vHaul: Towards Optimal Scheduling of Live Multi-VM Migration for Multi-tier Applications

Hui Lu[†],Cong Xu[†],Cheng Cheng[†],Ramana Kompella[†‡*],Dongyan Xu[†]
[†]Department of Computer Science, Purdue University, [‡]Google, Inc.

***Abstract*—Live virtual machine (VM) migration enables seamless movement of an online server from one location to another to achieve failure recovery, load balancing, and system maintenance. Beyond single VM migration, a multi-tier application involves a group of correlated VMs and its live migration will require careful scheduling of the migrations of the member VMs. Our observations from extensive experiments using a variety of multi-tier applications suggest that, in a dedicated data center with dedicated migration links, different migration strategies result in distinct performance impacts on a multi-tier application. The root cause of the problem is the inter-dependence between functional components of a multi-tier application.**

We leverage these observations in vHaul, a system that coordinates multi-VM migration to approximate the optimal scheduling. Our evaluation of a vHaul prototype on Xen suggests that vHaul yields the optimal multi-VM live migration schedules. Further, our application-level evaluation using Apache Olio, a web 2.0 cloud application, shows that the optimal migration schedule produced by vHaul outperforms the worst-case schedule by 43% in application throughput. Moreover, the optimal schedule significantly reduces service latency during migration by up to 70%.

***Keywords*-Virtualization; Cloud Computing; Live Migration;**

I. Introduction

Live VM migration techniques (e.g., XenMotion [1] and vMotion [2]) have been increasingly adopted in the cloud to achieve seamless movement of online services – executed by VMs – from one physical host to another by transferring active memory, CPU and storage states. However, resource contention during migration could result in significant performance degradation to a VM's workload [3, 4, 5]. While most state-of-the-art live VM migration techniques [6, 7, 8] mainly focus on minimizing the performance impact on *single VM* migration, less effort is made in understanding multi-VM migration.

In a virtualized cloud infrastructure, multi-tier applications consisting of multiple functional components are usually deployed in *multiple inter-dependent VMs* [9]. Such inter-dependent VMs are subject to migration as a group within a data center or across various data centers [10]. Recently, COMMA [11] sheds some light on live migration of multi-tier applications, which discovered that the performance of a multi-tier application can severely degrade, if the dependent components become split across a high-latency network path. To mitigate such impact, COMMA proposes to migrate a group of related VMs simultaneously – by starting and finishing the migration of related VMs at the same time, hence minimizing the VMs' communication via the high latency network path.

With the intention of minimizing the performance impact revealed by COMMA, we conducted live multi-VM migration within a dedicated data center environment, which offers low network latency between any two physical machines (e.g., less than 1 ms). Further, we designated a dedicated network link for VM migration to avoid the interference between application traffic and VM migration traffic. Surprisingly, the performance degradation of a multi-tier application still exists and sometimes becomes significant. In addition, migrating groups of related VMs simultaneously, as suggested by COMMA, does not seem to be the optimal option under our environment.

In the hope of finding other major factors impacting the performance of live multi-VM migration for a multi-tier application, we then conducted an extensive measurement study using a variety of multi-tier applications. Our results suggest that, in a dedicated data center with dedicated migration links, different migration strategies (e.g., *sequential migration* and *parallel migration*) could likewise result in distinct performance impacts on a multi-tier application. Further, the performance gap between different migration strategies becomes increasingly large as the application workload increases.

Furthermore, using measurement results, controlled experiments, and queueing theory, we identify the root cause of the problem as the inter-dependence between functional components of a multi-tier application. More specifically, in the sequential migration case (i.e., VMs migrating one after another), the pending requests backlogged from the VM that has just migrated will propagate to the following migration phase, negatively impacting the performance of the next VM to be migrated. Owing to various characteristics of each tier – loads of varying magnitude and duration of VM migration – different sequential migration orders may result in drastically different application-level performance impacts. While in the parallel migration case (i.e., VMs migrating simultaneously), the application performance tends to be more affected than in the sequential case, as the performance

[*]Contributed to this work while at Purdue University.

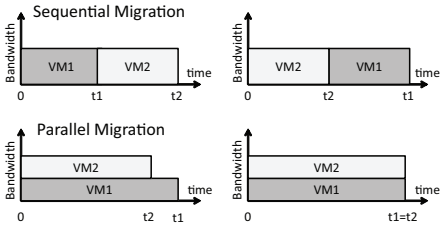


Figure 1. Strategies to migrate a multi-tier application.

degradation during parallel migration is caused by all VMs instead of one (in the sequential case).

