Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

Efficient online migration mechanism for memory write-intensive virtual machines

Pingping Li[1*] and Jiuxin Cao[1]

Abstract

Online migration of virtual machines (VMs) is indispensable for system maintenance as it helps to achieve several resource management objectives such as load balancing, proactive fault tolerance, green operation, and resource management of data centers. The migration efficiency and reliability are two major challenges in the online migration of memory write-intensive VMs. For example, pre-copy migration transfers a large amount of data and takes a long time to migrate. This study proposes an efficient and reliable adaptive hybrid migration mechanism for memory write-intensive VMs. The mechanism optimizes the data transfer mode of the common migration method and improves the performance of conventional hybrid migration. First, the virtual machine (VM) memory data to be migrated are divided into dynamic and static data based on the bitmap marking method, and the migration efficiency is improved through parallel transmission based on different networks. Second, to accelerate the migration reliability, an iterative convergence factor is proposed to evaluate the current system load state and adaptively calculate the switching time of the migration mode for adaptive hybrid migration based on the convergence factor. Through adaptive hybrid migration can achieve migration completed successfully, shorten the post-copy migration duration, and minimize the impact on the performance of VMs. Finally, this paper implements the system prototype based on a kernel-based virtual machine (KVM), and experiments are performed using multiple memory write-intensive load VMs. The results show that the proposed migration algorithm can significantly improve migration performance and complete migration quickly to solve the pre-copy migration failure problem with a memory write-intensive load. Compared with the traditional hybrid migration with only one round of pre-copy, the proposed migration algorithm reduces the total migration time and transmits data by 23.2% and 26.7%, respectively.

Keywords Virtual machine, Online migration, Parallel migration, Hybrid migration, Network fault tolerance

Introduction

VM online migration is a critical feature in cloud computing systems which refers to the migration of VM memory, virtual central processing unit (vCPU) context, virtual disk, and other devices status data to other hosts over the network [1, 2]. Online migration is critical for dynamic resource deployment, power consumption management, system upgrade and maintenance, load balancing, and system fault tolerance in data centers [3–6]. Since most modern data center deployments are based on Storage Area Network (SAN) or Network Attached Storage (NAS), data on virtual disks do not need to be transferred during migration. Thus, the task of VM migration in shared storage scenarios is to migrate VM memory data, vCPU context, and some device status data. Efficient and fast migration of VM memory data is a vital issue in online migration as most of the migration data are from VM memory data. Pre-copy is the most commonly used VM migration method [7], such as VMware, KVM, and Xen, mostly use the pre-copy mechanism [8, 9]. Pre-copy transmits memory data based on the iterative mechanism, and each iteration transmits the dirty memory pages of the previous round. Iteration

*Correspondence:
Pingping Li
230188114@seu.edu.cn
[1] School Of Cyber Science and Engineering, Southeast University, Nanjing, China

stops when pre-copy achieves a short downtime; then, the VM is temporarily suspended to migrate the remaining data.

With the development of information technology, increased memory write-intensive applications are available in data centers. However, online VMs migration with memory write-intensive loads are a difficult problem for existing cloud platforms. Since the loads update memory very quickly, numerous memory dirty pages are repeatedly written during migration, which needs to be retransmitted many times during pre-copy. Therefore, a large amount of data is transferred, resulting in a long total migration time. This inefficient transmission greatly reduces the efficiency of dynamic resource allocation, reduces system maintenance in the data center, causes the waste of resources, and increases energy consumption [10]. Additionally, when the network bandwidth is limited, and the memory page update rate is greater than the network transmits rate, pre-copy will not actively converge to the stop-copy phase and will lead to uncompleted migration within the expected downtime [11]. The common strategy to solve these problems is to forcibly suspend the VM or reduce the working frequency of the vCPU to reduce the memory update frequency [7, 12]. However, this approach will greatly prolong the VM downtime, degrade VM performance, and result in service-level agreement violation (SLAv). Without these enforcement measures, pre-copy migration will fail because iteration will be prevented from converging. The post-copy mechanism [13] can optimize the pre-copy migration problem. However, its reliability is poor. If a network fault occurs during migration, VMs restoration will be impossible. In addition, the VM performance will be seriously affected owing to page fault interrupt problems. A hybrid migration mechanism [14, 15] can reduce the impact of post-copy inherent disadvantages to a certain extent, but it still cannot avoid the existing problems. Moreover, the conventional hybrid migration with a fixed switching threshold ignores the dynamic nature of applications and fails to minimize the impact of post-copy on VM performance.

Many studies have focused on two aspects of pre-copy shortcomings: optimization of reducing data transfer [16–20] and the migration process [21–24]. However, for memory write-intensive VMs, the migration performance is mediocre. Our proposed migration scheme does not conflict with most of the existing studies and offers better performance for memory write-intensive VMs. Furthermore, better migration performance can be achieved by integrating existing studies into our scheme.

This study proposes an efficient and reliable adaptive hybrid migration mechanism for VMs with memory write-intensive load. In our work, we developed an efficient parallel transfer mechanism based on different networks to optimize the pre-copy. The goal is to shorten the migration time and improve the migration efficiency through parallel processing of the full memory pages transmitted in the first phase and the dirty memory pages in subsequent iterations phase. Furthermore, applying the compression migration technology [16, 17] to the data of parallel processing can further improve the migration efficiency and obtain better migration performance. Based on the parallel transfer mechanism, a reliable adaptive hybrid migration is proposed that solves the problem of pre-copy nonconvergence. Moreover, hybrid migration optimizes the problems associated with post-copy and VM performance. We calculated the best adaptive switching timing according to the designed iterative convergence factor. The iterative convergence factor dynamically senses the real-time load on the environment and accurately achieves the iterative convergence state. According to the iterative convergence factor, the switchover from pre-copy to post-copy is completed at the optimal time, which facilitates the smooth migration and shortest post-copy duration. Finally, we design a fault tolerance mechanism to address the problem associated with the network faults after switching to post-copy migration. The goal is to adaptively switch request mode when sensing migration network faults to facilitate a smooth completion of migration and improve migration reliability.

