network transfer rates. Web applications may have SLAs to meet, aiming to reduce response times as much as possible. In the case of web applications that expose Javascript-driven GUIs using AJAX, reducing network round-trip times (RTT) is key.

Another decision to be made is which clients should achieve a given goal. For example, the middleware may prioritise a subset of clients belonging to paying customers and reduce their response times. Alternatively, it could try to achieve a minimum SLA such that all clients would be guaranteed a maximum bound on their RTTs when supporting an interactive web GUI. In this work, we choose to reduce the response times of web requests for the majority of globally distributed clients. We leave more complex placement goals that trade-off gain and cost for future work.

3.3.2 *Placement algorithms*

We now present the algorithms that decide the placement location of the web application tiers. We treat the migration of application and storage VMs separately because storage VMs are more expensive to migrate due to communication costs. The middleware employs a *reactive* algorithm for the placement of application VMs and a *predictive* algorithm for storage VMs.

Reactive placement algorithm.

The goal of the reactive placement algorithm is to respond quickly to variations in application workload by moving application tier VMs. Intuitively, the system considers the optimal location of application VMs to be the site closest to the majority of the clients issuing requests within a given time window.

The control plane maintains a sliding window for each hosted web application. The window contains one entry per client web request recording its origin location and size; as new requests reach the servers, they are added to the window. The control plane also maintains a list of available data centre locations DC.

At regular intervals, the control plane iterates through the window counting requests per data centre location. A request is counted for a site if that site is closest to the requesting client in terms of geographic distance. Each site x is then assigned a value λ, which is the weighted sum of the requests at its origin:

$$\lambda(x) = \sum_{i=0}^{|\text{requests}(x)|} W_i$$

where W_i are the request weights, which can be tuned to prioritise some client requests over others. The new placement location is chosen as the data centre with the highest λ value:

$$\max\left(\lambda(x) \mid x \in DC\right)$$

The size of the window used to record the requests governs how sensitive the system is to changes in client workload. In our experiments in Section 4, we empirically determined a window size of 10,000 requests to be rigid enough to prevent unnecessary oscillations but sufficiently flexible to permit migrations that compensate for a large increase in requests from a site. In general, we would expect the window size to be chosen adaptively to achieve a desired number of migrations.

Predictive placement algorithm.

Because storage tiers are considered costly to migrate, their location must be chosen more strategically. In order to achieve a reasonable degree of accuracy when placing storage VMs, we use a predictive algorithm inspired by the technique employed by Rolia et al. [17]. It uses past observations of application workload to predict future peak demands.

The placement location is predicted at the granularity of hours. The best location of a storage VM for a given hour is the location that was closest to the majority of client accesses for that hour over the past several days. A storage VM is migrated at hour $t + 1$ if the predicted location is different from that at hour t and remains the same over the next w hours. Intuitively, this algorithm splits up a day into contiguous periods of at least w hours, moving storage VMs at the boundary between these periods.

This approach assumes that there are reoccurring patterns between days (cf. Figure 2). This means that it cannot adapt to unpredicted events such as flash crowds. However, such events would be handled by the reactive placement algorithm above. More advanced prediction techniques such as a combined reactive and predictive approach [19] could be used instead.

4. EVALUATION

The goal of our experimental evaluation is to demonstrate that our cloud management middleware decreases the response time of a globally accessed web application. By placing VMs strategically, it can achieve this without requiring additional resources in the data centres.

4.1 Experimental set-up

The prototype deployment of our system consists of two data centre sites, one in the US hosted at the Emulab testbed facility [22] and another in the UK hosted in the data centre of our university. We use the Xen Hypervisor [4] to multiplex each server into a number of separate VMs, which host individual application and storage tiers.

We use four virtual machines to host the various tiers in all experiments—two in the US and two in the UK. The application tiers use an Apache web server running PHP scripts that generate static data or retrieve data from a MySQL database in the storage tier. Client requests to the web server include the desired size of the response document and a flag indicating its source (database or static). A hard limit on the request size of 300 KB is set to put a bound on the amount of network resources used in the experiments.