With this observation, we propose a simple yet effective solution, called vHaul, that coordinates multi-VM migration to approximate the optimal scheduling. vHaul covers two typical migration scenarios. (1) Without a constraint on end-to-end migration time, the least performance impact can be achieved by migrating VMs one by one, *separated by a no-migration interval* between two VM migrations. This way the pending requests will be processed during the no-migration interval. (2) To complete the end-to-end migration without any delay, vHaul determines an optimal VM migration order according to the VMs' resource utilization and estimated migration time, with the goal of reducing the impact of pending requests. Our evaluation results validate the effectiveness of vHaul. Our results with application-level benchmarks (Olio) show that the application throughput under the migration schedule computed by vHaul outperforms the worst-case migration schedule by 43%. Moreover, vHaul significantly reduces application request processing latency during the migration by up to 70%.

The main contributions of this paper are summarized as follows: (1) We observe and demonstrate that the functional inter-dependence between participating VMs leads to varying degree of performance degradation for a multi-tier application, under different migration strategies (Section II and III). (2) We propose vHaul as a simple approach to mitigating such impact that can be deployed for a range of cloud application scenarios (Section IV). (3) We have implemented a prototype of vHaul based on Xen [12] to coordinate multi-VM migration and demonstrated improvement in application-level performance (Section V).

II. Investigation and Observations

A. Multi-VM Live Migration Background

In real world, most applications deployed in the cloud are multi-tier, consisting of several interactive VMs each running logically separated but inter-dependent workload (e.g., web servers, application servers and database servers). All (or part) of these VMs may need to be migrated as a group during cloud runtime. Existing solutions[13, 14, 6, 15] work well for migrating an individual VM. Yet, few of them can be applied to migrate a group of inter-dependent VMs.

In general, there are two strategies for multi-VM live migration as shown in Figure 1. (1) *Sequential migration*,

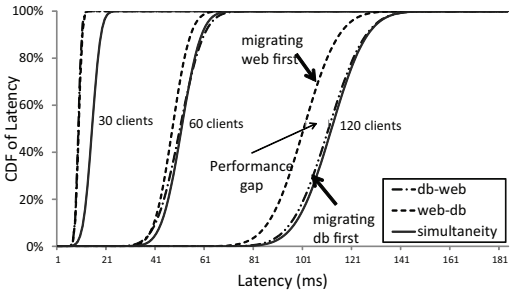


Figure 2. CDF of latency for various loads (30 clients, 60 clients and 120 clients) in RUBiS.

VMs are migrated one by one. In the two-VM case shown in Figure 1, there are two optional sequential schemes with reverse orders – VM1 first or VM2 first. (2) *Parallel migration*, VMs are migrated simultaneously. One scheme is to start the migration of all VMs at the same time but may stop at different times; another scheme is to start and stop the migration of all VMs at the same time [11].

Live migration in the single VM case does negatively impact on the performance of the VM [16]. Intuitively, the two significant performance influential factors – migration link bandwidth and page dirty rate – are also critical in the multi-VM scenario. In addition, COMMA [11] claimed that the performance of a multi-tier application may suffer from severe degradation if its dependent components become split across a *high latency* network path.

Nonetheless, in today's data centers the migration link is usually separated from the production network link to avoid performance interference. That is, we can have a high-speed network dedicated for VM migration traffic only (i.e. vMotion [2]). Considering the latency of modern network fabrics is typically sub-millisecond, the communication problem mentioned above should not be significant, if the multi-VM migration happens within the same data center. We will show this in our measurement study using a dedicated data center environment with designated migration links.

B. Measurement Methodology

We adopt a 2-tier web system consisting of a web server running in VM1 and a database server running in VM2. We study the application-level performance impact imposed by three multi-VM migration schemes. They are (1) migrating VM1 first and then VM2, (2) migrating VM2 first and then VM1, and (3) migrating both VMs simultaneously. As the 2-tier model is common in request-response multi-tier applications and usually serves as the basic unit in more complex multi-tier applications, we believe the observations based on this 2-tier model are helpful and generalizable for studying more complex models.

We measure the performance of the web service by computing the average response time of each request from the client side. Since a web request is typically composed of various sub-types (e.g., "preview", "purchase", etc.), we em-

ploy the geometric mean to represent the mean response time of all sub-types. The overall application-level performance during migration is gauged by averaging all response times in the entire migration duration, called the average latency. For each migration scheme, we run the experiments 30 times and plot the CDFs of the average latency. Note that higher average latency means worse application performance.

To avoid the communication impact mentioned in COMMA, we set up a dedicated network link (i.e., 1 Gb) for migration traffic. The network round-trip-times (RTTs) between source machines and destination machines are within 1 millisecond. With this setup, the overhead for VMs communicating across the working network is negligible. Besides, we adopt *pre-copy* VM migration technique and only migrate the memory and CPU states by adopting shared storages between source and destination machines.

C. Multi-VM Migration Characterization

We first choose RUBiS, a well-known benchmark for evaluating web system which simulates an online bookstore. We adopt PHP version RUBiS consisting of a web server and a database server. Correspondingly, the three basic migration strategies are: (1) migrating the web server first; (2) migrating the database server first; (3) migrating them simultaneously. We initiate three different loads: 30 clients, 60 clients and 120 clients to represent light, medium and high loads of the web server. Sufficient resources (e.g., CPU, memory and disk) are assigned to both VMs to ensure that neither of these VMs is overloaded by the clients.

