2015

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

Hui Lu

Cong Xu

Cheng cheng

Ramana Kompella

Dongyan Xu

Report Number:

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

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 52% in application throughput. Moreover, the optimal schedule significantly reduces service latency during migration by up to 70%.

1. Introduction

Live VM migration techniques (e.g., XenMotion [10] and vMotion [8]) 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 [15, 23, 26]. While most state-of-the-art live VM migration techniques [16, 19, 20, 24, 29] 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* [1]. Such inter-dependent VMs are subjected to migration as a group within a data center or across various data centers [4]. Recently, COMMA [30] 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 1ms). 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 conduct 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 strate-

gies 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 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 52%. 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 2 and 3). (2) We propose vHaul as a simple approach to mitigating such impact that can be deployed for a range of cloud application scenarios (Section 4). (3) We have implemented a prototype of vHaul based on Xen [14] to coordinate multi-VM migration and demonstrated improvement in application-level performance (Section 5).

2. Investigation and Observations

In this section, we discuss the performance degradation caused by single and multi-VM migration. Then we motivate the new problem by demonstrating the impact of various VM migration strategies on the performance of multi-tier applications during migration.

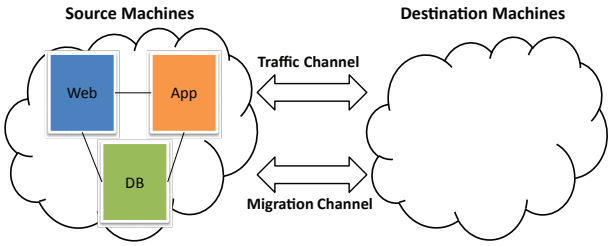


Figure 1: Multi-tier application migration scenario

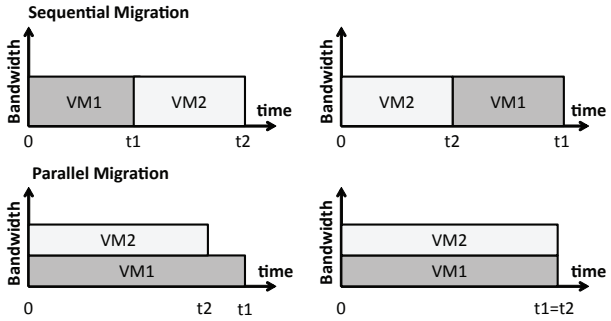


Figure 2: Strategies to migrate a multi-tier application

2.1 Single VM Live Migration

Live migration is a key technology driving virtualized data center which enables moving a running VM from one physical host to another for better load balancing or hardware maintenance with minimum service interruption. During a VM's migration, the source and destination hosts both need to run a per-VM migration process in the background to store and transfer the active memory, OS execution states and virtual I/O devices configurations, which consumes non-negligible system resources such as CPU cycles, memory and network bandwidth. We observed that the performance degradation of a VM's workload during migration just can be caused by the resource contention between the VM's running workload and the migration process.

Hence, it is important to shorten the duration of VM migration to mitigate its performance impact on user level applications. Several existing live migration techniques such as stop-and-copy[25], on-demand[28], pre-copy [16] and post-copy [19] have been applied to real world data centers to boost migration speed and reduce service downtime. For almost all of them, two parameters – total migration time and service downtime – are of particular concern. Particularly, we observe that two influential factors affect these two parameters: link bandwidth available for migration and the page dirty rate[12] in host. Clearly, the higher the bandwidth available, the shorter the total migration time and service downtime, because the involved active memory and VM running state can be transferred to the destination host faster. The page dirty rate affects the amount of memory pages

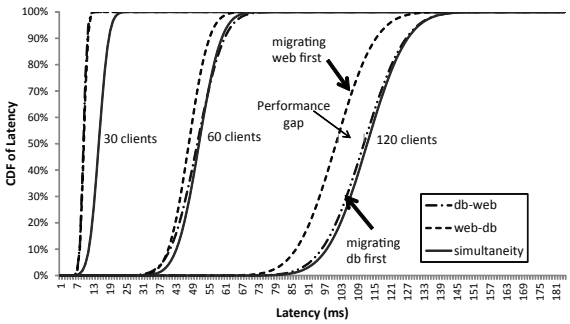
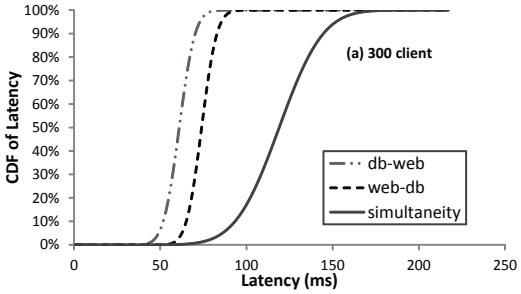
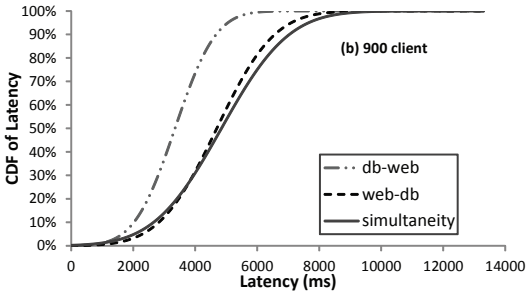


Figure 3: CDF of latency for various loads (30 clients, 60 clients and 120 clients) in RUBiS



(a) 300 clients



(b) 900 clients

Figure 4: CDF of request processing latency under various loads (300 clients and 900 clients) for the calendar-based application

transferred during migration. As a consequence, a higher page dirty rate translates into longer VM migration time and service downtime.

