iMIG: Toward an Adaptive Live Migration Method for KVM Virtual Machines

JIANXIN LI1*, JIEYU ZHAO1, YI LI1, LEI CUI1, BO LI1, LU LIU2
AND JOHN PANNEERSELVAM2

1State Key Laboratory of Software Development Environment, School of Computer Science & Engineering,
Beihang University, Beijing, China
2School of Computing and Mathematics, University of Derby, Derby, UK
*Corresponding author: lijx@act.buaa.edu.cn

With the energy and power costs increasing alongside the growth of the IT infrastructures, achieving workload concentration and high availability in cloud computing environments is becoming more and more complex. Virtual machine (VM) migration has become an important approach to address this issue, particularly; live migration of the VMs across the physical servers facilitates dynamic workload scheduling of the cloud services as per the energy management requirements, and also reduces the downtime by allowing the migration of the running instances. However, migration is a complex process affected by several factors such as bandwidth availability, application workload and operating system configurations, which in turn increases the complications in predicting the migration time in order to negotiate the service-level agreements in a real datacenter. In this paper, we propose an adaptive approach named improved MIGration (iMIG), in which we characterize some of the key metrics of the live migration performance, and conduct several experiments to study the impacts of the investigated metrics on the Kernel-based VM (KVM) functionalities, as well as the energy consumed by both the destination and the source hosts. Our results reveal the importance of the configured parameters: speed limit, TCP buffer size and max downtime, along with the VM properties and also their corresponding impacts on the migration process. Improper setting of these parameters may either incur migration failures or causes excess energy consumption. We witness a few bugs in the existing Quick EMUlator (QEMU)/KVM parameter computation framework, which is one of most widely used KVM frameworks based on QEMU. Based on our observations, we develop an analytical model aimed at better predictions of both the migration time and the downtime, during the process of VM deployment. Finally, we implement a suite of profiling tools in the adaptive mechanism based on the qemu-kvm-0.12.5 version, and our experiment results prove the efficiency of our approach in improving the live migration performance. In comparison with the default migration approach, our approach achieves a 40% reduction in the migration latency and a 45% reduction in the energy consumption.

Keywords: cloud computing; live migration; energy saving; configures; adaptive model

Received 17 December 2013; revised 1 May 2014
Handling editor: Zhangbing Zhou

1. INTRODUCTION

In recent years, cloud computing [1] has emerged as an efficient paradigm which enables ubiquitous, on-demand services to the continuous public computing and storage requirements. The continuous operation of massive datacenters and other computing resources in the cloud consumes a notable amount of energy and the level of the energy consumption has now become a serious question in the cloud arena. Virtualization has become an enabling technology for cloud computing as it simplifies the service delivery by providing a platform for optimizing complex IT resources in an effective and energy efficient way [2]. In the cloud datacenter, virtual machine (VM) encapsulates the

SECTION B: COMPUTER AND COMMUNICATIONS NETWORKS AND SYSTEMS
execution of a complete operating system (OS), allowing the implementation of multiple virtual OS environments to co-exist on the same computer with strong isolation between each other [3, 4]. Live migration [5] is a key feature of the VMs, which allows moving a continuously running VM from one physical host to another. The attractive benefits of this live migration includes energy saving, high availability and load balancing, etc. Various research works are being carried out recently, to optimize the performance of live migration, such as pre-copy [6], post-copy [7] and trace and replay [8]. Though these studies look comprehensive [6, 9–13], they do not account for the fact of configuration parameters affecting the migration time, the downtime of the VMs and also the power of the computing nodes.

In a dynamic scenario, attempting a quick VM migration with a speed of 100 MB/s in a low bandwidth network of 30 MB/s, often results in a poor performance. In such a case, deploying the task with right parameters and performance metrics plays a crucial role in the overall performance of the network. With this in mind, we highlight the need for the consideration of the parameters such as bandwidth, service-level agreements (SLA), local availability, current network status and many, while attempting a VM migration in the cloud. An SLA, which is directly related to the perceivability of the end users, is an important part of the VM services. SLAs are mostly specified before the deployment of the VMs, which requires the cloud providers to gain a prior knowledge about the desired configuration parameters in order to satisfy the agreement in terms of the migration performance and the datacenter energy management criterion. Migration is a complex process affected by many factors such as network bandwidth, local system capability, etc. The cloud environment is generally dynamic, and so the pre-configured parameters should be adopted according to the dynamic environments and the energy management requirements. An improper parameter setting might lead to unexpected migration failures, thus performance degradations.