Figure 2 shows the CDFs of the average latency of three loads during migration separately. We observe very consistent results from these three migration strategies: *migrating the web server first* always brings the best performance (the lowest latency) during migration. While the strategy *migrating both VMs simultaneously* always leads to the worst performance (the highest latency). More specifically, in Figure 2, with the light loads (30 clients) both sequential strategies, either migrating web or database first, show the same result; with the medium loads (60 clients) migrating web server first outperforms the other sequential scheme with lower latency. Notably, the performance gap between two sequential schemes widens as the workload goes up.

Further, we characterize the behaviors of RUBiS. The main findings are: (1) the web server is low-stressed consuming relatively less resource –10 ∼30% CPU utilization and a small memory footprint; while (2) the database server is relatively highly-loaded – 30 ∼ 80% CPU utilization and a relatively large memory footprint; (3) thus, the database server suffers from 1.7 times of migration duration as the web server in both sequential cases (two VMs with both 2 GB memory).

Curious about whether such outcome commonly exists, we developed a 2-tier application that simulates an event calendar (by adapting Olio code [17]) to repeat the same

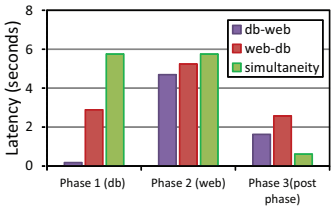


Figure 3. Latency breakdowns of three migration strategies.

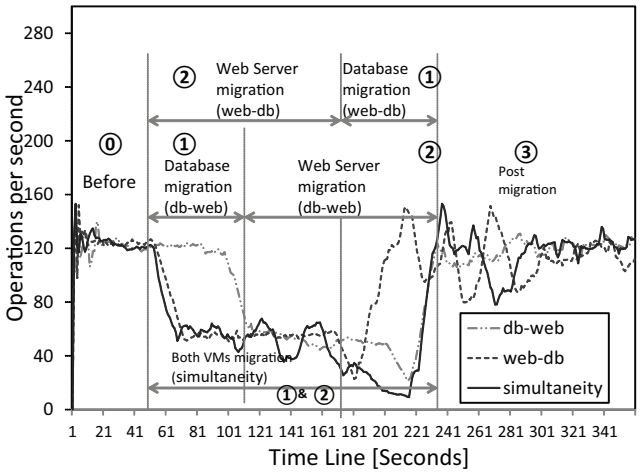


Figure 4. Performance breakdowns of three migration strategies.

migration experiment. Different from RUBiS, the web server of this calendar-based application is relatively highly-loaded due to many dynamic contents while the database server has a smaller amount of load. We still observe the consistent trends for three migration schemes. Similarly, migrating VMs simultaneously still leads to the worst performance. Differently, migrating the database server first appears to be the best scheme. The performance gap between two sequential strategies also increases as the load goes up (detailed data is presented in [18]).

III. Root Cause Analysis

As observed in Section II, different migration strategies impact the performance of multi-tier applications and the performance gap increases as application workload goes up. Then, what is the root cause? COMMA suggested that high latency network path between different data centers could be the culprit. However, in our local data center environment, by separating application and migration traffic, the application network is never congested (within 1 millisecond), suggesting that network latency is not a main factor. We will show empirically in this section that the inter-dependence between different components in a multi-tier application causes this problem. We further analytically prove it in [18].

To explore the root cause of the performance gap observed above, Figure 4 illustrates the application-level throughput in terms of operations per second over the migration time (sampled from the "event calendar" web application in

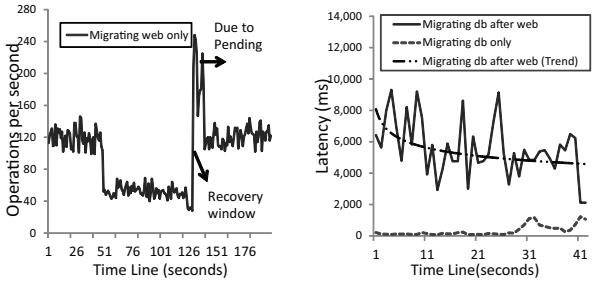


Figure 5. (A) Migrate web alone; (B) Migrate db alone vs. migrate db after web.

Section II). Further, in Figure 3, we break down the corresponding average latency. As illustrated in Figure 4, there are 4 phases during each migration case: **phase 1** is the migration duration of the database server; **phase 2** is the migration duration of the web server[1]; while **phase 0** and **phase 3** are phases before and after the migration. For the parallel migration, phase 1 and phase 2 fully overlap.

