future internet

MDPI

Article

A Novel Hybrid-Copy Algorithm for Live Migration of Virtual Machine

Zhou Lei, Exiong Sun *, Shengbo Chen, Jiang Wu and Wenfeng Shen

School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China; leiz@shu.edu.cn (Z.L.); schen@shu.edu.cn (S.C.); wj1994@i.shu.edu.cn (J.W.); wfshen@mail.shu.edu.cn (W.S.)
* Correspondence: sunexiong@163.com; Tel.: +86-137-322-52150

Academic Editor: Sabah Mohammed
Received: 14 June 2017; Accepted: 13 July 2017; Published: 18 July 2017

Abstract: Live migration of virtual machines is an important approach for dynamic resource scheduling in cloud environment. The hybrid-copy algorithm is an excellent algorithm that combines the pre-copy algorithm with the post-copy algorithm to remedy the defects of the pre-copy algorithm and the post-copy algorithm. Currently, the hybrid-copy algorithm only copies all memory pages once in advance. In a write-intensive workload, copy memory pages once may be enough. However, more iterative copy rounds can significantly reduce the page faults in a read-intensive workload. In this paper, we propose a new parameter to decide the appropriate time to stop the iterative copy phase based on real-time situation. We use a Markov model to forecast the memory access pattern. Based on the predicted results and the analysis of the actual situation, the memory page transfer order would be adjusted to reduce the invalid transfers. The novel hybrid-copy algorithm is implemented on the Xen platform. The experimental results demonstrate that our mechanism has good performance both on read-intensive workloads and write-intensive workloads.

Keywords: live migration; virtual machine; hybrid-copy; Xen

1. Introduction

Cloud computing [1] has become one of the most dominant features in the field of computing. In cloud environments, dynamic resource scheduling [2] is a key problem. The live migration [3] of virtual machine is an important approach for dynamic resource scheduling in cloud environments. With the help of live migration, virtual machine (VM) can be transferred from a host to another host while the services provided by the VM are still available. It is useful in several of scenarios, such as load balancing [4], online maintenance [5], fault tolerance [6], and energy conservation [7].

The key to VM migration is how to handle the resources associated with the migrating VM. The main resources related to VM migration are as follows: memory data, network connecting, CPU state, virtual devices, and storage. In practice, the storage of a VM is always stored in a network-attached storage device (NAS) [8]. The NAS mechanism can avoid the migration of storage because the NAS is accessible to all hosts in the datacenter. In this situation, memory data becomes the main transferred data in the migration process.

There are many approaches to migrate a VM from a host to another host. However, downtime and total migration time should be considered when the migrated VM is in working condition [3]. On the one hand, we need to minimize the downtime in which period the service is totally unavailable. On the other hand, the source host may be shut down at any time, and the performance of the VM is seriously affected during the migration process. So, the total migration time should be shortened. Many live migration algorithms have been proposed to minimize both downtime and total migration time. In terms of the memory data transfer order, live migration algorithms can be classified into three

categories: the pre-copy algorithm [3,9–12], the post-copy algorithm [13–15], and the hybrid-copy algorithm [16–18].

The hybrid-copy algorithm is an excellent algorithm that combine the pre-copy algorithm with the post-copy algorithm to remedy the defects of the pre-copy and post-copy algorithms. The hybrid-copy algorithm consists of two phases: the pre-copy phase and the post-copy phase. In the pre-copy phase, all memory pages are transferred from the source to the destination in advance. Therefore, the page faults occurring in the post-copy phase can be effectively reduced. Currently, the hybrid-copy algorithm copies all memory pages only once in advance. In a write-intensive workload, copying memory pages once may be enough. However, more copy behavior can significantly reduce the page faults in a read-intensive workload. The page faults always incur a significant performance loss [13–15]; sometimes page faults can even cause the services in migrated VM to be totally unavailable.

In this paper, we propose a novel hybrid-copy algorithm. The main target is to improve the performance of hybrid-copy algorithm by reducing the number of page faults, while keeping the migration time at the same level. A new parameter named the switched decision factor (SDF) is proposed to decide the moment to switch from the pre-copy phase to the post-copy phase in the hybrid-copy algorithm. It helps reduce the number of page faults and minimize the downtime caused by page faults in the migration process. A Markov model is used in our novel hybrid-copy algorithm to reduce invalid transfers. This ensures that the migration time will not be increased.

We implemented the novel hybrid-copy algorithm based on Xen virtualization software and ran experiments on a write-intensive workload and a read-intensive workload. Through extensive evaluations, we demonstrate that the novel hybrid-copy algorithm can significantly reduce page faults in read-intensive workloads while achieving the same level of total migration time. In write-intensive workloads, we also keep the good performance of original hybrid-copy algorithm.

In this paper, we analyze the memory access pattern of different workloads and adopt a Markov model to predict the memory page accessed order. We propose a new parameter to decide the opportune moment when the pre-copy algorithm stopped and the post-copy algorithm kicked. As a result, we can get great performance in both read-intensive workloads and write-intensive workloads. We tested our work with two workloads. We successfully demonstrate the novel hybrid-copy algorithm can get better performance with kinds of workloads.

The rest of the paper consists of Section 2, which analyzes the pre-copy algorithm, post-algorithm and the hybrid-algorithm, Section 3, which introduces the prototype of the novel hybrid-copy algorithm, including how to use the Markov Model and how the new parameter works, and the evaluation is shown in Section 4. Related works are introduced in Section 5, and the summary and the future works follows in Section 6.