The main contributions of this paper are as follows:

(1) We propose a parallel migration mechanism using different networks to improve migration efficiency.
(2) We propose an adaptive hybrid migration based on a parallel mechanism to improve migration reliability.
(3) We implemented our scheme based on KVM/QEMU platform, and our proposed migration mechanism could effectively reduce migration data and time. For memory write-intensive VMs, the existing migration mechanism often leads to migration failure or long downtime. However, our proposed migration mechanism demonstrates significant performance.

The rest of this paper is organized as follows: Sect. 2 introduces the research background and challenges. Section 3 discusses the core idea of the proposed migration mechanism. Section 4 provides an overview of our design. Section 5 describes the experimental setup and results. Section 6 introduces related work, and Sect. 7 summarizes the thesis and puts forward some suggestions for future work.

Background and challenges

Online VM migration is the cornerstone of cloud computing and dynamic resource deployment allocation. It improves data center performance, such as power consumption, resource usage, and load balancing [10]. Nevertheless, online VM migration demands many resources, such as memory capacity, communication bandwidth, and cache memory, which consequently affects the data efficiency and degrades running application performances [25]. This will seriously affect the cloud service provider's quality of service (QoS) during resources scheduling in the cloud data center. Efficient and reliable VM migration can effectively ensure the cloud service provider's QoS to reduce SLAv [26, 27]. Moreover, migrating the VMs that are running a write-intensive workload generates more dirt pages. Migration failures and large downtimes can lead to abortion of TCP (Transmission Control Protocol) connection. What's worse, when the VM memory size becomes larger, it will be more difficult to migrate VM with memory write-intensive loads.

Online migration technology

Total migration data, total migration time, and downtime are three key indicators for evaluating online migration. Total migration data represents the amount of data transferred from the source node to the target node during migration. Data migration consumes the network resources of both the source and target nodes because applications running on VM and migration threads share the same network infrastructures. The total migration time is defined as the time migration starts to the time the VM runs independently on the target host. The migration time directly affects the resource dynamic deployment efficiency in the data center and performance of VM. Avoiding issues that affect VM performance during migration is vital. Downtime is the period between VM suspension on the source node and VM recovery on the target node. The downtime should be short enough to have no noticeable impact on applications running inside the VM. The default maximum-expected downtime for KVM/QEMU [28] is 300 ms.

Clark et al. [8] proposed pre-copy online migration in 2005, and it was the most popular migration method today. The migration process is divided into three phases. Phase 1: full-copy phase, in which the migration begins by marking all memory pages as dirty to migrate all memory data. For example, if the 4 GB of VM memory is available for usage, the 4 GB of data is transferred in this phase. Therefore, this phase has the largest transmission data and the longest migration time in the entire migration iteration. Phase 2: iterative-copy phase, which iteratively transfers memory pages written dirtily during the previous round of data transfer. As the VM is running during the migration, some memory pages are updated, and the dirty page bitmap marks update memory pages and transmit them in the next iteration. Iteration will stop until the remaining number of pages is small enough. **Phase 3**: stop-copy phase, which suspends the VM to transmit the remaining memory pages and the device state data when the iterative copy is stopped. Pre-copy keeps iterating to achieve a short downtime. As a result, some memory pages are transferred multiple times, leading to a large amount of data transfer and a longer transfer time. In addition, when VMs are running memory write-intensive workloads, pre-copy faces the iterative convergence problem, failing to complete the migration within the expected downtime.

Hines et al. [13] proposed a post-copy migration mechanism to alleviate the pre-copy problem. First, post-copy suspends the VM at the start of migration. Then, it migrates the vCPU context and device status data to the target host and resumes the VM. Finally, the source VM memory pages is synchronized through on-demand and active push methods. When the target VM visits an unsynchronized memory page, the VM is interrupted and uses the Virtual Machine Monitor to obtain the required memory page from the source host through page fault request processing. Furthermore, the source host actively pushes unsynchronized memory pages to the target node. This mechanism transfers each memory page once, does not have the iterative convergence problem, and provides a reliable migration time. However, post-copy has poor reliability. If a migration network fault occurs during migration, the source and target host possesses the incomplete memory status to prevent VMs from being restored on any host [29]. In addition, frequent page faults interrupt and increase the memory access latencies and greatly reduce the VM performance [30].

Hybrid migration [14, 15] is the combination of the pre-copy and post-copy. In the first phase, multiple rounds of pre-copy migration are performed, during which most of the memory data is migrated. In the second phase, switches to post-copy migration are performed to migrate the remaining data. The process allows only a small amount of data to migrate during post-copy. Hybrid migration avoids the problems associated with a pre-copy non-convergence and facilitates smooth migration. However, the existing conventional hybrid migration mechanisms only perform one or two rounds of pre-copy migration and ignore the dynamic characteristics of applications. When the application load changes, setting a fixed number of pre-copy iterations either increases the duration of the post-copy or increases the number of invalid pre-copy iterations. In addition, if a

network failure occurs during post-copy migration, the VM cannot be restored.

Memory write-intensive VM migration

The VMs with memory write-intensive workloads consume a huge amount of memory and produce memory dirty pages at a very fast rate. As a result, these dirty pages are transferred repeatedly in the pre-copy iteration, leading to inefficient migration. When the rate of dirty pages generated exceeds the network bandwidth, the iteration fails to actively converge to the stop-copy phase and prevents the migration not completed within the expected downtime. With these problems, the migration process is forced into the stop-copy phase with a large number of dirty pages to transfer which leads to extended migration downtime and prolongs total migration time. This will greatly affect the performance of VMs which reduce customer satisfaction and bring unnecessary energy consumption. Efficient VM migration driven by customer satisfaction and energy efficiency can greatly reduce SLAv and energy consumption [31, 32]. In addition, an extended migration downtime can lead to service interruptions and possibly disconnection of clients, loss of database connections, or other issues. The following experiments illustrate the problems and challenges of migrating a write-intensive VM.