Figure 1 depicts such an example where the max downtime is configured as 200 ms. Here, the migration is lasting for a longer time than expected, because of the fact that the estimated downtime is larger than 200 ms and also the migration is not allowing a stop-and-copy phase. In this case, the migration is considered to be a failure. VM migration is very common in a datacenter environment with hundreds of VMs, and hence the energy consumption criterion is always attached to it. Hence, the adopting strategies of these pre-configured parameters to achieve the desired performance of live migration are important and desirable. To address the above problems, we propose an adaptive live migration approach named improved MIGration (iMIG) based on Quick EMUlator (QEMU)/Kernel-based VM (KVM) [14]. Kernel-based VM (KVM) is a Linux kernel module that allows a user space program to utilize the hardware virtualization features of various processors. QEMU makes use of KVM when running a target architecture that is same as the host architecture. The KVM project is maintaining a fork of QEMU called qemu-kvm. We conduct several experiments in our study based on this QEMU/KVM framework.

The design goal is a 2-fold strategy, as shown in Fig. 2. The first is to study the parameters and analyze their configured relationships with the VM via empirical study, along with their impacts on the VM live migration performance. The second phase is to develop an analytical model with an adaptive mechanism to improve the migration performance and the success ratio. The major contributions of the paper are as follows:

(i) The impacts of the key parameters affecting the KVM live migration have been evaluated through empirical studies based on QEMU/KVM. By analyzing the results of the KVM live migration performance under different conditions, we have identified a few constraints and also...
observed the impacts of these studied parameters on the live migration performance.

(ii) We design a new adaptive migration mechanism, called iMiG, to reduce the migration duration in order to gain a better migration performance addressing the existing issues of live migration in KVM. During our study, we have also identified a few drawbacks of the existing migration mechanism, and adopted the necessary modifications to achieve better performance.

(iii) The proposed mechanism, iMiG, achieves better performances and exhibits a higher adaptability in comparison with the existing live migration mechanism of QEMU/KVM. Also, iMiG provides a suit of profiling tools and methodologies for guiding the administrator to deploy the VMs effectively.

The remaining of this paper is organized as follows: Section 2 gives the description of our experiments and results. In Section 3, we establish a model to describe the impact of the configuration parameters on the overall migration process. Section 4 represents the design and the implementation of iMiG. In Section 5, we discuss the related research works focused on the live migration. Section 6 concludes this paper and along with our future research directions.

2. OVERVIEW AND EMPIRICAL STUDY

In this section, we give a brief introduction to the functionalities of the KVM live migration [15] and an overview of iMiG, and then we present our experiments in detail, including the environment that we use. This section also describes the configuration parameters and their impacts on the migration performance.

2.1. Overview of iMiG methodology

Live migration is affected by many parameters, such as network bandwidth, dirty memory speed (which reflects on the application workload) and so on. The initial operation of iMiG is to assess the relationship between the configuration parameters and the live migration performance and their corresponding impacts. Based on this assessment and observation, we build our model, which can be used for adaptive migration guidance and SLA prediction. Thus, iMiG provides a patch on QEMU/KVM to enable an adaptive migration and improves the performance of existing migration.

2.2. Description of KVM live migration

KVM is an open source system virtualization module, which has been integrated in every major distribution of Linux since Linux 2.6.20 [16], and it leverages the simulated devices implemented in QEMU. QEMU/KVM has now become the mainstream VM Monitor (VMM) in academia and industry, supporting a wide variety of guest OSs.

To evaluate the performance of KVM live migration, we must analyze the core codes related to migration. KVM live migrations do not involve basic core state, and is mainly implemented in QEMU. Function do_migrate (Monitor *mon, const QDict *qdict, QObject **ret_data) is the entry function of the live migration. The parameter uri implies that the migration is be divided into four modes, i.e. tcp, exec, unix and fd. The files Migration-tcp.c, migration-exec.c, migration-fd.c and migration-unix.c correspond to the four migration methods, respectively. The tcp way is the main mode used in practice, which is adopted in this study.