A. Sequential migration

Let us first compare two sequential migration schemes, – Scheme web-db (migrating the web server first) and Scheme db-web (migrating the database server first). In Figure 3, during phase 1, Scheme web-db shows 16 times the latency of Scheme db-web. During phase 2, Scheme web-db suffers almost the same latency as Scheme db-web. While during phase 3, Scheme web-db shows 1.58 times the latency of Scheme db-web. [2] Notably there is a big throughput fluctuation in phase 1 of Scheme web-db in Figure 4 illustrating unstable working status of web services.

The decomposed results indicate that the main performance difference between two sequential migration schemes lies in phase 1 – the migration phase of the database server. Notice that in Figure 3, there is little performance degradation or latency explosion during phase 1 of Scheme db-web. On the contrary, latency explosion happens during phase 1 of Scheme web-db. Then, why does Scheme web-db lead to higher latency during phase 1?

To answer this question, we conduct another experiment by migrating the web server alone and examine the workload's dynamics. Figure 5(a) shows the throughput before, during and after migration of the web server. Surprisingly, when the migration of web server completes, there is a *spike* in throughput lasting for 10 seconds. After this spike, the throughput gets back to normal (i.e., before the migration). The results first assert that network latency is not sufficient to cause the performance degradation issue described in COMMA, as there is neither throughput drop nor latency increase, even when the web server and the database server communicate across the network path. Note that after migration, the web server resides on the destination host while the database server still on the source host.

On the other hand, much of the blame of latency explosion during phase 1 of Scheme web-db should rest upon the spike. The spike is twice as the normal throughput, definitely putting more stresses on the web server. This spike also introduces more loads for other component(s), such as the database server, because of interactions between application tiers. We can imagine if the database server migrates right after web server, its performance would be impaired due to the spike. So, it's imperative to know what causes the spike.

Through reviewing the main steps of the pre-copy migration technique [3], we could infer such a spike in throughput stems from two sources. First, during the iterative pre-copy period, an increasing number of requests are pending [4] at the web server side, as the request processing rate decreases while the request arrival rate keeps fixed. Second, during the stop-and-copy period, the incoming requests continuously become pending, as services are interrupted. Note that, thousands of requests could be pending and waiting to be processed during the 4∼5 second downtime. Hence, these two sources of pending requests result in the spike in throughput when the web server resumes after migration.

After inferring the causes of the spike in throughput, we depict the latency comparisons in Figure 5(b) to verify that the spike does impact the average latency during migration using two scenarios: (1) migrating the database server alone and (2) migrating the database server right after the web server, which is Scheme web-db. When the database server is migrated alone, little latency explosion occurs, with only a small latency increase at the end of the migration because of stop-and-copy. It implies that only a very small amount of pending requests exist after migration of the database server. It also exmplains why we do not observe higher throughput degradation during the migration of the database server (phase 1) in Scheme db-web than that in Scheme web-db in Figure 4. However, in the scenario that the database server is migrated right after the web server, the application-level latency remains high. Such latency tends to decrease as pending requests becomes fewer, supporting the fact that the high latency is caused by the pending requests.

So far we have figured out the root cause of the performance gap between the two sequential schemes: **it is because of the pending requests from the preceding VM that has just been migrated, negatively impacting the application-level performance of the next VM to be migrated**.

[1]Please note that the phase numbers do not necessarily reflect the temporal ordering of the phases. In particular, for the web-db scheme, the temporal ordering of phases is 0, 2, 1, 3.

[2]For phase 3, we only measure a short period (e.g., 10 seconds), as the average latency quickly bounces back to the normal level as phase 0.

[3]Four phases of pre-copy migration are initialization, iterative pre-copy, stop-and-copy and activation [19].

[4]An application server typically maintains multiple request queues at the front end. Requests would *wait*(or be pending) for a while before being processed, when the application server is busy.

B. Parallel migration

Due to space constraint, we briefly investigate the parallel scheme. Why is the performance of this parallel scheme always the worst in our measurements?

Note that during parallel migration, the overall performance degradation is decided by both VMs, since both migrating VMs suffer the performance degradation simultaneously. Figure 3 proves this – the parallel scheme suffers high latency during the whole migration - phase 1 and phase 2. In contrast, the sequential schemes only suffer high latency during the migration of the web server – phase 2.

Moreover, in our experimental settings, two VMs are placed on the same source machine and migrate to the same destination machine. The migration processes are observed to be CPU-intensive and usually consume more than 1 vCPU for each VM's migration. Therefore, physical resources become more competitive under the parallel scheme, hence causing more application performance degradation. We can imagine that the resource competition would become more severe as the number of parallel migrated VMs increases in the same physical host.

As a result of these two influential factors above, the parallel scheme potentially results in higher performance degradation than the sequential schemes. We further prove this analytically using queuing theory in [18]. Note that our conclusion is based on the assumption that total migration bandwidth is *dedicated-but-fixed*, following real-world VM migration setup (e.g., vMotion [2]). The unlimited-migration-bandwidth scenario is left to future work.

