Alrajeh O, Forshaw M, Thomas N.

Performance of Virtual Machine Live Migration with Various Workloads.

In: 32nd UK Performance Engineering Workshop.
8-9 September 2016, University of Bradford.

This is the authors’ accepted manuscript of a paper that was presented at the 32nd UK Performance Engineering Workshop.

Conference website:

http://computing.brad.ac.uk/ukpew2016/

Date deposited:

31/10/2016

This work is licensed under a Creative Commons Attribution-NonCommercial 3.0 Unported License
Performance of Virtual Machine Live Migration with Various Workloads

Osama Alrajeh∗ Matthew Forshaw† Nigel Thomas‡

Abstract

Virtual machine (VM) consolidation is one of the strategies implemented to accomplish energy efficiency in data centres. Data centres take advantage of VM live migration to reduce the energy consumption without application downtime. However, the cost of VM live migration is not considered in some of the VM consolidation approaches. The key focus of this paper is to show how different workloads can impact the time of VM live migration. We demonstrate through live experiment the link between various workload characteristics and the time of VM live migration. We used the Kernel-based Virtual Machine (KVM) as a hypervisor and SPECjvm2008 benchmark to generate various workloads. Our results show a link between VM migration time and memory size of the VM as well as the speed of the network. We also provide a testing framework to facilitate automated experimentation and benchmarking of VM live migration by other researchers.

1 Introduction

Cloud computing has been achieved mainly due to the ability of modern equipment and large-scale manufacturing processes that are able to deliver inexpensive, convenient and user-friendly products to consumers all over the world. Virtualisation is a core component of Cloud computing which powers the Cloud due to various benefits such as partitioning, isolation, easy manageability, cost efficiency and flexibility.

Virtual machine live migration is one of the features that provided by the hypervisor. Live migration refers to the procedure of moving a running VM between physical hosts without powering down the VM. It widely used in data centres due to its ability of energy management, load balancing, and fault tolerance [1]. However, some of VM management approaches that take advantage of live migration such as VM consolidation do not consider the cost of VM live migration.

In this paper, we perform a live experiment to measure the duration time of the VM live migration. We use two physical hosts to migrate VMs between them by using Kernel-based Virtual Machine (KVM) [2]. We choose SPECjvm2008 benchmark [3] to generate workloads on the VMs due to its ability to produce

∗School of Computing, Newcastle University, o.alrajeh@ncl.ac.uk
†School of Computing, Newcastle University, matthew.forshaw@ncl.ac.uk
‡School of Computing, Newcastle University, nigel.thomas@ncl.ac.uk
various workloads characteristics. Then, we compare the migration time of the same workloads on VMs whose hardware characteristics differ.

The rest of the paper is structured as follows. Section 2 introduces related work. In Section 3 we present the experiment environment. We present and evaluate the results of our preliminary experimentation in Section 4, before concluding and discussing future work in Section 5.

2 Motivation and related work

Our main research interest is VM consolidation in large-scale computing which is one of the strategies implemented to accomplish energy efficiency in data centres. The cost of live migration is not considered in most of VM live migration models. In some cases, the energy consumption of VM live migration can be higher compared to keeping the VM running in its current physical machine. It is important to know the parameters that influence the cost of migration before starting the migration of VMs to ensure the most significant discussion on saving energy.

Beloglazov et al. [4] proposed algorithms and policies for live migration and dynamic VM consolidation in order to reduce the energy consumption and SAL violations. They used the CloudSim toolkit to evaluate their techniques. They classified the physical machines into overload machines and under load machines in order to reallocate the VMs. Furthermore, they implemented three different selection VM policies to decide which VM should be emigrated from the current host to new host. However, their approach does not guarantee that the cost of VM live migration does not surpass the benefit of it.

Feller et al. [5] used the multi-dimensional bin-packing (MDBP) problem to solve the workload consolidation difficulty. Also, they designed a novel nature-inspired workload consolidation algorithm based on the Ant Colony Optimization algorithm (ACO) for energy efficient Cloud Computing. Their aim is to reduce the number of physical machines that are used to compute the current workload. The authors developed a simulator based on Java in order to evaluate their approach due to limitations of the CloudSim tool by comparing ACO with First-Fit Decreasing algorithm (FFD). The simulation results demonstrated that ACO saved much more energy than FFD. However, the proposed method does not consider the migration cost.