KVM uses the pre-copy strategy [6] to do the migration, as shown in Fig. 3. The first stage of migration is carried out by the function qemu_savevm_state_begin (Monitor *mon, QEMUFile *f, int blk_enable, int shared). And then the migrate_fd_put_ready (void *opaque) iterately copy the dirtied pages generated during the last transformation. Finally, the migrate_fd_cleanup stops the VM, copies the remaining dirty pages, closes and releases the socket, frees other resources, and sets the state as MIG_STATE_COMPLETED, to complete the migration process.

We need some principles to justice this migration and there are two key metrics to measure the performance of live migration [9].

- **Migration time.** It reflects the total time from which the migration begins to the time it totally finishes. We denote the migration time by $T_{\text{mig}}$.
- **Downtime.** It describes the maximum time allowable, during which the service is unavailable and is denoted by $T_{\text{down}}$.

Based on the source code analysis, we found that there are several parameters that impact these two metrics, shown in Table 1.

2.3. The experimental environment setup

The environment is a PC cluster with 32 nodes connected by 1 Gbps network. We choose two hosts from them with Debian-6.0-amd-64 installed, and the kernel version 2.6.32.5, and the QEMU version is 0.12.5. During the experiments, we use Network File System (NFS) [17] to provide these two hosts with the file sharing service. NFS Sever is set in Host1, and Host2 can mount the sharing files by the ‘nfs mount’ command. The image of the VM waiting to be migrated is saved in the NFS directory. We use a Voltech PM1000+ power analyzer unit to measure and record the transient power and the total energy consumption.

The default values of some key parameters are shown in Table 2. The workload running on the VM is a program that randomly allocates a specified memory size and writes data on to it. In the experiments, we vary some of the default
TABLE 1. Parameters of iMIG.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Denotation</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM memory</td>
<td>$M$</td>
<td>The larger memory size, the longer it will take to finish the migration</td>
</tr>
<tr>
<td>Migration speed limit (used bandwidth)</td>
<td>$speed_limit$</td>
<td>It stands for the real-time throughput during the migration. We denoted it as $B$</td>
</tr>
<tr>
<td>Page dirty rate (workload)</td>
<td>$T_{\text{max-down}}$</td>
<td>It describes the longest time allowable during which the VM is unavailable</td>
</tr>
<tr>
<td>Max downtime</td>
<td>$T_{\text{max-down}}$</td>
<td>It describes the longest time allowable during which the VM is unavailable</td>
</tr>
<tr>
<td>TCP_Buffer size</td>
<td>$\text{tcp_buffer}$</td>
<td>It represents the size of TCP_Buffer in our experiments</td>
</tr>
<tr>
<td>Iterate count</td>
<td>$N$</td>
<td>It states the count of the iterate copy happens during the migration</td>
</tr>
<tr>
<td>$i$th Round</td>
<td>$I$</td>
<td>It means that at the current time, it is the $i$th round of the migration</td>
</tr>
<tr>
<td>Time of current round</td>
<td>$t_i$</td>
<td>It reveals the time it takes to finish the current round copy</td>
</tr>
</tbody>
</table>

TABLE 2. Default parameters value.

<table>
<thead>
<tr>
<th>VM memory</th>
<th>Workload</th>
<th>Max downtime</th>
<th>TCP_Buffer size</th>
<th>Speed limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 GB</td>
<td>20 MB/s</td>
<td>1000 ms</td>
<td>16 MB</td>
<td>50 MB/s</td>
</tr>
</tbody>
</table>

values to identify the associated parameters. For example, we set the default value of max downtime 1000 ms, but in some experiments, we change this value to 300 or 30 ms with special statements.

2.4. The experimental results

During the experiments, we change one parameter at a time to, respectively, study the effect of the other parameters on the final migration performance. The experiment mainly uses the max downtime, speed limit, TCP_Buffer size, rate of dirty pages and VM memory size for analysis. Using Statistical Product and Service Solutions (SPSS) software to fit the curve of the experiment results, the specified impact of every single parameter on the migration performance is obtained, which in practice can give guidance for efficient migration.

(i) Impact of speed limit
In this experiment, we vary the speed limit from 10 to 200 MB/s and study the changes in the migration time and the downtime. We analyze the causes for this change and use curve fitting method to study the specific impact of the speed limit on migration time and downtime.