2. Background

Many live migration algorithms have been proposed. In terms of the memory data transformed order, live migration algorithms always be classified into three categories:

2.1. Pre-Copy

The pre-copy algorithm is the most popular algorithm in VM live migration domain. The core thought of the pre-copy algorithm is to minimize the downtime in the migration process through maximizing the memory mirroring synchronization between the source host and the destination host. Figure 1 provides an overview of the pre-copy algorithm architecture. The pre-copy algorithm can be divided into the following two phases:

(1) Iteration phase: In this phase, memory mirroring synchronization between the source host and the destination host will be maximized by iteratively copying the memory data from the source to the destination. In this phase, the source VM still works, so part of memory pages would be modified during the previous transmission turn. These memory pages should be synchronized

until the number of the remaining memory pages is less than a threshold or the iteration turn is greater than an iterative threshold.

(2) Stop-and-copy phase: In this phase, the source VM will be stopped and the remaining pages will be copied to the destination VM. Then the new VM in the destination host is started.

The pre-copy algorithm can shorten the downtime significantly in case of a read-intensive VM. However, if the VM has write-intensive workloads, the speed of memory write may be faster than the network transmission speed. As a result, the iteratively copy behavior would just waste bandwidth resources and overrun the total migration time.

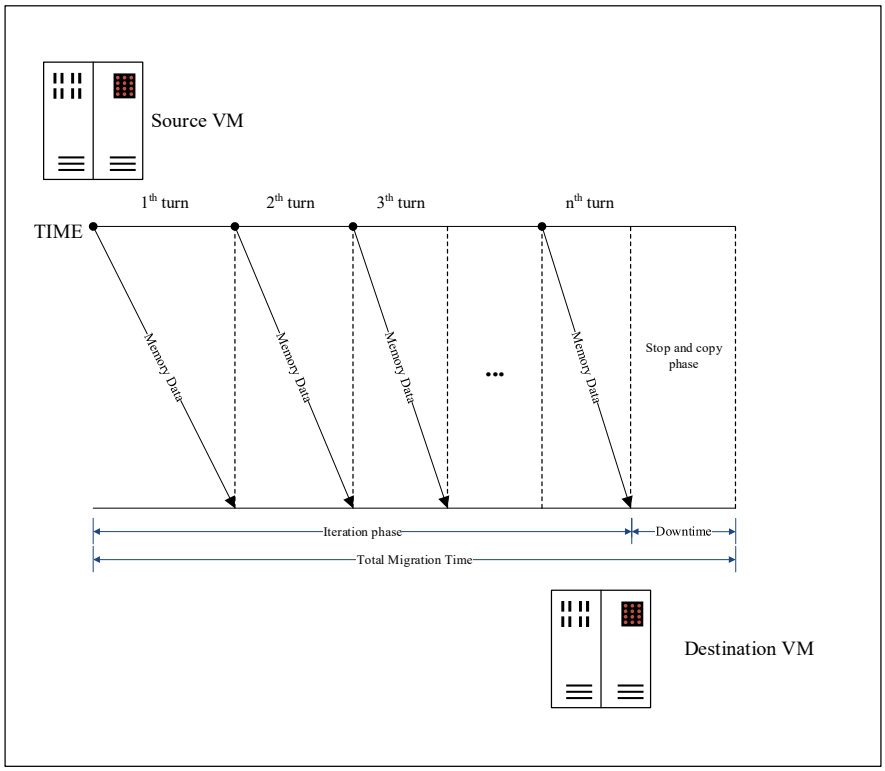

Figure 1. Overview of pre-copy algorithm.

2.2. Post-Copy

In contrary to the pre-copy algorithm, the post-copy algorithm transfers the memory data to the destination after the new VM in the destination begins. The first step of the post-copy algorithm is to suspend the VM at the source. Then, the VCPUs, device states, and some kernel data are transferred to the destination. Then, the new VM resumed at the destination. Last, memory data at the source are transferred to the destination mainly by two ways: on-demand paging and active pushing.

On-demand paging is the simplest way to synchronize the memory data. Once the VM starts at the destination and visits the memory page, which is not synchronized, it will result in page faults. The VM will then get the corresponding pages from the source. However, the network delay will cause a seriously degradation of the VM performance. From the user's perspective, it feels like the server crashed when the page faults occur. So, we should try to avoid page faults resulting in the migration process. Besides, if there is no other mechanism, the migration process can last a long time because some part memory pages of the VM may not have been visited in a long term.

Active pushing is one way to reduce the duration of residual dependencies on the source node. The source will remain until the migration process is complete. The source VM will proactively push the remaining memory pages to the destination.

In fact, on-demand paging and active pushing always work together in the post-copy algorithm. The post-copy algorithm ensures that all memory data are transferred only once, so the post-copy strategy has minimal total migration time closer to its equivalent time achieved by non-live VM migration. However, the disadvantages of post-copy algorithm are obvious: too many page faults may create network traffic, and the performance of the VM is seriously affected.

2.3. Hybrid-Copy

The hybrid-copy algorithm works by running the pre-copy algorithm just once as the first step of the migration. In this phase, the source VM continues providing service to users while all the memory data are copied to the destination. The VM is then suspended and its processor state is copied to the destination without the remaining dirty memory pages. Further, the VM resumes at the destination immediately, and the post-copy algorithm kicks. The rest of the memory pages will be synchronized in the post-copy phase.