2.2 Multi-VM Live Migration

So far, we only discuss individual VM migration scenario. However, 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. Figure 1 shows an example of migrating a 3-tier e-commerce application from the source to the destination.

Existing solutions described in Section 2.1 work well for migrating an individual VM. However, 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 2. (1) *Sequential migration*, VMs are migrated one by one. In the two-VM case shown in Figure 2, 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 [30].

Live migration in the single VM case does negatively impact on the performance of the VM. Intuitively, the two performance influential factors, migration link bandwidth and page dirty rate, are also critical in the multi-VM scenario. In addition, COMMA 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. Such performance degradation would become even worse if the working traffic and migration traffic share the same network link. To avoid this communication problem, COMMA strived to migrate VMs of a multi-tier application simultaneously – all VMs are managed to start and end the migration at the same time.

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 [4]). Considering the latency of modern network fabrics is typically sub-millisecond, the communication problem above should not be significant, if the multi-VM migration happens *within* the same data center. So the migration link bandwidth is not the main influential factor any more in modern virtualized date center. We will show this in our measurement study using a dedicated data center environment with designated migration links.

2.3 Measurement Methodology

To identify the factors impacting multi-VM migration performance, let us focus on a concrete example. 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 clients side. Since a web request is typically composed of various sub-types (e.g., "preview", "purchase", etc.), we employ 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 during the migration. Note that higher average latency means worse application performance.

To avoid the communication impact mentioned in Section 2.2, we set up a dedicated network link (i.e., 1Gb) for VM migration traffic. The network round-trip-times (RTTs) between source machines and destination machines are within 1 millisecond (in Amazon EC2, the latency among zones is in the same millisecond range). With this setup, the overhead for VMs communicating across the working network is negligible. Besides, we adopt *pre-copy* VM migration technique in our experiments and only migrate the memory and CPU states by adopting shared storages between source and destination machines.

2.4 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 3 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 3, 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 in terms of lower latency. Notably, the performance gap between two sequential schemes widens as the workload goes up.

To further explore the possible factors affecting the multi-tier application performance during migration, we characterize the behaviors of RUBiS. The main findings are: (1) the web server, as the front-end of the client-server application, is low-stressed consuming relatively less resource (10 $\sim$30% CPU utilization and a small memory footprint); while (2) the database server, as the back-end of this 2-tier application, is relatively highly-loaded (30 $\sim$ 80% CPU utilization and a relatively large memory footprint); (3) thus, the database server suffers from 1.7 times of duration as the web server to complete the migration in both sequential cases (two VMs are configured with the same memory size of 2 GB).

We are curious about whether such outcome commonly exists. So we choose another 2-tier application that simulates an event calendar to repeat the same migration experiment. Different from RUBiS, the web server of this calendar-based application is relatively highly-loaded due to many dynamic contents. The database server has a smaller amount of load. We initiate two different loads: 300 clients and 900 clients to represent light and high loads.

Figure 4 shows the CDFs of the average response time during migration for this calendar-based application. 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.

What causes the different results for these two 2-tier applications? We analyze the the main characteristics of the calendar-based application: (1) the web server is highly-loaded (30 $\sim$80% CPU utilization with a large memory footprint); while (2) the database server is relatively lightly loaded (10 $\sim$ 30% CPU utilization with a small memory footprint); (3) the web server suffers from almost twice the duration as the database server to complete the whole migration in sequential migration cases.

Consequently, we can draw some conclusions based on the observations above. (1) The sequential migration order does impact the performance of a multi-tier application. (2) Different applications have different migration preferences due to their varying component workloads. (3) The performance impact of different migration schemes becomes increasingly significant as workload increases. Since we eliminate the influence of the communication impact by separating VM migration and application traffic, the differences in performance impact among the three migration strategies are only determined by the semantic interactions of different components in each multi-tier application.

3. Root Cause Analysis

As observed in Section 2, 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 the 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 millisec-

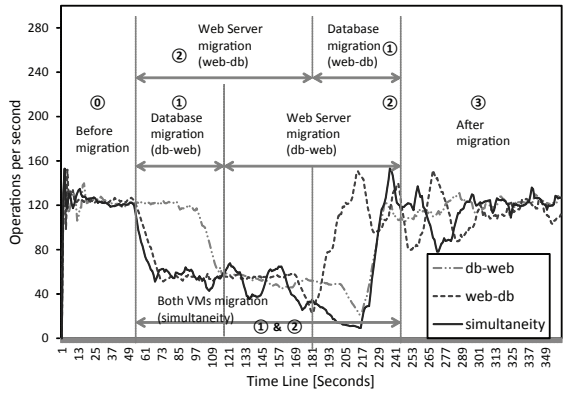


Figure 5: Performance breakdowns of three migration strategies

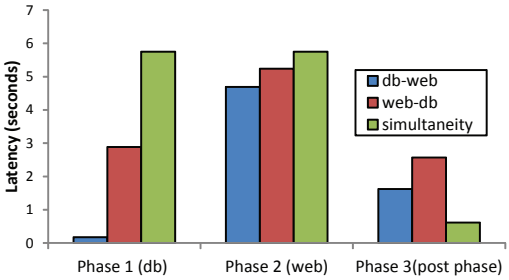


Figure 6: Latency breakdowns of three migration strategies

ond), suggesting that network latency is not a main factor to cause the performance degradation.