IV. Optimal Multi-VM Migration Scheduling

Based on the root cause analysis, we have the observations for two-tier application migration: (1) if there is no performance impact from pending requests, two sequential strategies should be equivalent and result in the same performance degradation, while the parallel migration strategy results in worse performance; (2) if there is performance impact from pending requests, by calculating the impact of pending requests (analytically defined in [18]), we can compute the optimal migration strategy. However, such decision-making is impractical to implement, as it requires accurate measurement of VM migration time and workloads.

A. vHaul Design and Implementation

Instead, we propose vHaul, a multi-VM migration coordination system, to approximate the optimal solution. vHaul takes an application semantics-agnostic approach to avoid per-application instrumentation, which is often infeasible in public application-hosting clouds.

Specifically, vHaul covers two typical migration scenarios considering different migration requirements. (i) If we wish to achieve the minimum performance impact on applications while performing migration *without* any constraint on end-to-end migration time, VMs can be migrated one by one, separated by a long non-migration interval between two consecutive VM migrations. The underlying rationale is to mitigate the performance impact by pending requests, as pending requests are supposed to be processed during the non-migration interval. This simple method benefits directly from observation (1) above. (ii) If we need to complete the end-to-end migration without any delay, the migration strategy requires shortest migration time while maintaining an acceptable service downtime for the least performance loss. To this end, vHaul devises a heuristic multi-VM migration scheduling algorithm in Algorithm 1 (for simplicity, we only show the pseudo code). vHaul assumes there is a dedicated migration link shared by all VMs for traffic non-interference and security [20].

In Algorithm 1, given a set of VMs to be migrated, vHaul first categorizes them according to their logical relationship – VMs belonging to the same application are grouped together. According to Section III-B, it's practically impossible that the parallel strategy could outperform the sequential strategies in terms of performance with the dedicated-but-fixed migration bandwidth. In addition, considering the high resource contention caused by parallel migration processes, vHaul prefers to migrate VMs in a specific sequence.

Next, in order to determine the sequential migration order, vHaul sorts VMs in the same group by the product of resource utilization (U_{vm}^{curr}) and migration time t, $U_{vm}^{curr} \cdot t$. vHaul adopts $U_{vm}^{curr} \cdot t$ to approximate the impact of both pending requests and migration time. Large $U_{vm}^{curr} \cdot t$ means a VM potentially impacts more on the next migrated VM, and vice versa. The product above is a heuristic metric reflecting pending requests' impact, with both factors equally important. We will show in Section V, any single factor cannot decide the migration order; whereas the product selects the optimal order. vHaul prefers to migrate VMs with smaller $U_{vm}^{curr} \cdot t$ ahead of VMs with larger $U_{vm}^{curr} \cdot t$ for the purpose of reducing the impact by pending requests.

In this paper, we mainly study the performance impact on one multi-tier application using vHaul. Hence we only focus on the migration order of VMs belonging to the same application.

B. Parameterization

To realize Algorithm 1, vHaul needs to group related VMs, define U_{vm}^{curr} and estimate migration time t.

First, we developed a traffic monitor inside Xen's driver domain to construct the traffic matrix between VMs, because all VMs' IO traffic have to go through the driver domain [21]. Using this traffic matrix, vHaul is able to group VMs accordingly – VMs with communication traffic are treated within a multi-tier group.

Next, in order to represent the resource utilization of a VM, vHaul chooses the following performance metrics: CPU, memory and IO resource. CPU, memory, IO utilization calculation follows existing methodology/tools. For CPU,

vHaul uses Xentop's command version [22] to collect average vCPU utilization for each VM (normalized between 0%-100%). For memory, vHaul injects a script into each domU, then collects and calculates memory utilization (0%-100%) through Xenstore [23]. For IO, vHaul uses IOSTAT tool. Overall resource utilization is a weighted sum of the three. Weights are determined by their impacts on VM migration: CPU and memory activities incur higher impacts than IO. Hence vHaul assigns lower weight to IO and higher weights to CPU and memory utilizations. The weights are empirically set to 4(CPU):4(memory):2(diskIO). We use the same ratio for all applications in Section V for evaluation.

Finally, to determine the migration time for each VM, we employed the "AVG Simulation Model" in [16]. The dirty page rates are measured and reported from the hypervisor, while the migration network bandwidth is known in advance.

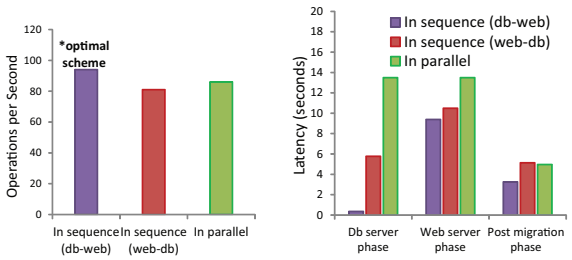
Algorithm 1 A heuristic multi-tier migration algorithm.