Page faults in post-copy phase always incur a significant performance loss; sometimes page faults can even cause the services in migrated VM to be totally unavailable. Hybrid-copy algorithm can avoid many page faults because of the pre-copy algorithm. Further, as with the post-copy mechanism, it also helps solve the trouble of the pre-copy algorithm in write-intensive workloads. In the original hybrid-copy algorithm, the pre-copy algorithm only works for a single round. The page faults occurring in the post-copy phase can be further reduced if more copy iterations are executed. However, a predetermined number of rounds cannot be satisfied with various workloads. If the network is in good condition and the VM runs a read-intensive workload, more iterative copy round should be executed, reducing the page faults further. On the contrary, if the network condition is poor or the VM runs a write-intensive workload, less iterative copy behavior should be performed to save the network resources and the migration time. In this paper, we proposed a new parameter to determine the right moment to stop the iterative copy phase based on the real situation.

3. Novel Hybrid-Copy Algorithm

Here, we present the design of the novel hybrid-copy algorithm. The novel hybrid-copy algorithm needs to forecast the memory write pattern in each round of iteration in the pre-copy phase. Only a half of dirty pages that are least likely to be modified in the transmission process are transferred in each iterative copy round, instead of transmitting all dirty pages. Part of any duplicate transmissions can therefore be avoided. After each round of iterative copy, the SDF value is calculated to determine the proper time to stop the iteration phase. The post-copy algorithm will then continue running until the VM is transferred completely. In the post-copy phase, the Markov Model will still forecast the memory access pattern, and the source will actively push the remain dirty memory pages to the destination based on the predicted result to further reduce the number of page faults.

3.1. Markov Model of Memory Write

Assume that the set $M = \{m_1, m_2, \ldots, m_n\}$ represents all memory pages in the source VM. The set $M' = \{m'_1, m'_2, \ldots, m'_n\}$ is a set of memory pages in the destination VM. We need to ensure that $M = M'$ after the live migration of the VM.

Definition 1 (Dirty Page). *A dirty page at the source is a memory page whose state is different from the destination. These pages should be copied to the destination.*

All memory pages in the set M are marked as dirty pages in the beginning. Assume that m_1 would be modified in i^{th} round of the iteration phase. If the m_1 page was synchronized before the i^{th} round of the iteration phase, this page should be synchronized again after i^{th} iterative copy round. If it can be forecasted when the memory page would be modified, lots of time and bandwidth resources will be saved. The Markov model is a classical forecasting model: Joseph et al. [19] have proven that

the Markov model achieves a good result in memory prefetching. Therefore, we chose the Markov model as the forecasting model in our novel hybrid-copy algorithm.

In the observation period, let all memory pages in the source be the input. We need to build a forecasting model based on the analysis of the input data to forecast which pages are going to be dirtied in the next round. Assume a VM possessing 1 GB memory with a 4-KB page size. Such a VM will contain 262,144 pages. In a real case, there may be even larger memory in a VM. It is a heavy burden to analysis such a large data set for the live migration task.

In the pre-copy algorithm implemented by Clark et al. [3], a concept of the "writable working set" (WWS) is proposed. It is a set of the global frequently modified pages in the iteration phase. We can use the WWS as the input instead of all the memory pages. Assume $W = \{w_1, w_2, \ldots, w_t\}$ represents the memory pages that belong to the WWS.

To simplify the question, we use a dirty memory page to describe a memory write state. For example, if w_1 is modified, then w_1 is used to describe present state. There will arise to kinds of states during the observation period. The VM is configured with 1-GB RAM and two VCPUs. Various workloads are running in different times. We track dirty pages through the shadow page every 50 ms. Assume that the w_1 and w_2 were modified during the first 50 ms, the w_2, w_3, and w_4 are modified in the next 50 ms, and the w_1, w_4 and w_5 are changed during the third 50 ms. We believe that every state that occurred during the second 50 ms are associated with the states appeared during the first 50 ms. Similarly, the states happed in the third 50 ms are related to the states that arise during the second 50 ms. If there are only three observation periods, we can calculate the state transition probabilities. The conversion relations between all states are shown in Figure 2. In this example, different states are described by the dirtied memory pages. Each transition from node X to node Y in the diagram is assigned a weight representing the transition probability from X to Y. For example, if the w_1 is modified, there is a 50% possibility that the w_2 will be changed in the next 50 ms, and both w_3 and w_4 have a 25% probability to be changed in the next 50 ms. More concretely, we can use a state transition matrix to represent the Markov transition diagram. The observation period lasts 8 s at least, and we can get more than 160 groups of states transitions like these. We can then calculate the state transition matrix P_{t*t} based on these state transitions [19].

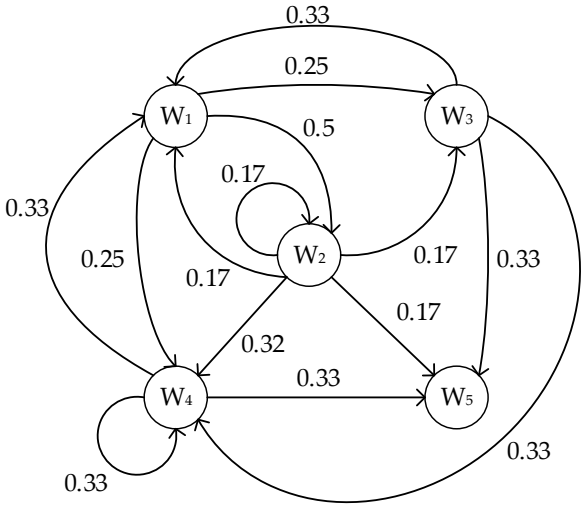

Figure 2. The Markov model representing the previous dirty pages via transition probabilities.