We will show both empirically and analytically in this section that the inter-dependence between different components in a multi-tier application causes this problem.

3.1 Pre-copy VM Migration

To fully understand the impact of live migration, we must first present some background on the single VM migration technique, we selected "pre-copy". Pre-copy VM migration combines a bounded iterative push step with a final and typically short stop-and-copy phase. The design leverages the idea of iterative convergence, which involves the following main steps:

(1) Initialization: a destination is selected and the resources are reserved.

(2) Iterative pre-copy: the entire RAM is sent in the first iteration;in the following iterations, only the dirty pages during the previous iteration are transferred to the destination.

(3) Stop-and-copy: the VM is halted for a final transfer round due to conditions such as, less than a minimum number of pages (i.e., 50 pages) are dirtied during last period; a maximum number of iterations (i.e., 29 iterations) have been executed; and more than a number times (3x) the total amount of RAM of a VM has been transferred to the destination.

(4) Activation: resources are re-attached to the VM on and destination and the VM is resumed.

Since the iterative pre-copy processes compete for CPU and memory resources during migration, the performance of the workload running inside the migrating VM will be affected. Further, services will be interrupted for a very short period (i.e., downtime) during the stop-and-copy phase.

3.2 Empirical Explanation

To explore the root cause of the performance gap observed above, we break down the migration procedure into several phases. In this section, we mainly focus on the case of the "event calendar" web application with a workload of 900 clients. For comparison, one representative experimental sample of each migration scheme, discussed in Section 2, is chosen.

Figure 5 illustrates the application-level throughput in terms of operations per second over the migration time. Further, in Figure 6, we break down the corresponding average latency, the same performance metric used in Section 2. As illustrated in Figure 5, 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.

We observe that, during the end-to-end migration process, the application-level throughput becomes much lower than that during the non-migration phases. Especially, during the migration phase of the web server, the throughput is reduced by more than 50%. We also observe the lowest throughput occurs at the end of each VM pre-copy stage because of stop-and-copy. These observations are consistent with the impact of corresponding pre-copy steps introduced in Section 3.1.

Sequential migration: Let's first compare two sequential migration schemes, Scheme web-db (migrating the web server first) and Scheme db-web (migrating the database server first). As shown in Figure 6, 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. 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. Notably, there is a big throughput fluctuation in

[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.

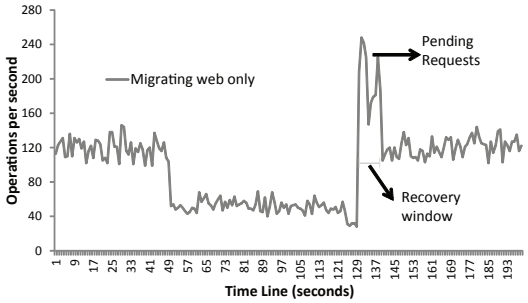


Figure 7: Application-level performance behaviors when migrating web server alone

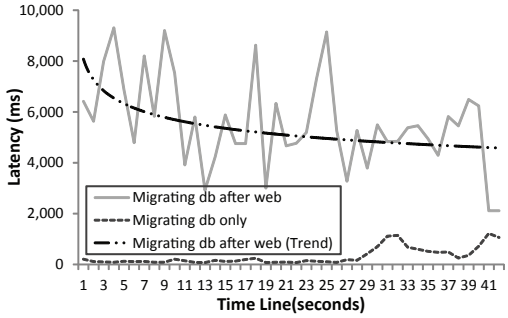


Figure 8: Latency comparisons between two cases: (1) purely migrating database server and (2) migrating database server after web server

phase 1 of Scheme web-db in Figure 5, demonstrating unstable working status of the web services.

The decomposed results indicate that the main performance difference between two sequential migration schemes lies in the migration phase 1 – the migration phase of the database server. Let's now turn back to Figure 4b, we can infer the reason that Scheme db-web has better average latency than Scheme web-db during whole migration arises from the lower latency in the migration phase of the database server. Notice that in Figure 6, 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 7 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, which definitely puts more stresses on the web server. Correspondingly, this spike also introduces more load for other component(s), such as the database server, because of interactions between application tiers. We can imagine that, if the database server migrates right after web server, its performance will probably be impaired in relation to the spike and decreased processing rate (because of the intervention of migration processes). Hence, it's imperative to know what causes the spike.

Recall that the length of the pre-copy and downtime period is mainly dependent on the memory size of a VM, the type of an application (how memory-intensive it is) and the migration bandwidth. Besides, the stop-and-copy policy also plays an important role in deciding the length of downtime. From previous work, we know pre-copy time could be long while downtime should be small (i.e., 2$\sim$ seconds [4]). In our measurements, the web server, which is memory-intensive, usually takes around 80 seconds to complete pre-copy and 4$\sim$5 seconds to transfer the last amount of memory (with the loads of 900 clients). The stop-and-copy condition is usually triggered by reaching to the maximum number of iterations (i.e., 29 in Xen).