Require:
VMs with communication traffic belong to a multi-tier application;
Assign unique ID for each multi-tier application;
Ensure:
Given set of VMs C_{vm} to be migrated;
/* Find out VMs for each multi-tier application*/
for each vm in C_{vm} **do**
id = $get\_application\_id(vm)$;
$G[id].append(vm)$;
end for
/* Calculate vm migration order within a multi-tier application*/
for each g in G **do**
for each vm in g **do**
$vm.migration\_order$ =
$(vm.U\_curr \cdot vm.migration\_time)$
end for
$sort\_vm\_by\_migration\_order$(g);
end for
Return G;

V. Evaluation

In this section, we first evaluate the effectiveness of vHaul by choosing simple client-server architecture applications running within two VMs. Next, to evaluate more complex multi-tier application migration scenarios, we use Apache Olio, a web 2.0 benchmark [17], with four VMs.

Experimental setup Our testbed consists of servers with quad-core 3.2GHz Intel Xeon CPUs and 16GB RAM. They are connected via two separate Gigabit Ethernets. One network is for application production traffic while the other is for VM migration traffic. All VMs share the same 1 Gbps migration bandwidth. By doing so, we purely focus on the multi-tier dependency issue. These physical servers run Xen 4.1.2 as hypervisor and Linux 3.2 in both domain0 and VMs. For each VM, we assign reasonable configurations with enough vCPU number, memory size and disk capacity to ensure there is no performance bottlenecks when no VM is being migrated.



(a) Throughput (b) Latency

Figure 6. Performance evaluation in highly-loaded web case.

A. vHaul 2-tier Evaluation

First, we evaluate vHaul using a similar setup as the "event calendar" web application in Section II, but with different configurations: (1) the web server gets relatively highly-loaded and (2) the database server gets relatively highly-loaded. For both scenarios, we equally assign 2.5 GB memory to the web server and 1 GB memory to the database server.

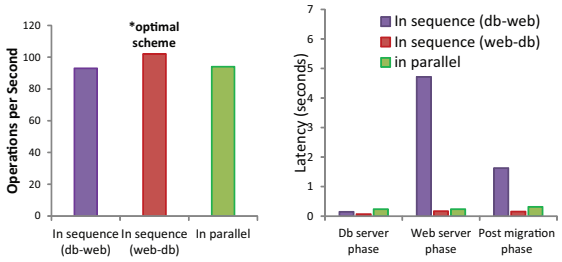
Highly-loaded web server In most two-tier web applications, the web server usually exercises the business logic, hence can easily get highly-loaded. We simulates this scenario by running 600 concurrent users, each sending requests at a speed of one request per five seconds. Once the benchmark starts running, the overall resource utiliazaiton of the web server becomes higher than that of the database server. Hence, through counting the resource utilization and estimating the migration time, vHaul computes that migrating the database server first leads to the *optimal* scheme in this scenario.

Figure 6(a) shows the average throughput during 5 minutes (including all migration phases). The results indicate the *optimal* scheme, Scheme db-web, leads to the best throughput (with the highest number) among three schemes. Further, Figure 6(b) depicts Scheme db-web also results in the lowest average latency in all phases, whereas Parallel scheme results in the highest latency in most phases. [5]

In addition, Scheme web-db results in longer total migration time (not shown in the Figure) for the database server than Scheme db-web. It is because, for Scheme web-db, during migration of the database server, pending requests from the web server make the database server busier. As a result of the corresponding higher dirty page rate, Scheme web-db incurs longer time to migrate the database server than Scheme db-web. On the other hand, Parallel scheme leads to the longest total migration time for both VMs.

Highly-loaded database server To model more complex two-tier applications, we deploy an OLTP workload [24] in the database server VM to simulate the typical online transaction processing scenario, widely used in the SQL Server database [25]. We run two workloads – one is the

[5]We display the latency in the post migration phase just for 10 seconds, as the average latency quickly bounces back to the normal level.



(a) Throughput (b) Latency

Figure 7. Performance evaluation in highly-loaded db case.

web service workload and the other is the OLTP workload – within the same two-tier virtual platform. For the OLTP workload, 400 requests per second are sent from the clients to the database server. For the web service workload, 500 concurrent users are simulated to visit the web server. Due to the high OLTP workload, the database becomes heavily loaded in this scenario.

In Figure 7(a), the average throughput during 5 minutes indicates that the *optimal* scheme – Scheme web-db, calculated by vHaul– achieves the best performance among the three. Specifically, Scheme web-db leads to the lowest request processing latency in Figure 7(b) as well as shortest migration time. Notably, in Figure 7(b) during the web server migration phase, the latency of the sub-optimal scheme, Scheme db-web, is much longer than other schemes.