3.2. Switched Decision Factor(SDF)

Before we introduce the SDF, let us discuss the flaw of the current hybrid-copy algorithm. It is the key thought of the hybrid-copy algorithm to combine the pre-copy algorithm and the post-copy algorithm to remedy the defects of the pre-copy and post-copy algorithms. In the current hybrid-copy

algorithm, all memory pages are transmitted to the destination only once. Then, the post-copy algorithm starts working. All memory pages are copied once so that the number of page faults are notably reduced, compared with the pure post-copy algorithm. However, if the iterative process of the pre-copy algorithm increases, the number of page faults and the downtime caused by the page faults will be reduced as well. Thus, the best time to switch the pre-copy algorithm to the post-copy algorithm is a key issue in hybrid-copy algorithm.

Here, we propose SDF to determine when to switch the pre-copy phase to the post-copy phase. Some definitions and symbols are first introduced in Table 1 to facilitate discussion.

Definition 2 (Invalid Transfer). *An Invalid transfer means a memory page at the source is dirtied again, after this page has been transferred to the destination at least once.*

Table 1. Symbols and definitions.

Symbol	Definition
N	The total number of the VM's memory pages
$S(n)$	The number of the transmitted memory pages in n^{th} round
$R(n)$	The total number of the transmitted memory pages after n^{th} round
$T(n)$	The number of invalid transfers in n^{th} round
$V_1(n)$	The total number of invalid transfers after n^{th} transfer
$V_2(n)$	The number of the dirty pages after n^{th} transfer

If a transmitted memory page at source is dirtied again, it should be retransmitted. This means that all transmissions of this memory page before this copy round did nothing but waste resources and time. Therefore, the number of the invalid transfers should be as small as possible.

At the beginning, there is no any transfer between source and destination, so the value of n in Table 1 is 0 at first. We therefore arrive at the following conclusion:

$$\mathrm{S}(0) = 0,\ T(0) = 0,\ V_1(0) = 0,\ V_2(0) = N \tag{1}$$

As shown in Table 1, $V_1(n)$ represents the total number of invalid transfers after n^{th} transfer, while $V_2(n)$ represents the number of the dirty pages after n^{th} transfer. It is obvious that $V_1(n)$ is related to the wasted bandwidth resources in the pre-copy phase, and $V_2(n)$ has ties with the number of page faults and the downtime caused by page faults. Therefore, we introduce the following function to evaluate the state of migration process:

$$\mathrm{F}(n) = \alpha \times V_1(n) + (1 - \alpha) \times V_2(n) \tag{2}$$

Here, the $0 \leq \alpha \leq 1$. If $\alpha = 0$, only the number of dirty pages needs to be considered, and the pure pre-copy algorithm should be adopted. In this situation, we get the smallest downtime. However, lots of bandwidth resources may be wasted and the migration process may last a long time. Otherwise, if $\alpha = 1$, then $(1 - \ \alpha) = 0$. This means that we should ensure that the number of invalid transfers is as small as possible. In this case, the pure post-copy algorithm is a good choice. We can guarantee that all memory pages are transferred only once and that there is no any invalid transfer. Numerous page faults always caused network traffic and seriously affect the user experience.

According to the definition of the symbols in the Table 1, Equations (3) and (4) can be derived as,

$$\mathrm{T}(n) = S(n) - (V_2(n-1) - V_2(n)) \tag{3}$$

$$V_1(n) = \sum_{i=1}^{n} T(i),\ n > 0 \tag{4}$$

Based on Formulas (1) and (2), we can easily get Formula (5) as follow:

$$\mathrm{F}(0) = (1 - \alpha) \times N \tag{5}$$

If $n > 0$, then we can get Formula (6) through Formulas (2)–(4),

$$\begin{aligned} \mathrm{F}(n) &= \alpha \times V_1(n) + (1 - \alpha) \times V_2(n) \\ &= \alpha \times \textstyle\sum_{i=1}^{n} T(i) + (1 - \alpha) \times V_2(n) \\ &= \alpha \times \textstyle\sum_{i=1}^{n} (S(n) - (V_2(n-1) - V_2(n))) + (1 - \alpha) \times V_2(n) \\ &= \alpha \times \textstyle\sum_{i=1}^{n} S(n) - \alpha \times V_2(0) + \alpha \times V_2(n) + (1 - \alpha) \times V_2(n) \\ &= \alpha \times \textstyle\sum_{i=1}^{n} S(n) - \alpha \times N + V_2(n) \\ &= \alpha \times R(n) + V_2(n) - \alpha \times N \end{aligned} \tag{6}$$

We hope the value of F(n) is as small as possible to save bandwidth resources and reduce the number of page faults. Under ideal condition, the $V_2(n)$ should tend to 0 and the $V_1(n)$ would not increase. As a result, the value of F(n) would tend to 0 with the increase of n. However, in write-intensive workloads, a large number of invalid transmissions are produced during the VM migration process. Therefore, the value of F(n) may actually increase rather than decrease. At this point, the iteration phase should be stopped, and the pre-copy algorithm should be switched to the post-copy algorithm because more iterative copies are just wasting network resources and time. If the n^{th} round iterative copy has a positive effect, the F(n) should be smaller than F(n-1). Then, Formula (7) can be derived as follow:

$$\begin{gathered} \mathrm{F}(n) - \mathrm{F}(n-1) < 0 \\ \alpha \times (R(n) - R(n-1)) + (V_2(n) - V_2(n-1)) < 0 \\ \alpha \times \mathrm{S}(n) + (V_2(n) - V_2(n-1)) < 0 \\ \tfrac{V_2(n-1) - V_2(n)}{S(n)} < \alpha \end{gathered} \tag{7}$$

We define $(V_2(n-1) - V_2(n))/\mathrm{S}(n)$ as the switched decision factor (SDF). SDF(n) represents the value of SDF in nth round. If the SDF(n) is smaller than α, the iteration phase will be stopped, and the post-copy algorithm kicks. According to formula (2), α represents the tradeoff between the number of page faults and bandwidth resource consumption. As α increases, a few iterative copies are executed.