Through reviewing the main steps of the pre-copy technique, we could infer such spike in throughput stems from two sources. First, during the iterative pre-copy period, an increasing number of requests are pending [2] 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$\sim$5 second downtime, and hence coupled with the ones accumulating from the period of pre-copy, the pending requests could result in the spike in throughput when the web server resumes after migration.

After inferring the cause of the spike in throughput, we depict the latency comparisons in Figure 8 to verify that the spike does impact the average latency during migration of the database in Scheme web-db 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

[2] In order to improve performance, an application server usually 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. The number of pending requests mainly depends on request arrival rate and application's service time [22].

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. That is the reason 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 shown in Figure 5. 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.

We have now 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. As an example, in our earlier event-calendar application, for Scheme web-db, pending requests during the migration phase of the web server propagate to the migration phase of the database server, resulting in high latency (phase 1 of Figure 6). While for Scheme db-web, a small amount of pending requests during the migration phase of the database server propagate to the migration phase of the web server. Hence, there is no significant impact on performance during the migration phase of the web server for Scheme db-web (phase 2 of Figure 6).

Parallel migration: Let's investigate the parallel scheme – migrating both VMs simultaneously. Why is the performance of this scheme always the worst in our measurements?

Recall that in our experiments, all VMs share the same migration network link. Assume both sequential and parallel schemes transfer the same amount of memory during the entire migration. It is easy to understand the total migration time should be the same for all schemes. In other words, for each VM in the parallel migration, it takes longer time to complete the migration. For example, given two VMs of the same memory size sharing the same migration bandwidth, each of them will (roughly) occupy half of the total bandwidth; thus the migration time doubles for each VM. We can easily confirm this in Figure 5 – the total migration time of the parallel scheme is nearly the same as that of the sequential schemes.

Note that during parallel migration, the overall performance degradation is decided by both VMs, since both migrating VMs suffer the performance degradation simultaneously. Figure 6 proves this – the parallel scheme suffers high latency during the whole migration - phase 1 and phase 2. 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 than sequential schemes, hence causing more application performance degradation. It explains what we have observed in Figure 3, the parallel scheme performs worse than the two sequential schemes, even when the overall application workload is very light (30 clients in RUBiS). We can imagine that the resource competition would become more severe as the number of parallel migrated VMs increases within the same physical host.

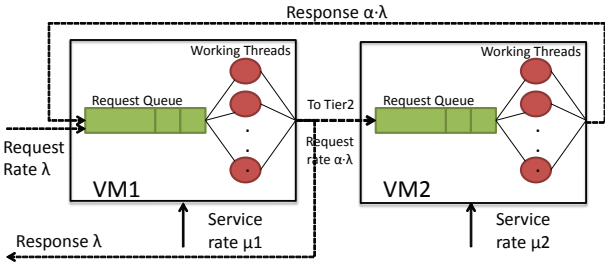


Figure 9: A two-tier queueing model

As a result of these two influential factors – the aggravated performance impact of two VMs and the high resource contention – the parallel scheme potentially results in higher performance degradation than the sequential schemes. We will further prove this analytically in Section 3.3.

So far, we have drawn comparisons between different migration strategies with an event- calendar application. To verify the observations above, we extensively examine other applications, which corroborates our conclusions. For example, in RUBiS, a large amount of pending requests are generated during the migration of the database server (instead of the web server), impairing performance during the migration of the web server in the sequential migration cases. For RUBiS, Scheme web-db outperforms Scheme db-web.

3.3 Analytical Explanation

In this section, we will confirm the conclusions above analytically. To formulate the application-level performance impact of various migration strategies, we present a basic two-tier queueing model in Figure 9. In this model, VM1 (tier 1) is responsible for processing (1) incoming requests from clients with the request arrival rate at λ and (2) the response results from VM2 (tier 2). Each client request to VM1 may trigger zero or more communication with VM2. So, to complete λ requests per second, VM1 needs to send $\alpha \cdot \lambda$ subsequent requests to VM2 and wait for corresponding responses. Usually, each tier consists of one or several request queues and multiple working threads. The performance capacity of each tier is represented by the service processing rate, μ. We assume the model in Figure 9 is a M/M/1 [22] queueing system, which has been proved to be a good approximation for a large number of queueing systems.

According to the queueing theory, given an M/M/1 queueing system, the average time (latency) spent in the

system is determined by:

$$w = \frac{1}{\mu - \lambda} \tag{1}$$

By the equation above, the average latency of the two-tier web application during migration – via different migration schemes can be expressed as:

$$\begin{aligned}
W_{tier1_tier2} &= \{(\underbrace{\frac{1}{\mu_1^{'} - (1+\alpha)\cdot\lambda} + \frac{1}{\mu_2 - \alpha\cdot\lambda}}_{phase2})\cdot t_1 + \\
&\underbrace{(\frac{1}{\mu_1 - (1+\alpha)\cdot(\lambda+\Delta\lambda_1} + \frac{1}{\mu_2^{'} - \alpha\cdot(\lambda+\Delta\lambda_1)})}_{phase1}\cdot t_2\}/(t_1+t_2) \\
W_{tier2_tier1} &= \{(\underbrace{\frac{1}{\mu_1^{'} - (1+\alpha)\cdot\lambda - \Delta\lambda_2} + \frac{1}{\mu_2 - \alpha\cdot\lambda - \Delta\lambda_2}}_{phase2})\cdot t_1 + \\
&\underbrace{(\frac{1}{\mu_1 - (1+\alpha)\cdot\lambda} + \frac{1}{\mu_2^{'} - \alpha\cdot(\lambda)})}_{phase1}\cdot t_2\}/(t_1+t_2) \\
W_{parallel} &= \{(\underbrace{\frac{1}{\mu_1^{'} - (1+\alpha)\cdot\lambda} + \frac{1}{\mu_2^{'} - \alpha\cdot\lambda}}_{phase1\&2})\cdot t_1 + \\
&\underbrace{(\frac{1}{\mu_1^{'} - (1+\alpha)\cdot\lambda} + \frac{1}{\mu_2^{'} - \alpha\cdot\lambda})}_{phase1\&2}\cdot t_2\}/(t_1+t_2)
\end{aligned} \tag{2}$$