For the application workload, we use a 24-hour request trace from our department's web server. The trace is an Apache log file that contains HTTP status codes, request sizes, request times and client IP addresses. We adapt the trace in the following ways: (1) Only HTTP "OK (200)" status codes are kept. (2) All internal university requests are removed. (3) All requests from locations outside the US or the UK are discarded. To expedite experimentation, the 24-hour trace is uniformly compressed into a 2-hour run. Since we do not know how many of the requests in the original trace involved database accesses, we assume 10% of the requests to require database transactions due to efficient caching.

We use six PlanetLab nodes to emulate requests from a world-wide client population. The clients are split into EU and US clients. Each group is placed at a minimum 800 km radius from their respective data centres. The EU (respectively US) clients are assigned the UK (respectively US) portions of the trace to replay. Since we use three clients per sub-trace, this is equivalent to tripling the trace, resulting in a mean request rate of 28.6 requests/second over the two hour period.

The clients replay the transformed trace by implementing a multi-threaded HTTP downloader. For simplicity of implementation, we did not implement the JavaScript layer on the clients, instead all URLs in the workload reference the IP addresses of gateways placed in the data centre. When the control plane makes a migration decision, the gateways issue redirect responses to redirect the client traffic to the new location.

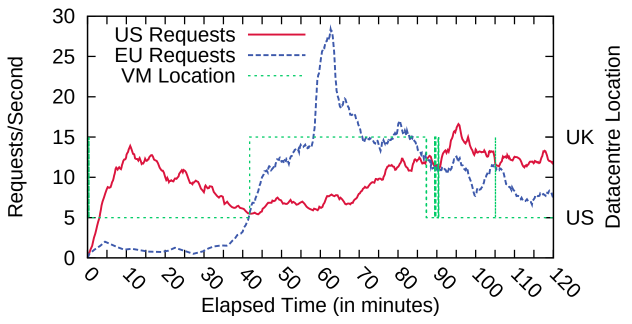

Figure 2: Observed client request rate, split into US and EU requests.

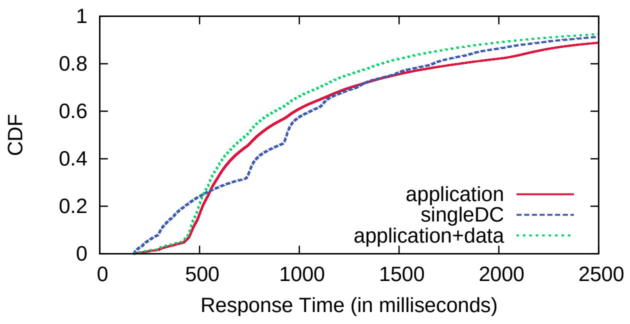

Figure 3: Distribution of client response times under different resource allocation strategies.

4.2 Results

Figure 2 shows the split in client request workload over time based on client location as observed at the application servers. As mentioned previously, the workload is obtained by uniformly compressing a day trace of our department's web servers into a 2-hour period. Due to our compressed replay of the trace, each hour in the original trace is mapped to a 5 minute slot in the workload. Minute 0 corresponds to 1:21am in the original trace, minute 5 corresponds to 2:21am and so forth—the intersection of the request rates at minute 40 corresponds to 9:21am in the original trace.

The VM Location line shows the location of the application VM as decided by the reactive placement algorithm, where a low (respectively high) value indicates that the VM is in the US (respectively UK). We can see that the location of the VM follows closely the user base issuing the majority of requests at any given time: The VM is in the US for the first 40 minutes and then moves to the UK because the UK rate becomes higher than the US rate. Similarly, after 90 minutes, it switches back to the US. There are oscillations causing the VM to move back and forth between continents after around 90 minutes because the UK and US request rates are similar. Such oscillations are unwanted side effects of workload variations and we are investigating dampening approaches to suppress them.