Challenges to migrating a memory write-intensive VM

Memcached is a write-intensive workload. We migrate a VM running the Memcached workload, and migration bandwidth is limited to 50 MB/s based on QEMU. During the pre-copy iteration, we keep track of the number of memory pages transferred in each iteration. As shown in Fig. 1, the number of memory pages migrated no longer decreases as iterations are performed in the third iteration because the dirty page generation rate is greater than the migration rate. If the maximum number of iterations is not set, iterations cannot converge to the stop-copy phase, resulting in migration failure. Setting the maximum number of iterations will result in unacceptable downtime and seriously affect VM performance. We set the maximum number of iterations to 30 and re-tested the iterations. The results show that the maximum downtime exceeds 12 s. This significantly degrades the Memcached performance. Therefore, completing memory write-intensive VM efficiently and reliably is a big challenge, which is the problem this study addresses.

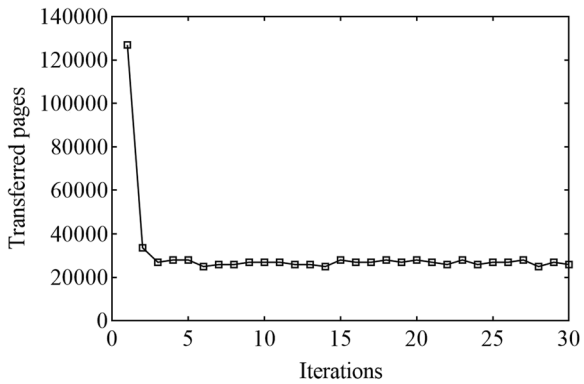

Fig. 1 Number of memory pages transferred in each pre-copy iteration when migrating a VM running Memcached

The VM memory sizes are becoming larger with the development of information technology. For example, Amazon EC2 X1e memory sizes range from 122 GB to 4 TB [33]. With this trend, it becomes trickier to migrate memory write-intensive VMs since longer iterating spent time leads to more dirty pages. To verify this problem through experiments, we increment the memory size of the VM running Memcached from 8 to 36 GB and set 10 concurrent access operations. The number of dirty pages for each VM in the first iteration was recorded during the migration. As shown in Table 1, the number of dirty pages increases almost linearly with the VM memory size. Therefore, as the memory increases, the migrated data also increases. In addition, each iteration transferred data is generated during the previous iteration. The more data pre-copy fully copies in phase 1, the more data it transfers in each iteration of phase 2. We performed the experiments to understand the influence of full-copy in phase 1 on subsequent iterations of phase 2. We tested on a 2:3 read–write ratio based on lmbench load to ensure the pre-copy iteration converged and set concurrent threads to increase as the VM memory grew. During the migration, we recorded the number of memory pages transferred in the first five iterations which contained most of the migration data. As shown in Fig. 2, as the VM memory size increases, the data transferred in the first iteration and subsequent iteration also increases. Thus, an increase in the data of the first iteration increases the amount of data of each subsequent iteration and the total migration time. However, we separate pre-copy phase 1 and phase 2 data migration for parallel processing to effectively improve the migration efficiency.

Table 1 Number of dirty pages of first iteration for different memory size VMs

mem (GB)	8	12	16	20	24	28	32	36
pages (K)	774	1386	1978	2595	3262	3893	4544	5153

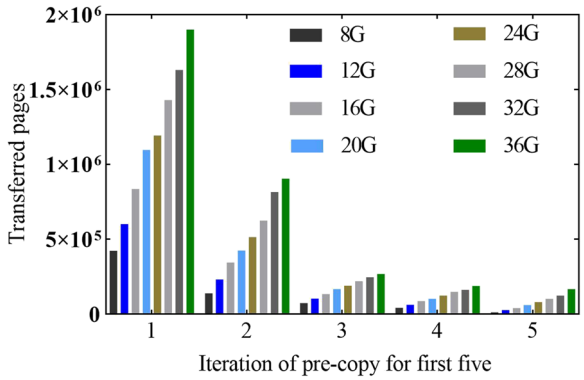

Fig. 2 Number of memory pages transferred in first five pre-copy iterations when migrating a VM running lmbench load for different memory size VMs

The above analysis shows that it is a great challenge to migrate VM with memory write-intensive workloads efficiently and reliably. In a production environment, the migration network and the production network share underlying network resources. Therefore, the bandwidth allocated for migration is limited. In addition, the VM memory size tends to increase, making it more difficult to migrate VM with memory write-intensive loads.

Core idea

To reduce the migration time and improve migration efficiency, the pre-copy full-copy (phase 1) and iterative-copy (phase 2) are conducted in parallel migration based on two channels. In addition, we propose an adaptive hybrid migration based on parallel mechanism and network fault tolerance to facilitate a smooth completion of the migration process and improve the stability of migration. Figure 3 presents the overall flowchart, elaborated in the next subsection.

Parallel migration using different net

The three-phase pre-copy migrations are serial tasks in time series. We migrate the full memory data in the first phase and the dirty memory pages transmitted in subsequent iterations phase through two channels based on the storage network and migration network, respectively. As shown in Fig. 3, the source VM is the full memory data to be copied in the first phase. At the beginning of the migration, the memory data in this area are saved to the shared storage based on channel-1, equivalent to taking a memory snapshot at the start of migration. We treat this part of the data as static data (StaticData). The data saved into shared storage is called StaticData backup. At the same time, channel-2 runs in parallel with channel-1 to transfer dirty pages of phase 2 to the target VM