**Migration time and downtime.** From Figs 4 and 5, it can be seen that the impact of the speed limit on the total migration time is roughly inversely proportional to each other. And for downtime, the curve escalates at the beginning and then turns into a smooth fluctuation, as the speed limit increases. The inverse proportion of the speed limit on migration implies that with the initial increase of the speed limit (about <100 MB/s in the experiment), the migration time reduces and after a certain level of the speed limit (say, 100 MB/s), the effect of the speed limit on the migration time is negligible. We use SPSS to analyze the results precisely, to obtain Equation (1). Detailed data may not be the same, but this method can be used to analyze the results of the experiments conducted in a totally different environment. The quantitative analysis of the SPSS enables better predictions about the impacts of the parameters on the SLA, reflecting in the improvement of the adaptive migration.

\[ T_{\text{mig}} \approx \frac{1700}{x} + 8. \quad (1) \]

**Network throughput.** We also measured the real-time bandwidth (throughput) during the migration process, shown in Figs 6 and 7. From Fig. 6, it is evident that the throughput increases with the speed limit. Also, when the speed limit exceeds 80% of the physical bandwidth (1 Gbps), the throughput fluctuates rapidly, as shown in Fig. 7. So we advise not to consume all the bandwidth during migration, doing so might lead to the instability of the network and degrades the performance quality.

**Energy consumption.** During the experiments, we measure the energy consumed when the speed_limit is 10, 30 and 100 MB/s, respectively. The results of which are shown in Figs 8...
and 9, from which we can see that increasing the speed increases the power consumption of the hosts and decreasing the speed results in excess energy consumption.

(ii) Impact of TCP_Buffer size

The analysis of the QEMU/KVM source code, reveals that the TCP Buffer size affects the control of the iteration loop, which in turn affects the entire migration process. We have also conducted experiments to study the impact of the TCP Buffer size on the migration process.

Migration time and downtime. From Figs 10 and 11, it can be observed that with the increase in the TCP_Buffer size, the total migration time reduces and reaches a stable value, i.e. the migration time lasts \(\sim 34\) s when TCP_Buffer size is \(>6\) MB. The downtime also shows similar trend, that is, the downtime lasts for 450 ms when TCP_Buffer size is above 6 MB.

Network throughput. In the previous experiment, we saw the stabilization of the curves, as shown in Figs 10 and 11.

To explore this behavior, we change the TCP_Buffer size and explore the throughput, respectively.

Though we set the same speed, the throughput varies, which is shown in Figs 12 and 13. So, we conclude that the throughput is also in relation with the TCP Buffer size.

In Figs 6 and 7, we know that the throughput is relative to the speed limit. A collective illustration of the TCP Buffer size and the speed limit is presented in Figs 14 and 15. From all these figures, we conclude that the throughput is decided by both the TCP_Buffer size and the speed limit.

Energy consumption. We then compare the energy consumption when the TCP_Buffer size is 3 and 16 MB. Figures 16 and 17, respectively, show the energy consumed on the destination and the source host. We can see that when the TCP_Buffer is 3 MB, the power on destination host is \(\sim 58\) W and the migration lasts for \(\sim 60\) s. When the TCP_Buffer is 16 MB, the power...
is ~60 W but the migration latency is only 20 s. The results show us that the latter consumes less energy though its power level is higher. And a similar phenomenon is also seen on the source host. Sometimes, with a larger TCP Buffer size, lower energy consumption can be achieved even with a higher power level.

(iii) Impact of max downtime

Max downtime is one of the key control metrics that allow the live migration to function in the stop-and-copy phase, and thus it has a key influence on the whole migration process. We change the max downtime from 30 to 3000 ms and study how the migration time and downtime varies.

As we can see from Figs 18 and 19, the migration time and downtime, both are proportional to the max downtime. Though we know that the migration time decreases with the increase in
the max downtime, only a minor change (from 32.8 to 34.6 s) the migration time is evident from Fig. 18.

In order to obtain a sharp change in the migration time, the max downtime should be increased considerably. But this practice is against the policies of the SLA and cannot be accepted. From Figs 20 and 21, we also conclude that when the max downtime changes, it has only a little impact on the power consumption of both the source and the destination hosts, during the migration process. But a shorter max downtime will lead to a longer migration latency and thus causes more energy consumption.