The distribution of client response times are shown in Figure 3. Response time is measured at the client as the time elapsed between a request being made and the full receipt of the server's response. We compare three resource allocation strategies: (1) singleDC places all VMs in the UK data centre and does not migrate them; (2) application is the reactive placement algorithm with a window size of 10,000 to move the application VMs only without migrating the storage VMs; and (3) application+data also migrates the storage VMs using the predictive placement algorithm. The placement algorithm is trained with "normal" days that did not exhibit any request surges in the workload.

In both reactive approaches, the control plane places the application components closer to the majority of clients at any given time. The results show that the application+data approach achieves a 21% decrease in client response times over the singleDC approach for the median and a 7% decrease for the 80th percentile. The application approach does not do as well, with a 15 % median decrease but a 7 % increase at the 80th percentile.

The lower performance is caused by the dependency between the application and storage tiers when the application tier is placed in the US: 10% of requests involve database accesses that require transatlantic communication between the two data centres. Since this happens over a slower network path, it is faster in these cases for the client to directly access the UK site. We believe that the performance could be improved if the data centres had better network connectivity or less data was transferred between the two tiers. This could be achieved by caching or optimisations at the application level.

Around 25% of requests perform significantly better in the singleDC allocation. At any point in time, the web application has requests originating from both regions. In cases where the adaptive approaches have placed application components in the US in response to client locality changes, the minority number of requests originating from EU clients exhibit significantly higher response times.

During experimentation, we observed that improvements are sensitive to the locality of the clients in relation to the data centres. To illustrate this, consider the following scenario that we encountered: an application runs in a London data centre, with the majority of clients in London. A sudden increase in requests from the US causes the application to move to the US site. Although the US clients benefit from this move, this is not reflected as an overall improvement because the London clients experience a drastic performance decrease. This is an interesting trade-off and relates back to our discussion on optimisation goals in Section 3.3.1—we leave this for future work.

5. RELATED WORK

Urgaonkar et al. [19] strategically allocate VMs to application tiers in web applications within a single data centre. Their work requires more granular workload prediction to provision the correct number of VMs per application tier. Volley [1] addresses the problem of data placement in a multi data centre context to reduce data access latency. They employ an iterative optimisation algorithm based on access patterns and data locations. They target large applications and their optimisation is performed at the order of days—in contrast to our fine-grained placement decisions.

To the best of our knowledge, we are unaware of work that places VMs closer to users on the Internet. However, migrating VMs for improved performance within a data centre has been widely studied in recent years. Sandpiper [24] alleviates server resource hot-spots using migration. Memory-buddies [25] increases data centre capacity by placing similar VMs on the same host and sharing memory between them.

There does not yet appear to be consensus as to how to perform live VM migrations across WANs. VMWare has recently provided support for WAN migrations but requires 622 MBps inter-data centre bandwidth and less than 5 msec network delay [7]. Wood et al. [23] improve upon this by optimising memory and disk transfers to migrate a live VM across a low bandwidth (50–100 MBps), high latency link.

Persisting a VM's network address is a fundamental challenge in WAN migrations. CloudNet combines cloud resources with a VPN. Hao et al. [13] propose a network architecture in which VM migrations are coordinated with a new routing mechanisms to provide WAN mobility. This requires modifications to existing routers and network infrastructure and thus is more complex to implement than our solution—our algorithms, however, could be integrated with their system.

6. CONCLUSIONS

Global and dynamic application deployment in the cloud is becoming possible and we expect its study to gain more traction in the near future. We have described the design and evaluation of a cloud management middleware that can reduce web application response times by strategically placing application components in VMs. It handles the migration of application tier VMs reactively and uses a predictive approach for the more costly migration of storage tier VMs. Our evaluation results with a global client base show the promise of this approach for managing VMs across multiple data centres.