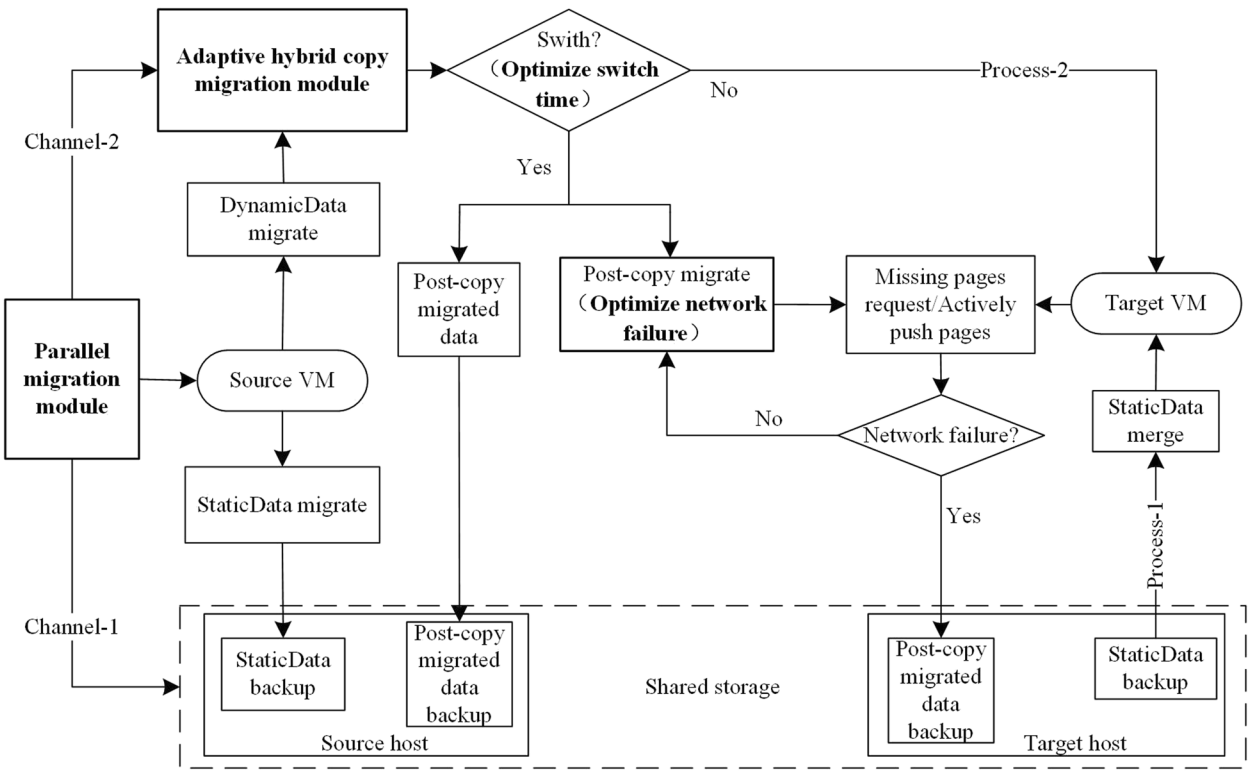

Fig. 3 Overview of flow char

through the migration network. Through continuous iteration, channel-2 completes the transfer of dirty pages generated during channel-1 migration. The transmitted data of channel-2 are modified continuously during migration. Thus, we treat this part of the data as dynamic data (DynamicData). The target node enables processes-1 and processes-2 to receive data on channel-1 and channel-2, respectively. Process-1 reads the StaticData backup from the shared storage, indirectly migrating the first phase data of pre-copy over the storage network. Process-2 receives dirty data from channel-2 over the migration network, which is equivalent to directly migrating data of the pre-copy second phase. After the StaticData migration is complete, we determine whether it is possible to proceed to the stop-copy phase, which is the third phase of pre-copy.

In the above process, StaticData migration and DynamicData migration are conducted in parallel and merged afterward. Data consistency is considered during merging. When the target node process-1 merges data, some memory pages of the VM already exist because channel-2 has transferred some memory dirty pages. Thus, when the process-1 merges data, it checks whether the corresponding memory page already has data. If the data already exists, the merge will not be performed since the data already exists in the VM memory.

Adaptive hybrid migration

The above parallel migration can effectively shorten migration time and improve migration efficiency. However, when the DynamicData is migrated based on channel-2 for memory write-intensive VMs, iterations will fail to converge, which increases the risk of migration failure and extends the migration time. To enhance the smooth migration of memory write-intensive VMs, an adaptive hybrid migration was proposed based on the optimization of the parallel transfer mechanism. The hybrid migration mechanism combines pre-copy with post-copy. This is crucial as the post-copy can avoid the iterative convergence of pre-copy and facilitate the completion of the migration. Traditional hybrid approaches typically perform one or two pre-copy iterations before switching to post-copy. Our proposed hybrid migration approach offers more effective leverage advantages of pre-copy and post-copy and minimizes the shortcomings associated with the traditional hybrid approach. Compared to traditional hybrid migration, our proposed method optimizes two aspects. Optimization of switching timing is the first aspect. This process adaptively switches the migration mode and senses the dynamic characteristics of the application to obtain the best migration benefits. Optimization of migration network fault is the second aspect that enhances the fault tolerance mechanism of the network. In this process, the network failure automatically switches request mode to further improve the stability and reliability of migration.

Optimization of switching timing

Channel-2 calculates an iterative convergence factor based on the number of dirty pages and transmission bandwidth of historical iterations that automatically determines whether to switch to post-copy. The principles of selecting the switching time are as follows: (1) The first is the adaptive sensing of the dynamic characteristics of the environment and network transport status during migration, which involves switching to post-copy migration on time when iteration fails to converge to avoid invalid iteration. When iteration is not converged, the memory dirty pages transmitted become invalid data, and the continuation of iteration increases the invalid data transmission, which is not conducive to the completion of the migration. (2) The computing switchover time ensures that the post-copy migration data and duration are minimized to avoid an impact on VM performance. This is necessary because the post-copy pages fault request greatly lengthens the memory access time.

Optimization of migration network fault

A network fault that occurs during post-copy migration can cause fatal problems to VM. Thus, we added network fault tolerance during post-copy migration to avoid this problem. As shown in Fig. 3, when switching to post-copy migration, the data to be migrated by post-copy is backed up based on shared storage. The backup data is the dirty pages generated during the last iteration, which is called post-copy backup data (PbData). Even if the migration network fails during post-copy, the target node chooses to load PbData from the shared storage to facilitate a smooth completion of the migration. The process increases the fault tolerance of the migration network and improves migration stability.

System design

We implemented the proposed scheme based on KVM/QEMU platform. For simplicity, our system prototype is implemented on QEMU-2.12.1 which contains the KVM module.