Forshaw et al. [6] used a high-level, trace-driven, simulation to evaluate the energy consumption and performance of various checkpointing strategies in High Throughput Computing (HTC) systems. Thier checkpointing strategies help to decide when to make energy-efficient checkpoints within HTC systems without affecting the performance. In near future, we are going to add the virtualisation to their simulation [7] and replace the checkpointing strategies with VM consolidation strategies.

Clark et al. [1] proposed the idea of VM live migration technique. Their point was to relocate the VM from source to target host with low downtime. They designed a pre-copy algorithm to accomplish their objective. Before switching the VM’s execution host, the algorithm copies pages from the source memory host to the target memory host until the number of pages remaining is sufficiently small. Then, the VM on the source host suspends and resumes on the target host after transferring the remaining pages from the source host.
Strunk et al [8] measured the energy overhead and the duration of VM live migration with various RAM sizes of the VM. Also, they used different network bandwidth capacities between two servers for VM migration. They used KVM as a hypervisor and developed their own tool to stress the memory. Their results showed that the time of migration increases when the memory size of VM increases and the time of migration decreases when the network capacity increases. However, their way of generating the workload did not represent the real VM live migration. Our work avoids this limitation by using a benchmark which generates various workloads characteristics.

Rybina et al [9] investigated the time of VM migration that runs with multiple running virtual machines and the impact of VM migration on the running virtual machines. They used SPEC CPU2006 benchmark suite to generate CPU Intensive workloads. They showed that starting to migrate VMs with intensive memory workload first is cheaper than migration VMs with intensive CPU workload first. However, their experiment base on homogeneous physical hosts which we avoid in this paper.

3 Experiment environment

In this paper, we use a live experiment to measure the time of VM live migration with various workloads. We use the Kernel-based Virtual Machine (KVM) [2] as a hypervisor and SPECjvm2008 [3] benchmark to generate various workloads. Section 3.1 discusses set up of the experiment and Section 3.2 introduces the benchmark that generates VM’s workloads, while Section 3.3 describes the scenario we experiment in the rest of this paper.

3.1 Experiment set up

![Image of VM live migration experiment setup](image)

Figure 1: The setup of VM live migration experiment

The setup of the experiment reflects the process of VM live migration in the real world. Accordingly, we used Kernel-based Virtual Machine (KVM), which is widely used in data centres. The VM live migration requires saving the VMs’ images in a storage area that is accessible to all physical hosts. Therefore, we used network attached storage (NAS) to save VMs’ images, which has caused the process of VM live migration limited to copy the memory pages and the
CPU state from one physical host to another. This technique is known as pre-copy live migration [1]. We used the free software licensed Openfiler [10] as NAS. Furthermore, our experiment setup contains two servers and one client computer, which is all connected with one another by a 100Mb switch (Figure 1). Each server runs CentOS 7 Linux [11] and has KVM installed on it. Server 1 employs 8 CPUs Intel Core i7 @ 2.80GHz and 4 GB DDR3 SDRAM and server 2 has 4 CPUs Core 2 Quad @ 2.66GHz and 4 GB DDR2 SDRAM. The difference between the two servers is a factor that assists us to obtain various results. The client computer runs Ubuntu 16.04 LTS [12] and employs 1 CPU Intel Core i7 @ 3.20GHz and 4 GB DDR3 SDRAM. The client computer is used to trigger the VM live migration between hosts.

3.2 Benchmarks

The SPECjvm2008 benchmark [3] includes 39 different workloads that test the performance of Java virtual machines (JVM) and hardware systems. The default run time for each workload is 4 minutes. The benchmark contains two running modes: base and peak. The base has a fix running time duration which is 120 seconds warm-up, continued by 240 seconds. Users can only tune JVM to raise the performance when they run the benchmark on the peak mode. In our experiment, we use 21 workloads of the SPECjvm2008 benchmark as illustrated in Table 1. Also, We use the base mode to run the benchmark without warm-up time duration. The remaining part of this section gives a brief description of each workload as mentioned in [14].