In the immediate future, we plan to extend our placement algorithms to cater for flash crowds and other unexpected workload peaks. We also want to investigate ways of strategically placing data across different regions to improve the reactive aspect of our system. Finally, we will factor migration and operational costs into the optimisation algorithms, and provide support for n-tier applications and a larger number of data centre locations.

References

[1] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan. Volley: Automated Data Placement for Geo-Distributed Cloud Services. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, April 2010.

[2] C. Amza, A. Ch, A. L. Cox, S. Elnikety, et al. Specification and Implementation of Dynamic Web Site Benchmarks. In *5th IEEE Workshop on Workload Characterization*, 2002.

[3] Amazon Web Services. `http://aws.amazon.com`.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, et al. Xen and the Art of Virtualization. In *19th ACM Symposium on Operating Systems Principles (SOSP)*. ACM, 2003.

[5] H. W. Cain, R. Rajwar, M. Marden, and M. H. Lipasti. An Architectural Evaluation of Java TPC-W. *Int. Symposium on High-Performance Computer Architecture*, 2001.

[6] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: An Overlay Testbed for Broad-coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3):12, 2003.

[7] Cisco. Virtual Machine Mobility with Vmware VMotion and Cisco Data Center Interconnect Technologies, 2009.

[8] C. Clark, K. Fraser, S. Hand, J. Hansen, et al. Live Migration of Virtual Machines. In *2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2005.

[9] J. Dean and S. Ghemawat. Map Reduce: Simplified Data Processing on Large Clusters. *Communications of the ACM-Association for Computing Machinery (CACM)*, 51(1):107–114, 2008.

[10] J. Elson and J. Howell. Handling Flash Crowds from your Garage. In *USENIX Annual Technical Conference (ATC)*, Boston, MA, 2008.

[11] R. Fielding. *Representational state transfer (REST)*. PhD thesis, University of California, Irvine, CA, 2000.

[12] J. Garrett. Ajax: A New Approach to Web Applications. `http://www.adaptivepath.com/ideas/essays/archives/000385.php`, 2005.

[13] F. Hao, T. Lakshman, S. Mukherjee, and H. Song. Enhancing Dynamic Cloud-based Services using Network Virtualization. In *1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*, 2009.

[14] A. Pathan and R. Buyya. A Taxonomy and Survey of Content Delivery Networks. Technical report, Grid Computing and Distributed Systems (GRIDS) Laboratory, University of Melbourne, Parkville, Australia, 2006.

[15] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs. Cutting the Electric Bill for Internet-scale Systems. *ACM SIGCOMM Computer Communication Review*, 39(4):123–134, 2009.

[16] Rackspace Cloud. `http://www.rackspacecloud.com/`.

[17] J. Rolia, X. Zhu, M. Arlitt, and A. Andrzejak. Statistical Service Assurances for Applications in Utility Grid Environments. Technical Report HPL-2002-155, HP Labs, 2002.

[18] J. Stribling, Y. Sovran, I. Zhang, X. Pretzer, J. Li, M. F. Kaashoek, and R. Morris. Flexible, Wide-area Storage for Distributed Systems with WheelFS. In *6th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2009.

[19] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. Agile Dynamic Provisioning of Multi-tier Internet Applications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 3(1):1–39, 2008.

[20] VMWare ESX Server. `http://www.VMware.com/ESX-Server`.

[21] W3C. Cross-Origin Resource Sharing. `http://www.w3.org/TR/cors/`.

[22] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Dec. 2002.

[23] T. Wood, K. Ramakrishnan, P. Shenoy, and J. Van der Merwe. CloudNet: A Platform for Optimized WAN Migration of Virtual Machines. In *University of Massachusetts Technical Report TR-2010-002*, January 2010.

[24] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Black-box and Gray-box Strategies for Virtual Machine Migration. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2007.

[25] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, et al. Memory Buddies: Exploiting Page Sharing for Smart Colocation in Virtualized Data Centers. *ACM SIGOPS Operating Systems Review*, 43(3):27–36, 2009.