Parallel migration process using different net

VM memory pages are marked as StaticData and DynamicData when the parallel transmission is first launched. We modify the QEMU code to add a StaticData Bitmap (SBitmap) and mark the StaticData. The DynamicData are marked by QEMU dirty pages bitmap, called DynamicData Bitmap (DBitmap). During migration, StaticData and DynamicData are transferred in parallel using

different networks based on SBitmap and DBitmap. The process involves three phases: The first phase is to transfer StaticData and DynamicData. For the StaticData, a file is saved to share storage based on SBitmap. DynamicData is transmitted iteratively based on DBitmap. Each iteration transfers the memory page marked as "dirty" in the previous iteration. The second phase merges StaticData. After the source node has saved the StaticData, the target node loads the StaticData from the shared storage and merges it with DynamicData. When merging, DynamicData is treated as the latest data. If the target VM (TVM) has a corresponding DynamicData, the page is not merged. The third phase is stop-copy. After the second phase ends, stop-copy judgment is performed, and if the remaining data transmission time is less than the expected downtime, we suspend the source VM (SVM) and transfer the remaining data to the TVM. Then, the TVM is restored. Algorithm 1 shows the pseudo-code of this process.

Algorithm 1 Parallel migration process

```
Input: SVM
Output: TVM
1.  Initialize: SBitmap ←0, DBitmap ←0
2.  StaticData = GetStaticData(SBitmap)
3.  DynamicData = GetDynamicData(DBitmap)
4.  Do in parallel /* Parallel process */
5.      SaveStaticData(StaticData)
6.      TransmitDynamicData (DynamicData) /* To iterate over dynamic data until the stop-copy */
7.  End Do in parallel
8.  While read(StaticData) do
9.      If no data exists in TVM then
10.         Merge(StaticData) /* Merge data at the target host */
11.     End If
12. End While
13. If Merge StaticData end then
14.     If stop-copy is true then /* stop-copy judgment */
15.         Suspend source VM
16.         Transfer the rest of the data
17.         Return TVM /* migration complete*/
18.     End If
19. End If
```

Adaptive hybrid migration process

(1) The adaptive switching of the migration mode when iteration fails to converge.

During the migration, a data structure migrateInfo is used to record the total number of dirty pages $pages(i)$ and convergence factors $\lambda(i)$ transferred during each iteration of DynamicData migration. The $\lambda(i)$ directly reflects the convergence of this iteration, and indirectly reflects the dynamic characteristics of the application load on the VM. The application load that changes during the migration is sensed by $\lambda(i)$. The $\lambda(1) <$ when the VM memory update rate is less than the network transfer rate, indicating converging of the iteration. Otherwise, the iteration is divergent. In each iteration, $\lambda(1) <$ allows the migration to be completed quickly. Occasionally, $\lambda(i) >$ when the network or application load fluctuates. This does not affect the normal migration completion. However, if the λ is greater than 1 many times, this indicates that the iteration is no longer convergent. Moreover, continuing iteration is not conducive to the migration completion as it increases the migration time and invalid data transfer. The λ is related to the memory dirty page generation rate R and transmission bandwidth B. Assuming that the transmission time is $T(i)$ and the transfer memory dirty pages are $pages(i)$ in a round i during the iterative migration process, the $R(1)$ and $B(i)$ are expressed as Eqs. (1) and (2), respectively:

$$R(1) = \frac{pages(i+1)}{T(i)} \tag{1}$$

$$B(i) = \frac{pages(i)}{T(i)} \tag{2}$$

The convergence factor λ is expressed as:

$$\lambda(i) = \frac{R(i)}{B(i)} = \left\{ \begin{array}{ll} \lambda(i) = 1 & i = 1 \\ \lambda(i) = \dfrac{pages(i+1)}{pages(i)} & i > 1 \end{array} \right\} \tag{3}$$

The target node notifies the source node after the StaticData is merged. From now on, at the beginning of each subsequent iteration, the source node decides whether to switch to post-copy based on the $\lambda(i)$. The specific strategy is as follows: Observe the value $\lambda(i)$ for the last 3 times, when all $\lambda(i)$ values are less than 1, the migration iteration is in the state of convergence, revealing that switching is not required. When all $\lambda(i)$ values are greater than 1 at least 2 times, we calculate the average value λ_{aver} of $\lambda(i)$ the last 3 times. If the $\lambda_{\text{aver}} \geq 1$, the iteration no longer converges, and the post-copy mechanism is used. The average value of the convergence factor is used as the reference value at the switching time, which accurately represents the convergence trend. This process prevents misjudgment caused by the occasional fluctuation of the network or application load.

(2) Adaptive switching of request method in case of the network failure.

When switching to the post-copy migration, the target VM has most of the source data. Thus, reducing the time of post-copy migration time is vital. However, if a network fault occurs during

post-copy migration, the VM will fail to be restored. To solve this problem, we saved the corresponding dirty pages to the shared storage and named PbData, according to the DBitmap of the final iteration of channel-2. When the target node requests the memory page from the source node and detects a migration network fault, the target node loads the remaining data from the shared storage to facilitate the completion of the migration. In this process, the fault migration network tolerance is realized, which increases migration stability and reliability. Algorithm 2 shows the pseudo-code of the network fault-tolerant adaptive hybrid migration process.

Algorithm 2 Adaptive hybrid migration process

```
Input: migrateInfo, DBitmap
Output: migrationMethod
If Merge StaticData end then
  PbData = getPostcopyBackupData(DBitmap)
  While transmit DynamicData do
    Get the number of λ greater than 1 from migrateInfo
    If the number of λ greater than 1 ≥ 2 then
      Compute λ_aver from migrateInfo
      If λ_aver ≥ 1 then
        migrationMethod=doPostCopyMigrate
        Save PbData to share storage
        While doPostCopyMigrate do
          If the migration network is ok then
            Get data from the source node
          Else
            Get data from PbData
          End If
        End While
      Else
        MigrationMethod=doPreCopyMigrate
      End If
    End If
  End While
  Return migrationMethod
End If
```

The experimental evaluation

Experimental evaluation index and scheme design

This section evaluates our proposed migration mechanism. The total migration time and data, VM downtime, and the impact on VM performance are important performance indicators. The lower the total migration time and data, the higher the migration efficiency, and the smaller the VM downtime, the lesser the impact on VM performance. In addition, the shorter the post-copy duration of hybrid migration, the smaller the impact on VM performance.