The overall latency consists of the latency in both phase 1 and phase 2. Because of migration contentions, the service rate of the web server and the database server during migration reduce to $\mu_1^{'}$ and $\mu_2^{'}$ separately. For sequential schemes, *we transform the pending requests into additional request arrival rate for the next VM to migrate*, as $\Delta\lambda_1$ and $\Delta\lambda_2$. The underlying rationale is that, the pending requests from the previous migration phase are supposed to be processed in the next migration phase. Thus, in addition to the normal request arrival rate λ, $\Delta\lambda_1$ and $\Delta\lambda_2$ are adopted to reflect such impact of pending requests – respectively, $\Delta\lambda_1$ and $\Delta\lambda_2$ are proportional to the number of pending requests. According to Equation (2), larger $\Delta\lambda_1$ or $\Delta\lambda_2$ leads to higher latency. Total migration time is represented as t1 (for the web server) and t2 (for the database server). Thus, the optimization objective for this two-tier model is to determine the minimum overall latency of three schemes:

$$\underset{k}{\operatorname{argmin}}\, W_k \mid k \in \{tier1\_tier2, tier2\_tier1, parallel\} \tag{3}$$

In practice, it's not easy to solve this optimization problem accurately, although we can make some rough comparisons. The question, "which sequential scheme is better in terms of lower overall average latency?" mainly depends on two factors – pending requests and migration time. Equation (2) also implicates that the parallel scheme only beats the sequential schemes when both $\Delta\lambda_1$ and $\Delta\lambda_2$ are significantly high while $\mu_1^{'}$ and $\mu_2^{'}$ are not significant low, which contradicts each other, especially for the real-world applications.

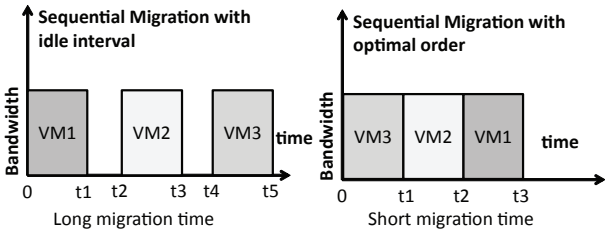


Figure 10: Multi-VM migration senarios

This supports the fact that, in our measurements, we did not observe such a situation in which the parallel scheme outperforms the sequential schemes.

Using the event-calendar application as an example, where $\Delta\lambda_2 \approx 0$ and $\Delta\lambda_1 \gg 0$. We can draw the same conclusions presented in Section 3.2 – Scheme db-web is better than Scheme web-db, as $W_{db\_web} < W_{web\_db}$. Then let us compare Scheme db-web with the parallel scheme under the same example – the latency during parallel migration is higher than both phase 1 and phase 2 of Scheme db-web. So Scheme db-web is better than the parallel scheme.

Though the practical multi-tier queueing model could be more complex than the one presented in Figure 9, this simplified model is useful for performance modeling because it renders a smaller parameter space which is easy to estimate. Through this model, we understand that, (1) the pending requests and migration time are two critical factors for determining the optimal sequential migration order; (2) the parallel migration scheme usually leads to the worst performance under the shared migration link configuration.

4. Optimal Multi-VM Migration Scheduling

Based on the root cause analysis, we have the following observations for two-tier application migration: (i) 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; (ii) if there is performance impact from pending requests, by solving Equation (3), we can obtain the optimal migration strategy.

However, solving Equation (3) is difficult in practice. There are several reasons: (1) Migration time is hard to accurately estimate because of the non-deterministic property of multi-tier applications. (2) A VM's working status is not easy to discern without accessing the running VMs. For example, in a public cloud environment, the cloud provider cannot obtain full control of the customer's VMs. (3) Pending requests are particularly hard to measure, as they are specific to applications.

4.1 vHaul Design and Implementation

Instead of solving Equation (3) directly, we propose vHaul, the multi-VM migration coordination system, to approximate the optimal solution. vHaul mainly covers two typical

migration scenarios considering different migration requirements illustrated in Figure 10.

(a) 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 (i) mentioned at the beginning of Section 4.

(b) 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, we devise a concrete multi-VM migration scheduling algorithm in Algorithm 1. vHaul assumes we have a dedicated migration link shared by all VMs for traffic non-interference and security [9]. For simplicity, we only show the algorithm pseudo code in the next subsection.