It is worth mentioning that, we observe the web server takes longer time to migrate than the database server, mainly because of the larger memory assigned (i.e., 2.5 GB for the web server and 1 GB for the database server). However, both CPU and memory utilizations of the database server are much higher than the web server. Thus, the product of resource utilization and migration time of the database server is higher than that of the web server. It indicates that the product of resource utilization is a suitable metric to approximate the impact of pending requests.

	In sequence (db-file-cache-web)*	In sequence (web-db-cache-file)	In parallel
HomePage	1.42	3.78	5.63
Login	0.48	2.20	3.32
TagSearch	4.21	6.33	9.36
EventDetail	1.30	1.91	3.34
PersonDetail	8.21	10.11	13.30
AddPerson	2.90	3.34	6.67
AddEvent	17.71	26.02	32.04
Geomean Latency	2.86	5.09	7.74
Ratio	-	**1.78x**	**2.70x**

Table I
LATENCY BREAKDOWN FOR EACH REQUEST OPERATION FROM OLIO EXPERIMENTS.

B. vHaul Multi-tier Evaluation

Next, we use Apache Olio, a web 2.0 benchmark, to evaluate vHaul. The Apache Olio benchmark consists of four components: (1) a web server to process user requests, (2) a MySQL database server to store user profiles and event information, (3) an NFS server to store images and documents and (4) a memory cache server to cache recent accessed contents for better performance. The PHP version of this benchmark is adopted.

We run 650 concurrent users, each sending requests at a speed of one request per every five seconds. We allocate enough resources for each VM to ensure there is no performance bottleneck during the non-migration time. Particularly, we assign 2.5 GB memory to the web server, 1 GB memory to the database server and 0.5 GB memory to the file server and cache server respectively. The peak CPU utilization of the web server is about 70% [26], which is very close to the CPU load in cloud environments. The rest of the VMs have much lower CPU and memory utilization (10% - 40%). For each request, the web server first needs to check whether a response can be retrieved directly from the cache server. If the content is not cached, the web server requires contacting the database server and the NFS server to compose the complete content and reply to the client. All VMs migrate through the dedicated 1 Gbps migration link.

Initially, all VMs are running on the same source physical node. After certain time, a migration command is issued to vHaul to conduct the group migration. Then, vHaul computes the *optimal* migration scheme based on Algorithm 1 and coordinates the multi-VM migration. The *optimal* migration order calculated by our framework is marked with an asterisk in Figure 8 and Figure 9: *db-file-cache-web.* We also pick up the worst sequential migration case and the parallel case for comparison.

The *optimal* scheme provided by vHaul outperforms both the worst sequential scheme and the parallel scheme by 37% and 43% respectively as shown in Figure 8. Looking into the detailed performance data, we find that for the worst scheme, the performance is negatively impacted not only by the highly-loaded web server but also by the file server. The reason is that during migration, the file server could become highly-loaded as a consequence of the pending requests mentioned above. Sometimes, the file server becomes even *over-loaded* and just hangs without any responses for several seconds. On the other hand, the performance of the parallel scheme is mainly negatively impacted by the highly-loaded web server as well as high resource contention. For space constraint, we only present the “worst” and “optimal” (by Algorithm 1) migration orders. The results from all other orders are between those of “worst” and “optimal”.

Table I shows the latency breakdown for each Olio operation. The geometric mean is applied to calculate the average latency of seven Olio operations. The *optimal* scheme suggested by vHaul results in the lowest latency. Specifically, the worst sequential scheme shows 1.78× the latency; and the parallel scheme shows 2.7× the latency – of the *optimal* scheme. In addition, the *optimal* scheme also leads to a shorter migration time (in Figure 9) than other schemes.

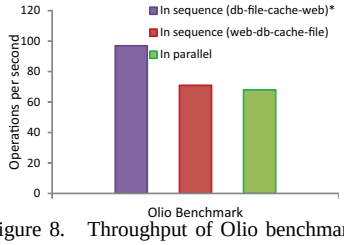


Figure 8. Throughput of Olio benchmark.

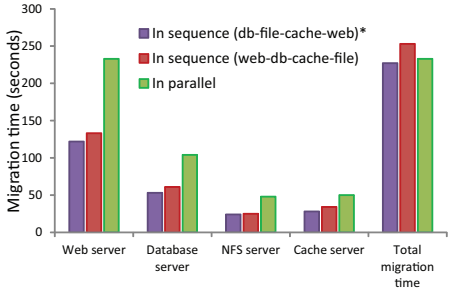


Figure 9. Migration time for Olio benchmark.

VI. Related Work

Ye et al. [27] evaluated live migration strategy of multiple VMs from experimental perspective. Deshpande et al. [28] and Al-Kiswany et al. [29] optimized concurrent live migration of multi-VM using de-duplication approach. LIME [19] leveraged Software Define Networking advances to bring up a transparent solution to migrate VMs of a tenant. However, none of existing work focused on the inter-dependence relationship between multi-tier VMs and consequential performance impact. COMMA [11] and Clique [30] tackled the multi-VM migration problem in geographically distributed clouds. COMMA identified that the performance of a multi-tier application can severely degrade if its dependent components become split across a high latency network path. Clique further optimized the group migration method by partitioning a large group of VMs into subgroups based on the traffic affinities among VMs. In contrast, vHaul studies multi-VM migration problem within a different environment – namely within a local data center with dedicated migration link of low latency.