The performance of the proposed scheme was evaluated based on the following workload, using the mean of the 10 test results.

Memcached

This is a memory and network-intensive workload that stores key-value pairs based on memory. When the Memcached server receives a request that contains a key value, the corresponding value responds to it. We configured the Memcached VM with 8 vCPU and 6 GB of memory, and allocated Memcached with 4 GB of cache and 2 threads. The client uses the Memaslap test suite from the libmemcached library. The client first preheats the Memcached server with multiple key values to fill the 4 GB cache. During the migration, the client sets concurrent connections to 4 based on set operations.

Kernel compilation

This program is a system call-intensive load that can test the performance of various resource types equally. We allocated a VM with 4 vCPU and 2 GB of memory to compile the Linux-3.10.0–957 kernel with 4 threads in parallel.

Lmbench

This is an open-source and widely used benchmark for performance testing. We allocated 8 vCPU and 6 GB of memory to a VM running lmbench and ran 4 threads concurrently to perform 2:3 read and write tests on a 4 GB memory area.

The results reveal that our scheme can significantly improve migration performance. Compared with pre-copy, our parallel transmission greatly reduces the data and time of migration and significantly improves migration efficiency. Some workloads that did not migrate within the expected downtime using pre-copy now completed the migration quickly based on our parallel mechanism. In addition, the adaptive hybrid migration based on the parallel mechanism guarantees the successful migration of all workloads. Compared with the traditional hybrid migration with fixed threshold switching, our scheme effectively reduces the failure probability of migration, shortens the duration of post-copy, and reduces the inherent weaknesses of post-copy. In the following, we discuss the proposed migration mechanism in detail.

Experimental environment setup

As revealed in Fig. 4, two physical servers perform the functions of the source and target node, and their specific

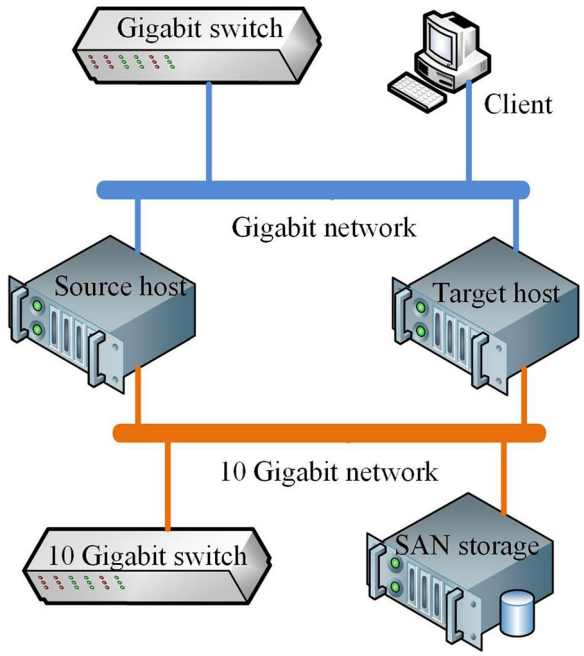

Fig. 4 Topology of the experimental environment

Table 2 Test physical server configurations

name	Value
Type	Inspur NF5280M5
CPU	CPU Intel Xeon Gold 5118 (2.3 GHz/12 C)
Memory	8*32 G RDIMM DDR4
NIC	One dual-port Gigabit NIC Intel I350, One dual-port 10 Gigabit NIC Intel 82599ES
Disk	M.2 interface Intel DC S3520

configurations are shown in Table 2. These servers connect to the gigabit network through gigabit switches. Additionally, the host node shares the back-end storage from another machine using the SAN storage, and the SAN storage server transmits data to the host node over a 10-gigabit network. The Client–server runs the memaslap benchmark as the test Client for Memcached. Channel-1 is based on a 10-gigabit network for data transmission, and channel-2 is based on a gigabit network for data transmission. The test VM was deployed on CentOS7.16–1810. The system prototype is based on QEMU-2.12.1. The maximum downtime and iterations are 300 ms and 30 times, respectively. The channel-2 network's bandwidth is 32 MB/s.

Parallel migration performance

First, we verify the performance of the parallel migrate (par-migrate) mechanism by comparing the par-migrate with the normal pre-copy [8].

As shown in Fig. 5, par-migrate significantly reduced the total migration time and data under different load tests compared to pre-copy. For Memcached, the total migration time and data were reduced by 37.7% and 20.2%, respectively. For the kernel compilation, total migration time and data were reduced by 29.5% and 8.3%, respectively, which were 39.1% and 19.2% lower for lmbench load, respectively. Two main reasons are associated with these significant performance improvements:

Firstly, the par-migrate transmits phase 1 and phase 2 data in parallel based on different networks. The phase 1 and phase 2 data transmissions do not affect each other in par-migrate, which greatly improves the data transmission speed and processing efficiency.