(iv) Impact of workload

The workload running on the VM also has a great impact on the migration metrics. In the experiments, we first use our own program to study how the workload affects the performance (Figs 22 and 23). The program allocates certain amount of memory and set random bytes in the memory to simulate...
the memory overheads. Besides, we use Phoronix [18], a set of benchmarks which consists of many Linux open source benchmarks to imitate the workload.

For further analysis, we choose some typical benchmarks to simulate the actual situation. The experimental conditions are listed in Table 3 and adopted benchmarks and the results are listed in Table 4.

From Table 4, we note that imagemagic generates most of the data at run time, and writes most of the memory. Gzip generates the least amount of data. The downtime of gzip is the longest, indicating the presence of more number of dirty pages between the leaving and entering phases of the last data in the stop-and-copy stage. The downtime of tiobench is short, indicating a small number of dirty pages.

In addition, we also choose memcached to carry out the load test. We run memcached server on a VM to carry out memory operations by sending read and write operations from the physical host. We used the data sent out every time and the remaining dirty pages to calculate the generation rate of the current dirty data, since the size of the load cannot be quantified. But in reality, the VMs waiting to be migrated always have a specific workload running on it. Energy consumption should be considered case by case, rather than the energy consumed on the whole occasion.

Figure 24 shows the part-curve of the migration process. This figure shows the memory dirty pages accumulated in each round. It can be found that the workload is lower when the bandwidth is higher, i.e. the workload is 4 and 1.6 MB when the...
speed limit is 20 and 50 MB/s, respectively. This indicates that more number of dirty pages has been sent out to the destination and the number of remaining dirty pages is reducing in unit time. This also highlights the importance of the speed limit on the migration performance. The lower the bandwidth, the bigger is the load accumulated, and this increases the negative effect on the migration performance.

(v) Impact of VM memory

We set the VM memory size to 512, 1024, 1536, 2048 and 2536 MB, respectively (Fig. 25). As expected, we learn that the migration time increases with the increase in the VM memory size. However, the downtime is relative to the speed limit and the data remains only in the last iteration, which shows no direct relation of the migration time to the VM memory size.

We also measure the energy consumed when the VM memory is 2 and 1 GB with all the other conditions remaining the same. From Figs 26 and 27, we conclude that increasing the VM memory sizes do not cause the power to raise. But a larger VM memory size can lead to more energy consumption.

In the next section, we simulate different network environments as a part of our empirical study and interpret the obtained results. We have offered a general analysis method which in fact can be deployed in the real application environments.

3. iMIG MODEL

In this section, we set up a model to describe the relation between the parameters and their corresponding impacts on the migration performance, as described in Section 2. The notation definition is mentioned in Table 1.
speed\_limit. Based on this observation, we obtain the formula $B$. To guarantee a successful migration, workload should be less than the transfer speed.

This is a static model. We can use Equation (2) to predict the $T_{\text{mig}}$ when we know the workload, VM memory, tcp\_buffer and speed\_limit before conducting the migration.

3.2. Dynamic model of $T_{\text{down}}$

$T_{\text{down}}$ is also a key metric to describe the migration performance. But it is difficult to predict this metric before the migration actually starts. So we use a dynamic model to predict $T_{\text{down}}$. We use to denote the dirty data generated in the $i$th round and to represent the remaining dirty data in the $i$th round.

$$\Delta M_i = \omega \times t_i, \Delta M_{ik} = \omega \times t_{i-1} - B \times t_i,$$

where $B = \min\{\text{speed\_limit}, \text{tcp\_buffer} \times 10\}$.