Table 1: SPECjvm2008 Benchmark workloads

<table>
<thead>
<tr>
<th>Group name</th>
<th>Workloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler</td>
<td>compiler.compiler,</td>
</tr>
<tr>
<td></td>
<td>compiler.sunflow</td>
</tr>
<tr>
<td>Compress</td>
<td>compress</td>
</tr>
<tr>
<td>Crypto</td>
<td>crypto.aes, crypto.rsa,</td>
</tr>
<tr>
<td></td>
<td>crypto.signverify</td>
</tr>
<tr>
<td>Derby</td>
<td>derby</td>
</tr>
<tr>
<td>Mpegudio</td>
<td>mpegudio</td>
</tr>
<tr>
<td>Scimark Large</td>
<td>scimark.fft.large,</td>
</tr>
<tr>
<td></td>
<td>scimark.lu.large,</td>
</tr>
<tr>
<td></td>
<td>scimark.sor.large,</td>
</tr>
<tr>
<td></td>
<td>scimark.sparse.large,</td>
</tr>
<tr>
<td></td>
<td>scimark.monte_carlo</td>
</tr>
<tr>
<td>Scimark Small</td>
<td>scimark.fft.small,</td>
</tr>
<tr>
<td></td>
<td>scimark.lu.small,</td>
</tr>
<tr>
<td></td>
<td>scimark.sor.small,</td>
</tr>
<tr>
<td></td>
<td>scimark.sparse.small,</td>
</tr>
<tr>
<td></td>
<td>scimark.monte_carlo</td>
</tr>
<tr>
<td>Serial</td>
<td>serial</td>
</tr>
<tr>
<td>Sunflow</td>
<td>sunflow</td>
</tr>
<tr>
<td>Xml</td>
<td>xml.transform, xml.validation</td>
</tr>
</tbody>
</table>
3.2.1 Compiler

There are two workloads within the compiler group, namely compiler and sunflow. The compiler.sunflow workload determines the sunflow benchmark’s compilation, while the OpenJDK compiler’s compilation time is measured by the compiler.compiler workload. Input data is stored either in a file cache or in memory to reduce the impact of I/O as the aim of these two workloads is the evaluation of the compiler’s performance.

3.2.2 Compress

The workload uses a modified Lempel-Ziv technique to compress and decompress data. The algorithm utilizes pseudo-random access based on the input data, which is extended to 34.36 MB from 90 KB. Data is buffered to minimise the impact of I/O and the compression is done using internal tables of approximately 67 KB. As JVM produces and operates on mixed length data accesses, the compress workload tests inlining, array access, just in time coupling and cache performance.

3.2.3 Crypto

There are three workloads within the crypto group crypto.signverify, crypto.aes and crypto.rsa. Between them, they cover three important aspects of cryptography and test not only JVM execution but also different vendors protocol implementations. The crypto.rsa works on input data of 16 KB and 100 bytes and encrypts and decrypts using the RSA protocol. The crypto.aes encrypts and decrypts data. This is done according to the AES and DES protocols, using CBC/PKCS5P adding and CBC/NoP adding. The respective input data sizes are 100 bytes and 713 KB, respectively. The crypto.signverify, as its name suggests, signs and verifies protocols. In particular, it does this with SHA1 with DSA SHA1 with RSA, SHA256 with RSA and MD5 with RSA, for input data sizes 1KB, 65 KB and 1 MB.

3.2.4 Derby

An open source, pure Java database called derby is used by this workload; several databases are instantiated each time the workload is started, with every four threads sharing a common database instance. Derby tests synchronization, database and BigDecimal operations. It took forward the telco benchmark of IBM such that it could synthesize business logic and test BigDecimal operations use. The BigDecimal calculations in this workload are longer than 64-bit.

3.2.5 Mpegaudio

The mpegaudio workload is based around floating-point calculations and uses as an MPEG audio decoder the JLayer MP3 library. The input data files, whose sizes range from 20 KB to 3 MB, are six MP3 files.

3.2.6 Scimark

Scimark comprises a group of workloads that together evaluate data access patterns and floating-point operations in demanding mathematical calculations. It
is based around the The scimark workloads are arranged into two groups scimark.small and scimark.large, according to the dataset size. Each workload thread uses one dataset; the small group uses a 512 KB dataset to simulate the performance of in-cache access while the large group uses a 32 MB dataset in order to reproduce the out of cache access performance. Each group comprises five workloads. These are monte_monte_carlo, sparse, lu, ftt and sor. scimark.monte_carlo runs once but is counted in both scimark.large and scimark.small; the workload does not operate on differently sized datasets.