3.3. Novel Hybrid-Copy Algorithm

With the Markov model and the SDF described above, the complete Novel Hybrid-Copy Algorithm is implemented here. Algorithm 1 shows the framework of the complete Novel Hybrid-Copy Algorithm, and Algorithm 2 shows the function that is used to determine the send list using the Markov model to forecast the memory write order.

The input parameters in Algorithm 1 consists of three variables. The "memory_size" is the total number of the memory pages in the VM. The "dirty_threshold" is the threshold of the number of the remaining dirty pages. If the number of the dirty pages is smaller than the threshold, the iterative copy phase should be stopped in advance because the dirty pages are running out. The last variable "a" represents the α. The function "Get_Dirty_List()" is a Xen system call: it returns the dirty pages set. We do the pre-copy phase in the condition-controlled loop. The stopping condition of the process is thus if the value of SDF is greater than α settled in advance or the number of the dirty pages is smaller than the threshold of the dirty page number. The function "Filter_Dirty()" is defined in Algorithm 2. The function "Send_live_dirty()" is a Xen system call to send the memory pages stored in "send_list". And the function "Get_Dirty_Size()" is also a Xen system call, which will return the number of the dirty pages. Last, the pre-copy algorithm is stopped, and the post-copy algorithm is started.

Algorithm 1: Novel Hybrid-Copy

```
Input:
    memory_size, dirty_threshold: Integer
    a: Float
Output:
Variables:
    send_list, dirty_list: Array of Integer
    decision_fac: Float
    pre_dirty_size, dirty_size, send_size: Integer
Instructions:
    decision_fac = 0;
    dirty_size = memory_size;
    pre_dirty_size = memory_size;
    send_list = NULL;
    dirty_list = Get_Dirty_List();
    While decision_fac < a AND dirty_size > dirty_threshold Do
            send_list = NULL;
            send_size = Filter_Dirty(send_list, dirty_list);
            Send_live_dirty(send_list);
            dirty_list = Get_Dirty_List();
            pre_dirty_size = dirty_size;
            dirty_size = Get_Dirty_Size();
            decision_fac = (pre_dirty_size - dirty_size) / send_size;
    END While
    Post-Copy-Algorithm();
END Novel Hybrid-Copy
```

The input parameters in Algorithm 2 consists of two variables. The "send_list" will store the memory pages, which should be transferred in present iteration round. The "dirty_list" stores the dirty pages at present, which is used to forecast dirty pages in the next period. Then, we only transfer the memory pages that would not be modified in a short period. The function "Markov_Forecast()" will forecast the dirty pages most likely occurring in the next round by Markov Model building in the observation period. These memory pages should not be transmitted in next turn.

Algorithm 2: Filter_Dirty

```
Input:
    send_list, dirty_list: Array of Integer
Output:
    send_size: Integer
Variables:
    forecast_list: Array of Integer
Instructions:
    If send_list!= NULL Then
            send_list = NULL;
    End If
    forecast_list = NULL;
    forecast_list = Markov_Forecast(dirty_list);
    send_list = dirty_list - forecast_list;
    send_size = Get_Size(send_list);
END Filter_Dirty
```

4. Experimental Results and Comparative Analyses

In this section, performances of the Novel Hybrid-Copy Algorithm and analyses are presented. The experimental platform consists of three physical hosts, each with four eight-core 2.6GHz Intel Xeon E5-2550 CPU, 32GB DDR3 RAM, and Intel 82,576 gigabit network connections. All VMs used in our experiment are configured to use two VCPUs and 1GB RAM. We use the latest stable version of Xen (version of xen is 4.8.0) as the VM monitor. The guest kernel is Linux 4.9.13 (stable version), and the host kernel is a modified version of Linux 4.9.13 for both the source and the destination. The storage is accessed via iSCSI protocol from the third physical host configured as the NAS. The workloads used in the experiments are as follow:

SPEC VIRT_SC 2013: A benchmark addressing performance evaluation of datacenter servers used in virtualized server consolidation. SPEC VIRT_SC 2013 (v1.1) measures the end-to-end performance of all system components including hardware, the virtualization platform, and the virtualized guest operating system and application software. The benchmark supports hardware virtualization, operating system virtualization, and hardware partitioning schemes. It is a write-intensive workload.

Linux Kernel Compile: A Linux 4.9.13 kernel is compiled in the migrated VM in our second experiment. It is an intensive workload.

4.1. Total Migration Time

Figure 3 shows the total migration time for both the SPEC VIRT_SC 2013 and Linux Kernel Compile. The α is set to 0.3, 0.5, 0.7 and 1.0. We also compare with the original hybrid-copy algorithm: the last column in the Figure 3 represents the results of the original hybrid-copy algorithm. We found that as the α increased, the total migration time decreased in Novel Hybrid-Copy algorithm because additional iterative copies are conducted with the increase of α.