After $N - 1$ rounds, the total dirty data that is left can be calculated as

$$(M - B \times t_1) + (\omega \times t_1 - B \times t_2) + \cdots + (\omega \times t_{N-2} - B \times t_{N-1})$$

$$= M + \omega \times \sum_{i=1}^{N-2} t_i - B \times \sum_{i=1}^{N-1} t_i.$$

Then, we obtain the data that will be sent in the $N$th round:

$$M + \omega \times \sum_{i=1}^{N-2} t_i - B \times \sum_{i=1}^{N-1} t_i = M + (\omega - B) \times \sum_{i=1}^{N-1} t_i.$$

The transition condition that turns the migration into the stop-and-copy stage is the expected downtime calculated in the $(N - 1)$th round which is less than the max downtime.

$$t_N = \frac{M + (\omega - B) \times \sum_{i=1}^{n-1} t_i}{B} \leq T_{\text{maxdowtime}}. \quad (3)$$

We can use Equation (3) to predict the real $T_{\text{down}}$ during the migration. We calculate the theoretical values of the two models correspondingly. But there’s still one problem: though our procedure allocates 20 MB/s, the rate of the dirty data in real migration, is not as high as 20 MB/s, since some new dirtied pages may overlap the pages that were not transferred. So we need to revise the value of the workload during the computation. Combining the results of many of our experiments, we add a coefficient and set its value to 0.1.

As we can see from Fig. 28, our model predicts the $T_{\text{mig}}$ effectively. We also calculate $T_{\text{down}}$ by model 2, and realized that the model generally gives a higher value than the experimental results shown in Fig. 29. This is because the dirty pages generation rate may vary violently during the migration and it is difficult to predict in (3), thus causing the variation in $T_{\text{down}}$.

We carefully build a static and a dynamic model, respectively, to analyze the migration process in different perspectives. Our model provides a clear interpretation of the way of analyzing the relationships between the configuration parameters, and also the strategies of predicting the migration time, which is an important index to evaluate the migration performance.

4. IMPLEMENTATION OF IMIG

In this section, we introduce the specific implementation of iMIG.
4.1. A Bug in QEMU/KVM

The network environment (Table 5) is the Gigabit network (1000 MB/s), so the network speed should be \( \sim 100 \) MB/s. However, the result shows that the real-time network bandwidth computed by the QEMU/KVM is 1000 MB/s, which is obviously wrong. This causes a wrong prediction of the expected downtime for the rest of the migration, leading to an inaccurate migration phase transmission, and ultimately makes the actual downtime longer than the max downtime.

In Table 6, bytes stands for the mean value of the data transferred in each iteration and iterate count represents the number of times that the iterate-copy occurs, \( t \) indicates the mean time period of every round, downtime implies the real downtime and the migration time describes the total migration time in real environments.

From our observations, we draw the following conclusions:

\[
\text{Total\_Bytes\_Transferred} = \text{bytes} \times \text{iterate\_count} = 1528MB \approx \text{mem}, \quad (4)
\]

\[
\text{Total\_Migration\_Time} = t \times \text{iterate\_count} = 1s \ll \text{migration\_time}. \quad (5)
\]

The sum of the data transferred is close to the VM memory size, and the total migration time is calculated by multiplying the time interval of each iteration with the number of iterations, which is far less than the actual time. Thus, the wrong calculation of \( t \) leads to incorrect throughput.

In QEMU/KVM, the function \texttt{ram\_save\_live} is called by command \texttt{qemu\_savevm\_state\_iterate}, and the latter is repeatedly invoked to realize the iterating transmission. So we can acquire the time, before entering the function \texttt{qemu\_savevm\_state\_iterate} and thus the time interval between two iterations is computed. This can be used to calculate the throughput and the procedure is shown in Fig. 30.

\[
\text{Total\_Migration\_Time}^2 = t \times \text{iterate\_count} = 46570\text{ ms} \approx \text{migration\_time}. \quad (6)
\]

### TABLE 6. Results before bandwidth revise.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Iterate count</th>
<th>T</th>
<th>Downtime</th>
<th>Migration time</th>
</tr>
</thead>
<tbody>
<tr>
<td>581035</td>
<td>2757</td>
<td>395 ( \mu )s</td>
<td>315 ms</td>
<td>46550 ms</td>
</tr>
</tbody>
</table>

After the modification, the experiment is repeated again and the obtained results are shown in Table 7. And the total migration time calculated this time is close to the actual value.

4.2. The optimization of existing migration

Computation of wrong bandwidth later affects the value of the expected migration time. And this causes the VM to stop the migration in undesirable situations. Thus, leading to migration failures and in turn brings great loss to the users and the service providers. After our modifications, we obtain a more accurate value of the expected migration time. As a result, we further obtain a more accurate downtime value, by which, reducing the migration failures.