In Algorithm 1, given a set of VMs to be migrated, we first categorize them according to their logical relationship – VMs belonging to the same application are grouped together. According to Equation (2), it's practically impossible that the parallel strategy could outperform the sequential strategies in terms of performance. In addition, considering the high resource contention caused by parallel migration processes, vHaul prefers to migrate VMs in a specific sequence.

In order to determine the sequential migration order, we sort VMs in the same group by the product of resource utilization and migration time, $U_{vm}^{curr} \cdot t$. We adopt $U_{vm}^{curr} \cdot t$ to roughly approximate the impact of both pending requests and migration time (proved to be the critical factors in Section 3.3). Large $U_{vm}^{curr} \cdot t$ means a VM potentially impacts more on the next migrated VM, and vice versa. According to Equation (2), 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.

Finally, to decide the migration order of different VM groups (i.e., applications), we assign priority values to these groups/applications in advance, and migrate them by this value. We note that, in this paper, we mainly focus on studying the performance impact on one multi-tier application using vHaul. Hence we focus on the migration order of VMs belonging to the same application.

4.2 Parameterization

To implement Algorithm 1, we need 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 [5]. Using this traffic matrix, we are 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, we choose the following performance metrics: CPU, memory and IO resource. Correspondingly, Xentop[11] and Iostat[3] are employed to collect VM-level CPU and storage IO utilization. While memory utilization of each VM is recorded from the hypervisor. We assign a weight for each metric to determine its relative importance, as we found that CPU and memory utilization play a more important role in deciding the overall resource utilization rather than storage IO.

Finally, to determine the migration time for each VM, we employed the "AVG Simulation Model" in [12]. The dirty page rates are measured and reported from the hypervisor, while the migration network bandwidth can be known in advance.

Algorithm 1 Multi-tier migration scheduling algorithm

Initialization:
VMs with communication traffic belong to the same group;
Assign unique group_ID for each application group;
Assign priority for each application group;
Function:
Given set of VMs C_{vm} to be migrated;
/* Categorize VMs for each application group */
for each vm in C_{vm} **do**
$group\_id$ = $get\_application\_id(vm)$;
$G[group\_id].append(vm)$;
end for
/* Calculate vm migration order in each group*/
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
/* Determine migration order among different groups by priorities */
q = $sort\_group\_by\_priority(g_1, g_2, ..., g_k)$;
/* where $g_j \in G$*/
$Q.append(q_i)$; /* Q, the migration queue */
Return Q;

5. Evaluation

In this section, we first evaluate the effectiveness of our system 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 [2], running within 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 net-

work 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 avoid the performance impact from the in-band migration traffic, hence purely focusing 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.

5.1 vHaul 2-tier Evaluation

First, we evaluate vHaul using a similar setup as the "event calendar" web application in Section 2, 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 (including CPU, memory and disk IO) 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 (note that the resource weights of CPU, memory and disk I/O are set to 4:4:2).

To show the performance benefits, the *optimal* scheme is compared with other schemes. Figure 11 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 12 depicts Scheme db-web also results in the lowest average latency in all three phases, whereas Parallel scheme results in the highest latency in most phases. [3]

In addition, as illustrated in Figure 13, we note that Scheme web-db results in longer total migration time for the database server than the *optimal* scheme, 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.

[3] Note that we display the latency in the post migration phase just for a short interval, i.e., 10 seconds. It is because, in the post migration phase the average latency quickly bounces back to the normal level, as no VM is being migrated or influenced by the earlier migration.

On the other hand, Parallel scheme leads to the longest total migration time for both VMs. But surprisingly, we notice the total migration time (equals to the migration time of the web server, because the web server completed migration last) of Parallel scheme becomes a little bit shorter than two sequential schemes. After investigation, we find it is because Xen's pre-copy algorithm is less efficient for a single VM migration than that for multiple-VM migration, in which one Gbps migration link cannot be fully utilized by a single VM migration process.

Highly-loaded database server: To model more complex two-tier applications, we deploy an OLTP workload [7] in the database server VM to simulate the typical online transaction processing scenario, which is widely used in the SQL Server database [6]. In other words, we run two workloads – one is the 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 14, 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 15 as well as shortest migration time in Figure 16. Notably, in Figure 15 during the web server migration phase, the latency of the sub-optimal scheme, Scheme db-web, is much longer than the other two schemes.

It is worth mentioning that, in Figure 16, we observe the web server takes longer time to migrate than the database server, mainly because of the larger memory (2.5 GB) assigned (with comparison to 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. According to Algorithm 1, vHaul prefers to migrate the database server last.

5.2 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. Partic-

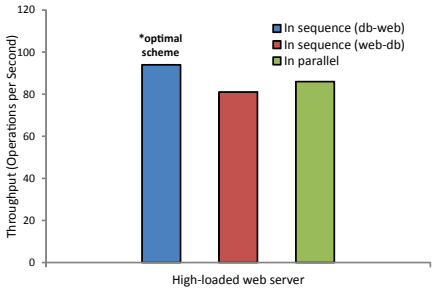


Figure 11: Throughput (Highly-loaded web)

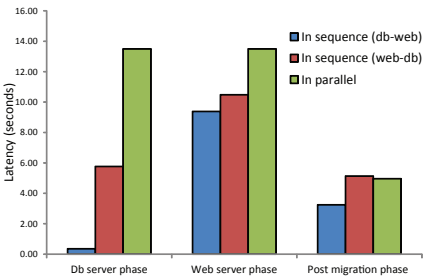


Figure 12: Latency (Highly-loaded web)