When the α is set as 1.0, the copy operation before the post-copy phase executed only once. Unlike the original hybrid-copy algorithm, it does not transfer all memory pages in the iterative copy phase. Only half of the memory pages selected by Markov Model are transferred in advance. So, less memory pages would be transferred in total. Less migration time should be spent in the Novel Hybrid-Copy algorithm when α is set to 1.0. The results in Figure 3b are broadly in line with our forecasts. However, Figure 3a shows a theoretically inconsistent result because the migration time was not only associated with the number of transferred memory pages. It also related to the real-time network conditions, VM states, and the number of the page faults occurred during the post-copy phase. In fact, because all memory pages are transferred at most twice in the original hybrid-copy algorithm, there is little reduction in total migration time.

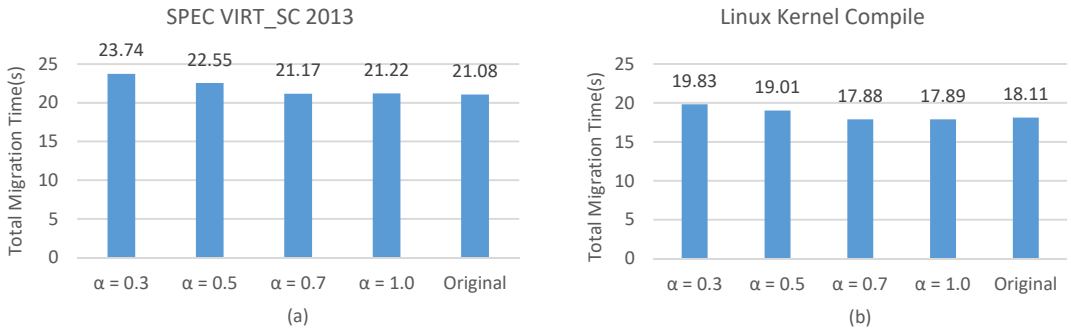

Figure 3. The total migration time of the experiment. (**a**) SPEC VIRT_SC 2013; (**b**) Linux Kernel Compile.

4.2. Total Transferred Pages

As shown in Figure 4, the number of transferred pages is associated with the α value. In the original hybrid-copy algorithm, all memory pages are transferred only once in the pre-copy phase.

We found that a total 297,564 pages are transferred in SPEC VIRT_SC 2013 workload case with the original hybrid-copy algorithm. In our experimental platform, every VM has 262,144 memory pages. This means that there are 35,420 invalid transfers in the migration process.

From Figure 4a,b, we know that if the α is large enough, the novel hybrid-copy algorithm has less transferred pages than original hybrid-copy algorithm because the Markov model avoided parts of the invalid transfers. In our experiments, if the α is set to 0.7, our novel hybrid-copy algorithm transfers a similar number of memory pages to the original hybrid-copy algorithm. If the α is set to 1.0, it always has less total transferred memory pages. If the α is small, we may transfer many more pages than the original hybrid-copy algorithm because more iterative copies are used in the pre-copy phase. The number of transferred pages is positively correlated with the consumption of network resources. Therefore, if the network condition is poor, we need set a large value for α to save more bandwidth resources.

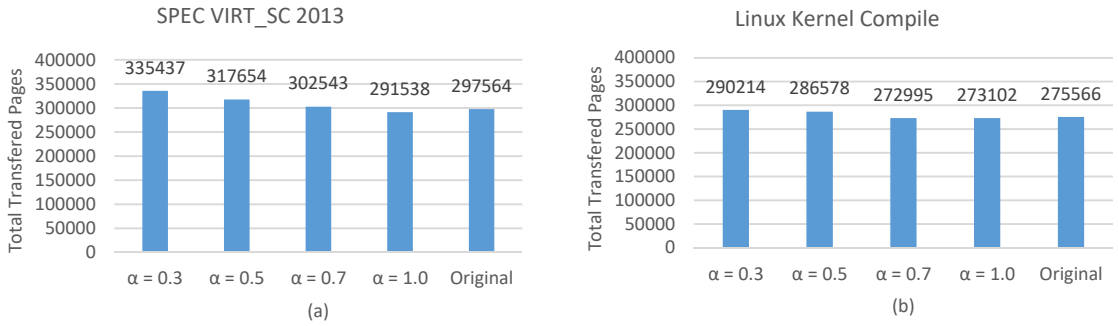

Figure 4. The total number of transferred pages. (**a**) SPEC VIRT_SC 2013; (**b**) Linux Kernel Compile.

4.3. Number of Page Faults

Figure 5a illustrates the number of page faults in the SPEC VIRT_SC 2013 workload, when the α takes a different value. As shown in Figures 3a and 5a, 1515 page faults are reduced when α is set to 0.7 and only extra 0.09 s are spent than the original hybrid-copy algorithm. Less page faults can greatly improve the user experience.

As shown in Figure 5b, when the VM runs a Linux Kernel Compile workload, it is best to set the α as 0.3. The least page faults will be occurred in the post-copy phase because the Linux Kernel Compile is a read-intensive workload compared to the SPEC VIRT_SC 2013 workload. In the SPEC VIRT_SC 2013 workload, even the α is set to 0.3, and there is no significant improvement. However, the total migration time is increased, which adds to the burden of the network. Thus, it is appropriate to set the α to 0.7 in the SPEC VIRT_SC 2013 workload.