Secondly, as shown in Fig. 2, the initial data of pre-copy iterative migration in phase 2 is related to the full copy of phase 1. The larger the amount of data migrated in phase 1, the longer the migration time, the more dirty

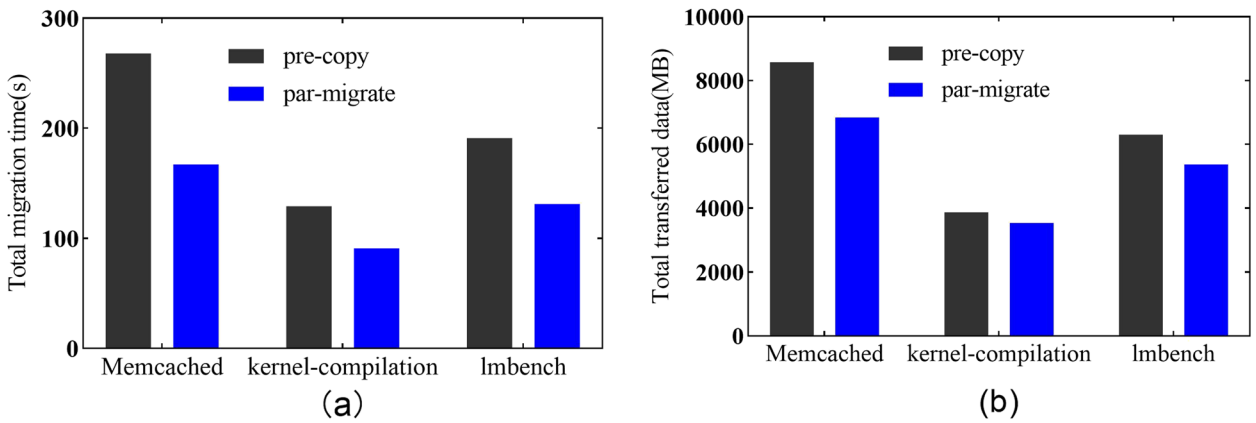

Fig. 5 Performance comparison for different workloads. **a** Total migration time **b** Total transferred data

pages generated in phase 1, and the larger the initial data of phase 2. By analogy, the data transferred and the execution time of each subsequent iteration increase. While par-migrate is not associated with this problem, phase 1 migration does not affect the phase 2 initial data because both are processed parallelly. Therefore, the data migrated in subsequent iterations was greatly reduced. We performed an experiment to understand that the iterative migration of data in phase 2 is not affected by phase 1 using par-migrate. We tested on a 2:3 read–write ratio based on lmbench load and set concurrent threads as the VM memory grows. During the migration, we recorded the memory pages transferred in phase 1 and the first iteration of phase 2. As shown in Fig. 6, as the memory pages transferred in phase 1 become larger, the memory pages in the first iteration of phase 2 remain almost unchanged, revealing that phase 2 iteration migrations can maintain a small initial data level using par-migrate.

Figure 7 shows that pre-copy downtime for all three loads exceeds the maximum downtime of 300 ms, in which kernel compilation is at a minimum of 1210 ms and Memcached is at a maximum of 11,217 ms. For par-migrate, the Memcached's load downtime exceeded 300 ms, at 5912 ms. However, the other two loads are within 300 ms.

For pre-copy, three loads generate memory dirty data at a rate greater than the rate of transfer, and the iterations fail to converge resulting in reaching the maximum iteration, which leads to a longer downtime due to a large amount of transferred data in the stop-copy phase. As revealed in Table 3, pre-copy transmitted data was significantly higher than par-migrate in all three loads during the stop-copy phase. However, with the Memcached load, par-migrate was unable to enter the stop-copy phase naturally due to the memory dirty pages ware generated very fast, resulting in significant downtime. But, our comprehensive scheme addresses this problem, which is discussed in section of comprehensive performance.

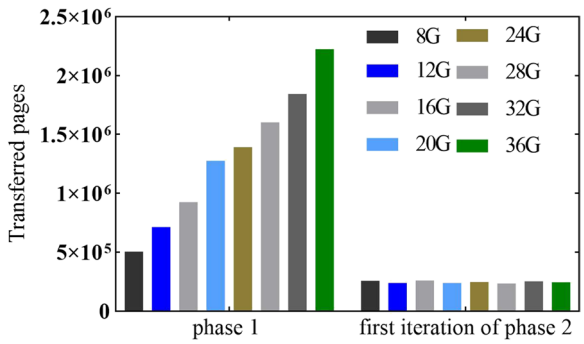

Fig. 6 Number of memory pages transferred in phase 1 and the first iteration of phase 2 when migrating a VM running lmbench for different memory size VMs

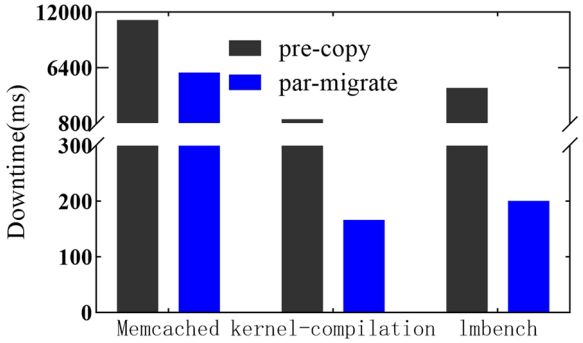

Fig. 7 Downtime

Adaptive hybrid migration performance

The adaptive switching for the proposed hybrid migration (As-hybrid) is compared with the traditional hybrid migration based on fixed switching thresholds. That is, traditional hybrid migration [14, 15] with only 1 round of pre-copy (1-hybrid) and hybrid migration with only 2 rounds of pre-copy (2-hybrid). This section analyzes the performance of As-hybrid separately and excludes the benefits of par-migrate. We performed migration tests on VM running Memcached loads for two scenarios:

> *Scenario 1*: The read/write ratio of memaslap client increases during migration, enabling As-hybrid to trigger migration mode switchover.
> *Scenario 2:* At the beginning of the migration, we set the read/write ratio of the memaslap client to 10:1 to slow the memory update frequency. During migration, the read/write ratio is occasionally set to 1:10 to test the performance under load fluctuation.

During the experiment, the average latency for the Memcached service and the amount of data transferred during post-copy migration are recorded in real-time. These two metrics were used to analyze the performance.