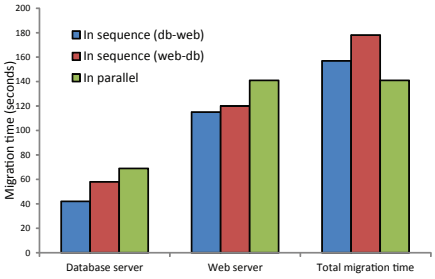


Figure 13: Duration (Highly-loaded web)

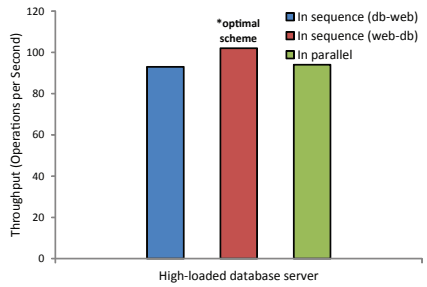


Figure 14: Throughput (Highly-loaded DB)

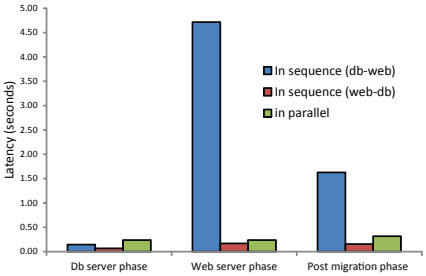


Figure 15: Latency (Highly-loaded DB)

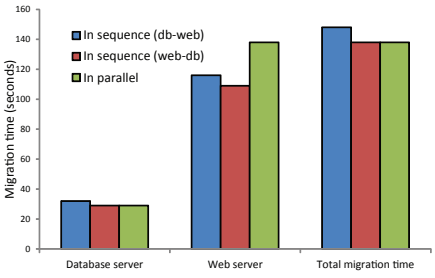


Figure 16: Duration (Highly-loaded DB)

	In sequence (db-file-cache-web)*	In sequence (web-db-cache-file)	In parallel (start simultaneously)
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 1: Latency breakdown for each request operation from Olio experiments

ularly, we assign 2.5 GB memory to the web server, 1 GB memory to the database server and 0.5 GB memory for the file server and cache server respectively. The peak CPU utilization of the web server is about 70%, 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 17 and Figure 18: *db-file-cache-web.* We also execute the worst sequential migration case and the parallel case for comparison. In the parallel case, we only start migrations of all VMs simultaneously but do not control the ending time.

The *optimal* scheme outperforms both the worst sequential scheme and the parallel scheme by 37% and 43% respec-

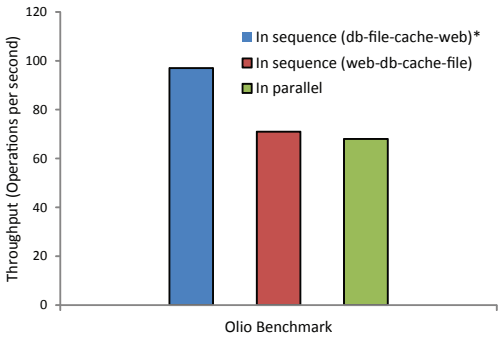


Figure 17: Throughput of Olio benchmark

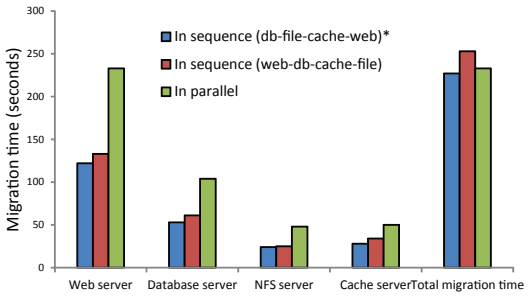


Figure 18: Migration time for Olio benchmark

tively as shown in Figure 17. 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 as discussed earlier. 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.

Table 1 shows the latency breakdown for each Olio operation. The geometric mean is applied to calculate the average latency of the 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 18) than other schemes.

6. Related Work

In this section we discuss related work in the problem space of this paper. We group them into two categories: (1) single VM migration and (2) multi-VM migration.

Single VM migration: Live migration of single virtual machine has been widely studied. Both pre-copy [16, 24] and post-copy [19] are classic single VM migration mechanisms. In [16], Clark et al. investigated live migration of an entire VM and discussed trade-offs between minimizing VM migration time and achieving minimum application/service disruption. The post-copy [19] approach improves VM migration by transferring every page only once to the destination host. Some compression and de-duplication techniques are developed to transfer less amount of memory pages. MECOM [20] adopted memory compression technique,while MDD [29] only transferred differences between dirtied and original pages instead of the entire page. vHaul is orthogonal to these techniques and can leverage their performance optimizations for live multi-VM migration.

Multi-VM migration: Recently, Ye et al. [27] evaluated live migration strategy of multiple virtual machines from experimental perspective and investigated the impact of resource reservation method. Deshpande et al. [13] and Al-Kiswany et al. [17] optimized concurrent live migration of multi-VM using de-duplication approach. LIME [21] leverages 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 relationships between multi-tier VMs and consequential performance impact. COMMA [30] and Clique [18] tackle 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. To handle a large group of VMs, Clique further optimized the group migration method by partitioning a large group of VMs into subgroups based on the traffic affinities among VMs. However, vHaul studies multi-VM migration problem within a different environment – namely within a local data center with dedicated migration link of low latency.