3.2.7 Serial

This workload exercises the java.lang.reflect and examines the serialization and deserialization of primitives and objects. The performance of these processes is evaluated using a dataset taken from a JBoss benchmark in memory byte arrays. Serial acts in a producer-consumer situation, in which the producer threads serialize the objects while, on the same system, the consumer threads deserialize them.

3.2.8 Sunflow

The sunflow workload is multi-threaded and runs a number of bundles of dependent threads. The workflow is reconfigurable. However, generally, there are four threads per bundle and as many bundles as there are hardware threads. Additionally, being floating-point intensive, the workload has a high object allocation rate, exercising the memory bandwidth. It is used as a benchmark simulating visualization and graphics using ray tracing.

3.2.9 XML

Two workloads, xml.transform and xml.validate, constitute the XML group. Both have high rates of contended locks and object allocation and exercise string operations intensively. By performing XSLT transformations with SAX and DOM stream sources, xml.transform exercises the JAXP implementation. The workload utilizes ten use cases from real life and the XSLTC engine (this compiles xsl stylesheets into java classes). The xml.validation workload also exercises the JAXP implementation and used just six use cases from real life.

3.3 Experiment scenario

We created a VM in server 1 by using KVM and the image was saved in NAS. The VM runs Ubuntu 16.04 LTS as an operating system and it has the SPECjvm2008 benchmark to generate various workloads. We used three different VMs' hardware capacities; VM with 1 CPU and 1 GB of RAM, VM contains 2 CPUs and 2 GB of RAM, VM has 3 CPUs and 3 GB of RAM. We call them VM 1, VM 2, and VM 3.

In our experiment, we use 21 workloads of the SPECjvm2008 benchmark and we run them individually as base mode. We move the VM from server 1 to server 2 after one minute from the beginning of running the workload.

The client computer is used to run the experiment in an automated way. We developed a bash script [16] that allowed the client computer to run the benchmark on the VM and trigger the live migration between server 1 and
server 2. The client accesses the VM and runs one of the workloads. After one minute of running the workload, the client migrates the VM from server 1 to server 2. The client then records the start and the end times of VM live migration in a logs file. Also every second, the script records the memory usage of each workload during its run time. We used Memusg script [13] to measure the memory usage of processes. In addition, we used Top [15] to record the CPU utilisation and memory usage of the VM during the migration. The bash script runs 10 times for each workload and each time we restart the VM.

The automated bash script is available in here [16]. The script allows other researchers to perform the experiment on their own hardware. The number of the test can be modified as well as the run time of the workloads. The script is able to record the start and end time of the workload, the start and end time of migration, and the memory usage of the workload.

4 Results

In this section, we present the results of our preliminary experimentation which show the different workload characteristics impacts of VM live migration time on various VMs capacities.