We have identified the prediction method used in the current QEMU/KVM mechanism to be less effective. In this section, we use a simplified method depending on the current circumstances, to predict the metrics more accurately. The constraints we added are listed as follows:

(i) The number of iterations.

We set a threshold level to the number of iterations. Once the \textit{iterate\_count} reaches this threshold, the migration turns into the stop-and-copy phase. This approach proves to promote the success ratio by compromising the migration convergence.

(ii) Migration Time \( T_{mig} \) vs \( T'_{mig} \)

The sum of the VM memory and the newly generated data of each iteration is the total amount of data that needs to be transferred. We estimate the overall time denoted by \( T'_{mig} \). Meanwhile, we record the time of each iteration to calculate the throughput. With the obtained data, the real migration time denoted by \( T_{mig} \), is computed. If \( T_{mig} \) exceeds \( T'_{mig} \), the whole migration might take a long time. In this condition, we appropriately increase the \textit{Downtime} and turn the migration into the stop-and-copy phase during the early stages of migration in order to ensure that \( T_{mig} \) is not too long. In this way, we adjust

### TABLE 7. Results after bandwidth revise.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Iterate count</th>
<th>T</th>
<th>Downtime</th>
<th>Migration time</th>
</tr>
</thead>
<tbody>
<tr>
<td>581035</td>
<td>2760</td>
<td>16874 ( \mu )s</td>
<td>32 ms</td>
<td>46600 ms</td>
</tr>
</tbody>
</table>

FIGURE 30. New time calculation method.
the $T_{down}$ dynamically according to the current circumstances, thus $T_{down}$ is limited to a minimum value lesser than $T_{mig}$.

(iii) Data generating and transferring

When the rate of data generation (denoted by data_rate) is more than the speed_limit, the counter (denoted by over_count) is increased by one count. The expected number of remaining iterations is denoted by remain_count. If

$$\frac{over\_count}{remain\_count} \geq \text{percent\_border}(\text{threshold}),$$

the migration turns into the stop-and-copy phase.

(iv) Optimized calculation of expected downtime

During our experiments, we found that once the max downtime has been assigned, KVM strictly adopts this value to determine whether to enter the stop-and-copy phase or not, even though the expected downtime is very close to the max downtime. This leads to a continuous iteration, thus extending the migration time. However, it is not essential to repeat the iterations, unless to extend the migration time, shown in Fig. 1.

To solve this problem, we turn the migration into the stop-and-copy phase, once either the expected downtime gets closer to the max downtime or the downtime converges to a certain value. These two scenarios imply the presence steady workloads that do not decrease with time.

(v) Optimization of bandwidth

During the migration, we calculate the throughput and compare it with the speed_limit. If the mean throughput during the migration is far less than the speed_limit, we then compare the speed_limit with the tcp_buffer. If the restrictions on the tcp_buffer reflect on poor throughput, the tcp_buffer is increased in order to complete the migration process.

4.3. Experiment results

From Tables 8 and 9, it is evident that with minor modifications of the downtime, the migration time of iMIG has reduced considerably, thus proving the effectiveness of the iMIG.

From Figs 31 and 32, both the expected downtime and the iterate count, are optimized with iMIG.

We know that the migration lasts for a very long time when the network environment is tough, which in turn causes excess energy consumption. After our optimizations, it is clear from Figs 33 and 34 that iMIG reduces as much as half of the energy consumed in similar network conditions. Our optimization techniques can be deployed in real cloud environments and we plan to extend our study by implementing our techniques in practise with the cloud service providers, in the future.

<table>
<thead>
<tr>
<th>TABLE 8. Experiment conditions of iMIG.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM memory</td>
</tr>
<tr>
<td>2 GB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE 9. Experiment results of iMIG.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Migration time (ms)</td>
</tr>
<tr>
<td>Original QEMU/KVM</td>
</tr>
</tbody>
</table>

FIGURE 31. Expected downtime of iMIG.