VII. Conclusion

In this paper, we demonstrate that different migration strategies result in distinct performance impacts on a multi-tier application in dedicated data centers. Using controlled experiments and queuing theory, we show the inter-dependence between different tiers of a multi-tier application causes this problem. Then we present a system, vHaul, which computes the optimal multi-VM migration scheme and improves the performance of multi-tier application during migration.

VIII. Acknowledgements

This work was supported in part by NSF under Award 1219004. Any opinions, findings, and conclusions in this paper are those of the authors and do not necessarily reflect the views of NSF.

References

[1] Xenmotion. http://blogs.citrix.com/2012/08/24/storage_xenmotion/.
[2] vmotion. http://www.vmware.com/products/vsphere/features/vmotion.
[3] W. Voorsluys, J. Broberg, and S. Venugopal, "Cost of virtual machine live migration in clouds: A performance evaluation."
[4] D. Breitgand, G. Kutiel, and D. Raz, "Cost-aware live migration of services in the cloud," in *SYSTOR '10*.
[5] H. Liu, C.-Z. Xu, H. Jin, J. Gong, and X. Liao, "Performance and energy modeling for live migration of virtual machines," in *ACM HPDC*, 2011.
[6] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *USENIX NSDI*, 2005.
[7] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan, "Live virtual machine migration with adaptive, memory compression." in *IEEE CLUSTER*, 2009.
[8] X. Zhang, Z. Huo, J. Ma, and D. Meng, "Exploiting data deduplication to accelerate live virtual machine migration." in *IEEE CLUSTER*, 2010.
[9] Amazon. aws reference architecture. http://aws.amazon.com/architecture/.
[10] The new business continuity: Moving applications across data centers without interruption. http://www.brocade.com/downloads/documents/solution_briefs/.
[11] J. Zheng, T. S. E. Ng, K. Sripanidkulchai, and Z. Liu, "Comma: Coordinating the migration of multi-tier applications," ser. VEE '14.
[12] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, 2003.
[13] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum, "Optimizing the migration of virtual computers," *SIGOPS Oper. Syst. Rev.*
[14] E. Zayas, "Attacking the process migration bottleneck," in *SOSP'87*.
[15] M. R. Hines, U. Deshpande, and K. Gopalan, "Post-copy live migration of virtual machines," *ACM SIGOPS operating systems review*, 2009.
[16] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper, "Predicting the performance of virtual machine migration," in *IEEE MASCOTS*, 2010.
[17] Apache olio. http://incubator.apache.org/projects/olio.html.
[18] vhaul: Towards optimal scheduling of live multi-vm migration for multi-tier applications. Department of Computer Science Technical Report (TR15-001), Purdue University. http://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=2775\&context=cstech.
[19] E. Keller, S. Ghorbani, M. Caesar, and J. Rexford, "Live migration of an entire network (and its hosts)," in *ACM HotNets-XI*, 2012.
[20] vsphere vmotion networking requirements. http://pubs.vmware.com/vsphere-51/index.jsp?topic=\%2Fcom.vmware.vsphere.vcenterhost.doc\%2FGUID-3B41119A-1276-404B-8BFB-A32409052449.html.
[21] Para-virtualization. http://wiki.xenproject.org/wiki/Paravirtualization_\%28PV\%29.
[22] Xentop. http://wiki.xen.org/wiki/Xentop\%281\%29.
[23] Xenstore. http://wiki.xen.org/wiki/XenStore.
[24] Sysbench. http://www.storagereview.com/sysbench_oltp_benchmark.
[25] Sql server on vmware. http://www.vmware.com/files/pdf/solutions/SQL_Server_on_VMware-Best_Practices_Guide.pdf.
[26] Data centers utilization. http://www.thegreengrid.org/en/Global/Content/Tools/~/media/Tools/DCMM%20-%20Compute.ashx.
[27] K. Ye, X. Jiang, D. Huang, J. Chen, and B. Wang, "Live migration of multiple virtual machines with resource reservation in cloud computing environments," in *IEEE CLOUD*, 2011.
[28] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, "Vmflock: virtual machine co-migration for the cloud," in *HPDC*, 2011.
[29] U. Deshpande, B. Schlinker, E. Adler, and K. Gopalan, "Gang migration of virtual machines using cluster-wide deduplication," in *IEEE/ACM CCGrid*, 2013.
[30] T. L. S. M. K. T. X. He, "Clique migration: Affinity grouping of virtual machines for inter-cloud live migration," *Networking, Architecture, and Storage (NAS), 2014 9th IEEE International Conference on*.