Table 2: Migration time and memory usage of VM 1, VM 2, and VM 3

<table>
<thead>
<tr>
<th>Workloads</th>
<th>VM 1 Migration Time</th>
<th>Memory usage</th>
<th>VM 2 Migration Time</th>
<th>Memory usage</th>
<th>VM 3 Migration Time</th>
<th>Memory usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>compiler.compiler</td>
<td>231</td>
<td>219202</td>
<td>268</td>
<td>639968</td>
<td>317</td>
<td>909196</td>
</tr>
<tr>
<td>compiler.sunflow</td>
<td>222</td>
<td>190620</td>
<td>232</td>
<td>358673</td>
<td>264</td>
<td>470702</td>
</tr>
<tr>
<td>compress</td>
<td>210</td>
<td>115873</td>
<td>212</td>
<td>159170</td>
<td>228</td>
<td>226510</td>
</tr>
<tr>
<td>crypto.aes</td>
<td>209</td>
<td>81937</td>
<td>215</td>
<td>213467</td>
<td>237</td>
<td>330427</td>
</tr>
<tr>
<td>crypto.rsa</td>
<td>207</td>
<td>92021</td>
<td>208</td>
<td>108605</td>
<td>213</td>
<td>119630</td>
</tr>
<tr>
<td>crypto.signverify</td>
<td>208</td>
<td>78373</td>
<td>210</td>
<td>116095</td>
<td>221</td>
<td>141932</td>
</tr>
<tr>
<td>derby</td>
<td>442</td>
<td>345463</td>
<td>594</td>
<td>539221</td>
<td>630</td>
<td>626739</td>
</tr>
<tr>
<td>mpegaudio</td>
<td>207</td>
<td>100632</td>
<td>210</td>
<td>118431</td>
<td>217</td>
<td>132093</td>
</tr>
<tr>
<td>scimark.fft.large</td>
<td>212</td>
<td>169702</td>
<td>254</td>
<td>533904</td>
<td>269</td>
<td>639192</td>
</tr>
<tr>
<td>scimark.lu.large</td>
<td>223</td>
<td>227558</td>
<td>262</td>
<td>490671</td>
<td>293</td>
<td>738374</td>
</tr>
<tr>
<td>scimark.sor.large</td>
<td>211</td>
<td>118444</td>
<td>221</td>
<td>183212</td>
<td>233</td>
<td>217745</td>
</tr>
<tr>
<td>scimark.sparse.large</td>
<td>218</td>
<td>169940</td>
<td>219</td>
<td>236246</td>
<td>283</td>
<td>725774</td>
</tr>
<tr>
<td>scimark.fft.small</td>
<td>106</td>
<td>76095</td>
<td>209</td>
<td>94235</td>
<td>220</td>
<td>99275</td>
</tr>
<tr>
<td>scimark.lu.small</td>
<td>210</td>
<td>78432</td>
<td>210</td>
<td>99344</td>
<td>225</td>
<td>106371</td>
</tr>
<tr>
<td>scimark.sor.small</td>
<td>209</td>
<td>81134</td>
<td>218</td>
<td>137706</td>
<td>240</td>
<td>227848</td>
</tr>
<tr>
<td>scimark.sparse.small</td>
<td>210</td>
<td>80910</td>
<td>210</td>
<td>94887</td>
<td>219</td>
<td>97246</td>
</tr>
<tr>
<td>scimark.monte_carlo</td>
<td>208</td>
<td>78741</td>
<td>209</td>
<td>94415</td>
<td>221</td>
<td>102125</td>
</tr>
<tr>
<td>serial</td>
<td>209</td>
<td>158552</td>
<td>227</td>
<td>362733</td>
<td>246</td>
<td>503948</td>
</tr>
<tr>
<td>sunflow</td>
<td>207</td>
<td>89595</td>
<td>236</td>
<td>277815</td>
<td>235</td>
<td>403761</td>
</tr>
<tr>
<td>xml.transform</td>
<td>228</td>
<td>136392</td>
<td>255</td>
<td>325566</td>
<td>280</td>
<td>419212</td>
</tr>
<tr>
<td>xml.validation</td>
<td>210</td>
<td>126865</td>
<td>271</td>
<td>572309</td>
<td>272</td>
<td>745437</td>
</tr>
</tbody>
</table>

Table 2 presents the results of VMs migration from server 1 to server 2. The VMs migration starts after one minute of running the workload. We migrated each VM with each workload 10 times from server 1 to server 2 to obtain the results. The table shows the average time of the VM live migration in seconds and the average peak memory usage of the workload in megabytes. Our results demonstrate the impact of various workloads characteristics on migration time.
with various VM capacities. We see significant increases in average migration time when the size of memory is increased. The reason is that live migration process copies the memory pages from the source host (server 1) to the destination host (server 2). When the memory size of VM is increased, the time of live migration increases due to the increment on memory pages. In our experiment, VM 2 and VM 3 take more time to be migrated than VM 1 due to the memory size of VM 2 and VM 3 which are bigger than the memory size of VM 1. Figure 2 illustrates the VM migration time of VM 1, VM 2, and VM 3.

Figure 2: Average migration time of VM 1, VM 2, and VM 3 with various workloads

Figure 3: Average peak memory usage of each workload on various VMs

The number of operations in the SPECjvm2008 benchmark is based on the available resources. When there are more resources available in the server, the
workloads generate more operations to test these resources. In our experiment, the workloads in VM 2 and VM 3 generates more memory pages than VM 1 due to the fact that they contain a bigger memory size compare to VM 1. Figure 3 shows the average peak memory usage of each workload on each VM.