FIGURE 32. Iterate count of iMIG.
5. RELATED WORKS

Live migration optimization. Clark et al. [6] proposed the pre-copy live migration on Xen platform. In this study, they first transfer the memory pages iteratively, and then pause the source VM in the final phase in order to send the remaining pages, then resuming the execution. However, in some scenarios with a low bandwidth or a heavy workload, the pre-copy phase might fail, since more number of dirty pages requires transmission. Hines et al. [7] proposed the post-copy live migration. They first transfer the VM processor state and then resuming the VM in the target host, only then they begin to copy the VM memory contents. This strategy of post-copy is advantageous as it ensures the completion of the migration process with a few number of page transfers, thus improving the performance. Liu et al. [8] introduced the ReVirt [19] migration, based on a full system trace and replay. They transfer the execution trace logs from the source host to the target host and then replaying them on the target host. This approach reduces both the migration downtime and the network bandwidth consumption. To enhance the usability of their HiTrust service, Li et al. [20] proposed an adaptive trust negotiation strategy which gave us lots of inspirations. Lei et al. [21] presented a one-to-many migration method called VMScatter. It can simultaneously migrate VMs from one to many other hosts with a shorter migration time. Nguyen et al. [22] introduced a strategy to limit the number of slots occupied during the migration process in a data center. Though a lot of research works have been carried out on live migration, the pre-copy is one schema that is most widely used owing to its ease functionalities.

Performance evaluation. Jin et al. [13] improved the performance of the live VM migration by an adaptive memory compression. This approach can decrease the amount of data transferred and balance the performance and the cost of VM migration. The experiments demonstrate that it can reduce the downtime, migration time and the total data transferred. Fereydoun et al. [23] proposed a method for memory transfer. They decreased the downtime by using the probability density function selection of the memory pages. Jin et al. [24] presented a method by using CPU scheduling; the application downtime can be reduced up to 88%, with an acceptable overhead. Song et al. [25] introduced a parallelizing live migration in which, they evaluated their method on Xen and KVM and the results show that their approach can accelerate the live VM migration and decrease the downtime. Sallam et al. [26] proposed a multi-objective migration policy which can direct the progress. And their experiments show that their policy can control the system performance.

A number of studies [23, 24, 27] have been proposed to improve the performance of the live VM migration. But to the best of our knowledge, they all focus on reducing the amount of data transferred in order to improve the migration metrics and none of them has ever focused on the migration environments. We consider the parameters effecting the migration itself to predict the time, well before the migration actually starts. We also optimize the existing KVM live migration by dynamically changing these parameters to adapt the environments during the migration.

Energy-saving computing. Jin et al. [28] investigated how virtualization influences the energy usage in servers under different workload. Li et al. [2] presented an energy-aware algorithm to enable application live placement dynamically. Liu et al. [29] proposed an energy model of the VM migration and they found that the energy consumption is linearly related to the network traffic. Li et al. [30] designed an architecture called CyberGuarder and measured the energy-consumption overhead with four benchmark applications. Yang et al. [31] analyzed the negative impact of the performance interference on energy efficiency after the VM co-allocation.
6. CONCLUSIONS

Live migration of VM is now widely used in the data centers of the cloud infrastructures. Still defects are witnessed in the live migration such as the lack of adaptive ability to make way for the green computing environments. To solve this problem, we designed iMIG to predict the migration time and adapted the dynamic mechanism to improve the migration performance, and to reduce the energy consumption. To achieve this, we analyzed the parameters that affect the performance of live migration and energy consumption by conducting compressive experiments. We calculated the impacts of speed limit, TCP_Buffer size, max downtime, VM memory, and workload. And, we also developed an analytical model to characterize the impacts of all the parameters on the live migration performance. During our experiments, we found that the bandwidth calculation in QEMU/KVM live migration is not accurate and to overcome this issue, we adopted a few modifications in the time record mechanism. Finally, we implemented a self-adaptive approach to optimize the existing KVM live migration mechanism. We proved the efficiency of our algorithm in improving the migration performance, supported by our experiment results. In the future, we plan to investigate the measuring techniques of the floating workload during the migration process to achieve accurate predictions of the migration time and the downtime.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their valuable comments and helps in improving this paper.

FUNDING

This work is supported by the 973 Program (No. 2011CB302602), NSFC Programs (Nos. 91118008, 61202424), China MOST grant (No. 2012BAH46B04), 863 project (No. 2013AA01A213), New Century Excellent Talents in University 2010 and SKLSDE-2012ZX-21.

REFERENCES