7. Conclusion

In this paper, we demonstrate that different migration strategies result in distinct performance impacts on a multi-tier application in dedicated data centers. Notably we find that such performance impacts are a property of the multi-tier application rather than the network, and that the performance gap becomes increasingly significant as the loads of the application increase. Using controlled experiments and queuing theory, we show the interdependence between different tiers of a multi-tier application causes this problem. We present a system, vHaul, which detects the resource utilization and the migration time for each VM within a multi-tier application, and computes the optimal multi-VM migration scheme. Our evaluation results indicate the effectiveness and significant performance benefits brought by vHaul. Though the prototype of vHaul is built on Xen using pre-copy live migration technique, it is easily portable to other VMMs and able to

leverage advanced live migration techniques to achieve better performance.

References

[1] Amazon. aws reference architecture. http://aws.amazon.com/architecture/.

[2] Apache olio. http://incubator.apache.org/projects/olio.html.

[3] Iostat. http://linux.die.net/man/1/iostat.

[4] The new business continuity: Moving applications across data centers without interruption. http://www.brocade.com/downloads/documents/solution_briefs/.

[5] Paravirtualization (pv). http://wiki.xenproject.org/wiki/Paravirtualization_\%28PV\%29.

[6] Sql server on vmware best practices guide. http://www.vmware.com/files/pdf/solutions/SQL_Server_on_VMware-Best_Practices_Guide.pdf.

[7] Sysbench. http://www.storagereview.com/sysbench_oltp_benchmark.

[8] Vmotion. http://www.vmware.com/files/pdf/VMware-VMotion-DS-EN.pdf.

[9] 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.

[10] Xenmotion. http://blogs.citrix.com/2012/08/24/storage_xenmotion/.

[11] Xentop. http://wiki.xen.org/wiki/Xentop\%281\%29.

[12] AKOUSH, S., SOHAN, R., RICE, A., MOORE, A. W., AND HOPPER, A. Predicting the performance of virtual machine migration. In *IEEE MASCOTS* (2010).

[13] AL-KISWANY, S., SUBHRAVETI, D., SARKAR, P., AND RIPEANU, M. Vmflock: virtual machine co-migration for the cloud. In *HPDC* (2011).

[14] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.* (2003).

[15] BREITGAND, D., KUTIEL, G., AND RAZ, D. Cost-aware live migration of services in the cloud. SYSTOR '10.

[16] CLARK, C., FRASER, K., HAND, S., HANSEN, J. G., JUL, E., LIMPACH, C., PRATT, I., AND WARFIELD, A. Live migration of virtual machines. In *USENIX NSDI* (2005).

[17] DESHPANDE, U., SCHLINKER, B., ADLER, E., AND GOPALAN, K. Gang migration of virtual machines using cluster-wide deduplication. In *IEEE/ACM CCGrid* (2013).

[18] HE, T. L. S. M. K. T. X. Clique migration: Affinity grouping of virtual machines for inter-cloud live migration. *Networking, Architecture, and Storage (NAS), 2014 9th IEEE International Conference on*.

[19] HINES, M. R., DESHPANDE, U., AND GOPALAN, K. Post-copy live migration of virtual machines. *ACM SIGOPS operating systems review* (2009).

[20] JIN, H., DENG, L., WU, S., SHI, X., AND PAN, X. Live virtual machine migration with adaptive, memory compression. In *CLUSTER* (2009), IEEE, pp. 1–10.

[21] KELLER, E., GHORBANI, S., CAESAR, M., AND REXFORD, J. Live migration of an entire network (and its hosts). In *ACM HotNets-XI* (2012).

[22] KLEINROCK, L. *Theory, Volume 1, Queueing Systems*. Wiley-Interscience, 1975.

[23] LIU, H., XU, C.-Z., JIN, H., GONG, J., AND LIAO, X. Performance and energy modeling for live migration of virtual machines. In *ACM HPDC* (2011).

[24] NELSON, M., LIM, B.-H., AND HUTCHINS, G. Fast transparent migration for virtual machines. In *Proceedings of the annual conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2005), ATEC '05, USENIX Association, pp. 25–25.

[25] SAPUNTZAKIS, C. P., CHANDRA, R., PFAFF, B., CHOW, J., LAM, M. S., AND ROSENBLUM, M. Optimizing the migration of virtual computers. *SIGOPS Oper. Syst. Rev.*.

[26] VOORSLUYS, W., BROBERG, J., AND VENUGOPAL, S. Cost of virtual machine live migration in clouds: A performance evaluation.

[27] YE, K., JIANG, X., HUANG, D., CHEN, J., AND WANG, B. Live migration of multiple virtual machines with resource reservation in cloud computing environments. In *IEEE CLOUD* (2011).

[28] ZAYAS, E. Attacking the process migration bottleneck. SOSP '87.

[29] ZHANG, X., HUO, Z., MA, J., AND MENG, D. Exploiting data deduplication to accelerate live virtual machine migration. In *CLUSTER* (2010), IEEE, pp. 88–96.

[30] ZHENG, J., NG, T. S. E., SRIPANIDKULCHAI, K., AND LIU, Z. Comma: Coordinating the migration of multi-tier applications. VEE '14.