The results do not show a link between the migration time of the VMs and the peak memory usage of the workloads. For example, Compiler.compiler workload in VM 2 and VM 3 uses more memory than Derby workload, but the VM with Derby workload takes a longer time to be migrated than the VM with Compiler.compiler. Also, the results from one VM do not present a high variation in migration time between the workloads event though the workloads memory usage are different. For example, Compress and Crypto.aes workloads on VM 1 have the same migration time, but the Compress workload uses more memory. In addition, the standard error bars are shown for all results. Here we see that average migration time and peak memory usage demonstrate small variance, across each benchmarks and VM configuration.

![Graphs showing memory usage](image)

Figure 4: Memory usage of the VM 1 with Compiler.compiler, Crypto.aes, Derby, Scimark.lu.large, Scimark.sor.small, and Xml.transform workloads.

This is encouraged us to investigate more to find out the reason behind that. We modified our bash script to record the CPU utilisation and the memory usage
of the VM during workload running time and migration procedure. Figures 4 shows the memory usage of the VM 1 with Compiler.compiler, Crypto.aes, Derby, Scimark.lu.large, Scimark.sor.small, and Xml.transform workloads. The VM migration starts after 60 seconds of running the workload and the workload runs for 240 seconds. As illustrated in Figure 4, the memory usage of the VM 1 starts to drop when the run time of workload is finished except Derby workload.

During the live migration procedure, the VM remains running on a source host until it suspends and moves to a destination host. In our experiment, the VM continues running on server 1 till the memory usage of the VM drops then it migrates to the server 2. This is the reason for the low variation between the average migration time of the workloads within one VM. This is the reason for the low variation in the average migration time between the workloads within one VM as well as the high average migration time of VMs with Derby workload.

Figure 5: CPU utilisation of the VM 1 with Compiler.compiler, Crypto.aes, Derby, Scimark.lu.large, Scimark.sor.small, and Xml.transform workloads

In our experiment, we use 100 Mb switch to connect our devices together. The available bandwidth in our network is 94.6 Mbits/sec. The VM live migration requires that the memory pages be moved from the resource host to the destination host. At the point when there is a high rate of changing the mem-
ory pages during the live migration with a low network speed, the migration might never finish or might take a long time. We changed the default run of the workloads to be 1800 seconds instead of 240 seconds. The VM live migration is finished after a few seconds of the workload finishing run time. That because of the low changing rate of memory pages which makes our network cope with it.

In addition, we looked at the CPU utilisation of the VMs during the run time of the workload and migration procedure. The CPU utilisation of workloads with one VM is about the same except Derby workload. Figure 5 exhibits the CPU usage of the VM 1 with Compiler.compiler, Crypto.aes, Derby, Scimark.lu.large, Scimark.sor.small, and Xml.transform workloads. Figure 5 verifies that the CPU utilisation of the VMs with various workloads characteristics does not affect the migration time between server 1 and server 2. For instance, the Derby workload uses less CPU but takes more time to be migrated. Also, the VM peak utilisation of CPU is around 60% with the others workloads and due to the page limitation we did not include figures for all of them.

5 Conclusions

This paper has measured the live migration time of different workload characteristics on various VMs capacities. We used KVM as hypervisor and SPECjvm2008 benchmark to generate the workloads. We provided a bash script which automated the run of the experiment. Our results showed an important link between the time of VM live migration and the memory size of VM. When the memory size of VM increases, the time of VM live migration increases. Also, our results showed that the live migration time of a single VM on a host does not affect by CPU utilisation of the VM. In addition, the results demonstrated a strong link between the time of live migration and network bandwidth. The live migration time is depending on the network speed. VM live migration is required a high-speed network to get the most of it.

In our ongoing work, we are measuring the live migration time with a higher network speed. Also, we are extending our experiment to measure the energy consumption of VM live migration. This will allow us to make an efficient energy decision on VM consolidation. Furthermore, in our current work we use KVM; in the future we hope to compare the results of KVM with other hypervisors such as Xen and VMware.

In addition, we are going to use our achievement in this paper to extend our previous work in [17]. We are aiming to add the virtualization on the simulation tool. Then, we will evaluate the energy consumption of the VM consolidation strategies.

References