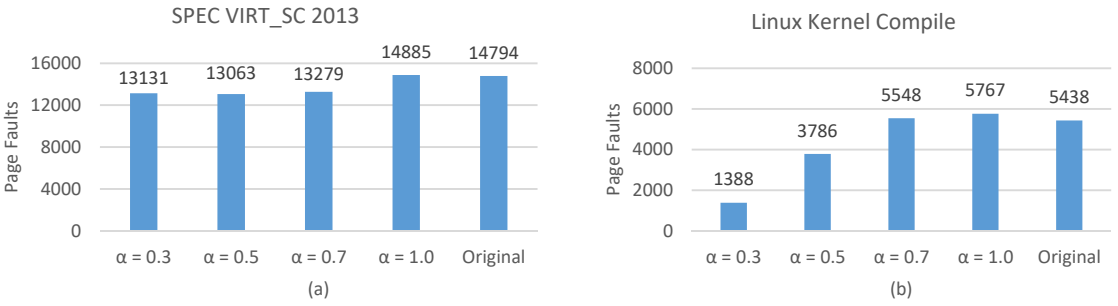

Figure 5. Number of page faults. (**a**) SPEC VIRT_SC 2013; (**b**) Linux Kernel Compile.

4.4. Summary of Experiments

The experiments described above prove that the novel hybrid-copy algorithm can effectively reduce the number of page faults, especially when the migrated VM runs a read-intensive workload. Based on the analysis of the experimental results, we arrive at the conclusion that as the α increases, fewer iterative copy are executed. In this situation, more bandwidth resources can be saved, but the page faults and the downtime caused by page faults may increase. Therefore, if a VM runs a write-intensive workload, the α should be set to a large value. For example, it is appropriate to set the α to 0.7 in SPEC VIRT_SC 2013 workload. In this case, we spent an additional 0.4% live migration time to reduce about 10% number of page faults. On the contrary, a small value of α is a good choice for a read-intensive workload. More iterative copies can significantly reduce the page faults, and only a little more migration time is spent. For instance, in a Linux Kernel Compile workload, when the α is set as 0.3, almost 75% number of page faults are reduced, and only an additional 9.5% of migration time is spent. Page faults always seriously affect the performance of the migrating VM. Sometimes, the services provided by the migrating VM are totally unavailable because of the page faults. Therefore, reducing the number of page faults will significantly improve the user experience.

5. Related Work

Since Clark et al. [3] proposed the live migration of virtual machines, there has been considerable work in the development of VM live migration. Nelson et al. [9] implemented their system in a VMware platform. It is the first system that supported the migration of unmodified applications on an unmodified mainstream Intel x86-based operating system. Memory compression technology [2,20] has been widely used to improve migration performance. In these systems, the compression ratio is very important. A small compression ratio will have little effect, while a larger compression ratio may consume too much computing resource.

Hsu et al. [10] presented an adaptive pre-copy algorithm called MPP, which only transmits memory pages when a predefined threshold is met. It significantly reduces the unnecessary migration of memory pages. Baruchi et al. [11] introduced a method to identify the workload cycles of a VM based on the information to reduce live migration overhead. Ruan et al. [12] proposed a novel data filter mechanism to improve the performance of the pre-copy algorithm in terms of the migration time and the amount of migrated data.

Adaptive pre-paging and dynamic self-ballooning [13] are applied in post-copy algorithm by Hines et al. Adaptive pre-paging technology will forecast the memory access pattern based on the page faults position and the spatial locality principle. The source VM will then adjust the memory pages of the transport order. Abe et al. presented a new mechanism for post-copy algorithm that focuses on recovering the aggregate performance of the VMs being affected. Su et al. [15] improved the traditional post-copy algorithm by eliminating unnecessary remote page faults.

Based on the pre-copy algorithm and the post-copy algorithm, Chen et al. [17] designed a hybrid-copy algorithm. All memory pages in the source are copied to the destination first, then the post-copy method kicks. Many other hybrid-copy algorithm [16,18] are proposed. However, there has been little research done on when to switch from the pre-copy phase to post-copy phase.

Recently, Arif et al. [5] studied live migration over wide area networks. They proposed a MLDO approach to reduce downtime during live migration over wide area networks. Esposito and Cerroni [21] proposed a geometric programming model and an online multi-VM live migration algorithm based on such model. Sun et al. [22] also studied the live migration for multiple VMs and the parallel migration strategy.

6. Conclusions and Future Works

Although scholars have done a lot of research in the field of VM live migration, little research has been done on when to switch from pre-copy phase to post-copy phase in hybrid-copy algorithm

based on real situation. In this paper, a novel hybrid-copy algorithm is proposed. We introduced SDF to decide the best time to switch the pre-copy phase to the post-copy phase. A Markov model is used to forecast the memory access pattern to reduce the number of invalid transfers. The experiments demonstrate that our mechanism has good performance on write-intensive workloads and read-intensive workloads. Compared with the original hybrid-copy algorithm, we can effectively reduce the page faults while achieving the same level of total migration time. Page faults always cause a seriously degradation of the VM performance. Therefore, our algorithm will effectively improve the user experience by reducing the number of the page faults.

In the future, we plan to design a new algorithm to calculate the SDF value automatically so that the SDF value will be adjusted automatically based on the real-time state of the VM. In addition, we can find a better model than the Markov model to further reduce the invalid transfer and save more migration time.

Acknowledgments: This work is partially supported by Shanghai Innovation Action Plan Project under the grant No. 16511101200.

Author Contributions: Exiong Sun and Zhou Lei conceived and designed the experiments; Exiong Sun and Jiang Wu performed the experiments; Exiong Sun wrote the paper. Shengbo Chen and Wenfeng Shen provided technical support and revised the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Armbrust, M.; Fox, A. *Above the Clouds: A Berkeley View of Cloud Computing*; Eecs Department University of California Berkeley: Berkeley, CA, USA, 2010.
2. Jin, H; Deng, L; Wu, S. Live virtual machine migration with adaptive, memory compression. In Proceedings of the IEEE International Conference on CLUSTER Computing and Workshops, New Orleans, LA, USA, 31 August–4 September 2009; pp. 1–10.
3. Clark, C; Fraser, K; Hand, S. Live migration of virtual machines. In Proceedings of the Symposium on Networked Systems Design and Implementation, Berkeley, CA, USA, 2–4 May 2005; DBLP: Trier, Germany, 2005; pp. 273–286.
4. Tiwari, P.K.; Joshi, S. Dynamic Weighted Virtual Machine Live Migration Mechanism to Manages Load Balancing in Cloud Computing. In Proceedings of the IEEE International Conference on Computational Intelligence and Computing Research, Chennai, India, 15–17 December 2016; IEEE: Piscataway, NJ, USA, 2016.
5. Arif, M.; Kiani, A.K.; Qadir, J. Machine learning based optimized live virtual machine migration over WAN links. *Telecommun. Syst.* **2017**, *64*, 1–13. [CrossRef]
6. He, J.; Dong, M.; Ota, K. NetSecCC: A scalable and fault-tolerant architecture for cloud computing security. *Peer-Peer Netw. Appl.* **2014**, *9*, 1–15. [CrossRef]
7. Xu, X.; Wu, J.; Yang, G. Low-power task scheduling algorithm for large-scale cloud data centers. *Syst. Eng. Eletron.* **2013**, *24*, 870–878. [CrossRef]
8. Gibson, G.A.; Van Meter, R. Network attached storage architecture. *Commun. ACM* **2000**, *43*, 37–45. [CrossRef]
9. Nelson, M; Lim, B.H.; Hutchins, G. Fast Transparent Migration for Virtual Machines. In Proceedings of the annual Conference on USENIX Annual Technical Conference, Berkeley, CA, USA, 10–15 April 2005.
10. Hsu, C.H.; Peng, S.J.; Chan, T.Y. An Adaptive Pre-copy Strategy for Virtual Machine Live Migration. In *Internet of Vehicles–Technologies and Services*; Springer International Publishing: Cham, Switzerland, 2014; pp. 396–406.
11. Baruchi, A.; Midorikawa, E.T.; Sato, L.M. Reducing Virtual Machine Live Migration Overhead via Workload Analysis. *IEEE Lat. Am. Trans.* **2015**, *13*, 1178–1186. [CrossRef]
12. Ruan, Y.; Cao, Z.; Cui, Z. Pre-Filter-Copy: Efficient and Self-Adaptive Live Migration of Virtual Machines. *IEEE Syst. J.* **2016**, *10*, 1459–1469. [CrossRef]

13. Hines, M.R.; Gopalan, K. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. Procedings of the International Conference on Virtual Execution Environments, VEE 2009, Washington, DC, USA, 11–13 March 2009; DBLP: Trier, Germany, 2009; pp. 51–60.
14. Abe, Y.; Geambasu, R.; Joshi, K. Urgent Virtual Machine Eviction with Enlightened Post-Copy. In Proceedings of the 12th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, Atlanta, GA, USA, 2–3 April 2016; Volume 51, pp. 51–64.
15. Su, K.; Chen, W.; Li, G. RPFF: A Remote Page-Fault Filter for Post-copy Live Migration. In Proceedings of the IEEE International Conference on Smart City/socialcom/sustaincom, Chengdu, China, 19–21 December 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 938–943.
16. Sahni, S.; Varma, V. A Hybrid Approach to Live Migration of Virtual Machines. In Proceedings of the IEEE International Conference on Cloud Computing in Emerging Markets, Bangalore, India, 11–12 October 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 1–5.
17. Chen, Y.; Huai, J.P.; Chun-Ming, H.U. Live Migration of Virtual Machines Based on Hybrid Memory Copy Approach. *Chin. J. Comput.* **2011**, *34*, 2278–2291. [CrossRef]
18. Jaswal, T; Kaur, K. An Enhanced Hybrid Approach for Reducing Downtime, Cost and Power Consumption of Live VM Migration. In Proceedings of the International Conference on Advances in Information Communication Technology & Computing, Bangalore, India, 12–13 August 2016; ACM: New York, NY, USA, 2016; p. 72.
19. Joseph, D.; Grunwald, D. Prefetching Using Markov Predictors. *IEEE Comput. Soc.* **1999**, *48*, 121–133. [CrossRef]
20. Rd, P.; Hudzia, B.; Tordsson, J. Evaluation of delta compression techniques for efficient live migration of large virtual machines. In Proceedings of the ACM Sigplan/Sigops International Conference on Virtual Execution Environments, Newport Beach, CA, USA, 9–11 March 2011; ACM: New York, NY, USA, 2011; pp. 111–120.
21. Esposito, F.; Cerroni, W. GeoMig: Online Multiple VM Live Migration. In Proceedings of the IEEE International Conference on Cloud Engineering Workshop, Berlin, Germany, 4–8 April 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 48–53.
22. Sun, G.; Liao, D.; Anand, V. A new technique for efficient live migration of multiple virtual machines. *Future Gen. Comput. Syst.* **2016**, *55*, 74–86. [CrossRef]