Figure 8a shows the test results of scenario 1. During the migration process, 1-hybrid and 2-hybrid switch to post-copy at 140 s and 157 s, respectively. Moreover, As-hybrid automatically senses the load changes based

Table 3 Send data in stop-copy phase (MB)

	Memcached	kernel-compilation	lmbench
pre-copy	349.4	38.7	126.8
par-migrate	165.5	5.2	6.1

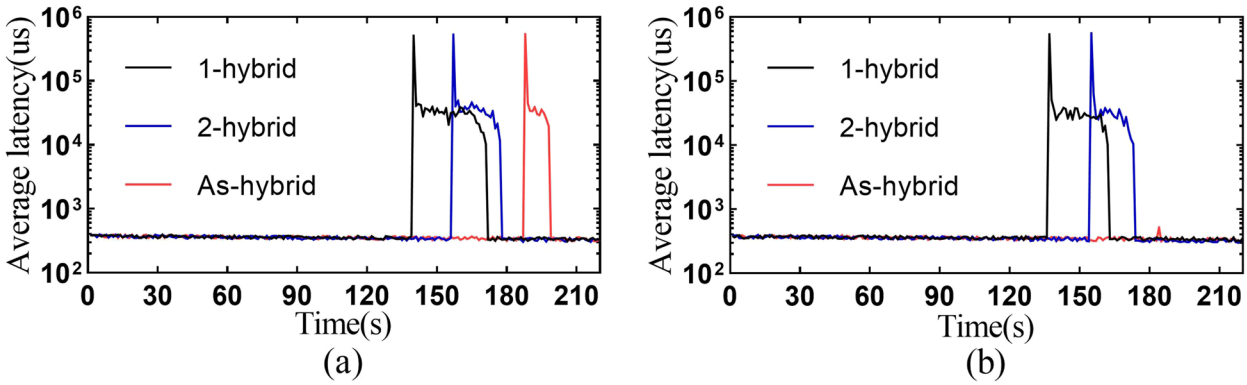

Fig. 8 Memcached service performance. a with increasing load. b with occasional load fluctuations

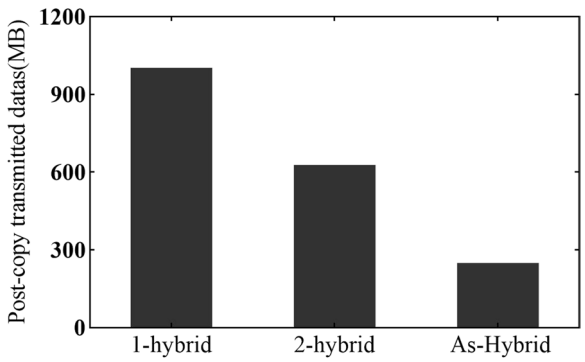

Fig. 9 Transferred data during post-copy

Table 4 Total migration time and transferred data

	1-hybrid	2-hybrid	As-hybrid
Total time (s)	299	321	332
Total data (MB)	8970	9951	10,292

on iterative convergence factors and triggers the switch to post-copy at 187 s. However, As-hybrid has the shortest duration, 66% and 48% shorter than 1-hybrid and 2-hybrid, respectively. As shown in Fig. 9, As-hybrid transmits the least data during post-copy, which effectively shortens the post-copy duration. As revealed in Table 4, the total migration time and data for As-hybrid increased. Given this problem, it is solved by combining As-hybrid with the par-migrate, that is our comprehensive scheme.

The test results of scenario 2 are shown in Fig. 8b, revealing that 1-hybrid and 2-hybrid are unable to sense load fluctuations. Thus, the post-copy migration switch is bound to trigger, leading to the Memcached service latency. While As-hybrid uses the proposed iteration factor to filter such load fluctuations, it avoid unnecessary switching and the Memcached service latency.

As shown in Fig. 8, the average latency of Memcached service response time decreases significantly during post-copy migration, and pre-copy has no significant impact. Post-copy page fault request is the direct cause of Memcached service response latency, which seriously affects VM performance. When the pre-copy iteration is convergent in hybrid migration, the migrated data becomes smaller with the increase of pre-copy iterations. If the load changes during the migration process, the pre-copy iteration will fail to converge. With this issue, it is no longer meaningful to continue iterating as the data to be migrated will not decrease but may even increase. Based on Scenario 1, we test with different fixed switching thresholds (1–10 pre-copy iterations). As shown in Fig. 10, when the threshold is less than 5, the data transferred by a post-copy decrease as pre-copy iterations increase. However, when the threshold is greater than 5, the data transferred by the post-copy increase as pre-copy iterations increase. Thus, switching to post-copy at the 5th pre-copy iteration can obtain the optimal post-copy duration and total transfer data. The duration of post-copy is significantly shortened as As-hybrid adaptively senses the iterative convergence state based on the iterative convergence factor and switches at the optimal moment.

Network fault tolerance

This section tests the adaptive switchover of data request mode. When a network failure occurs during post-copy, the source and target VM are impaired and unable to work properly. This problem can be solved by As-hybrid of network fault tolerance mechanism. The VM running Memcached load was migrated, and the migration network was disconnected after switching to post-copy.

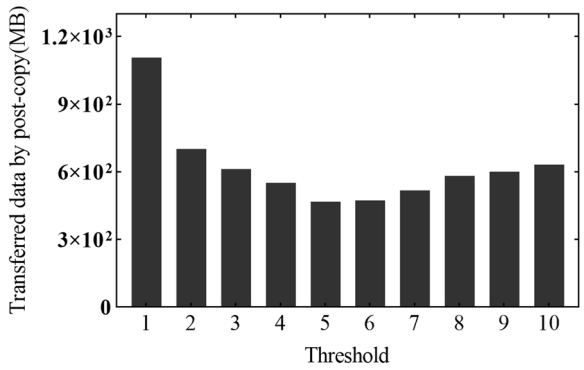

Fig. 10 Data transferred using post-copy based on different fixed switching thresholds when migrating the increasing workloads of Memcached

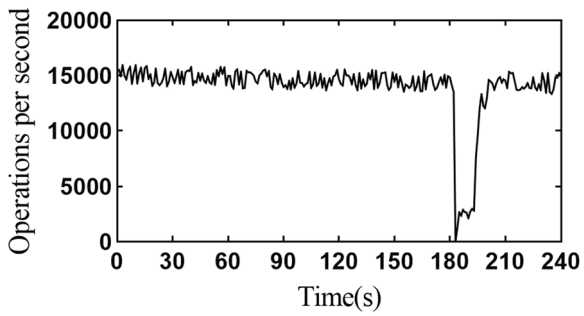

Fig. 11 Memcached service throughput rate

During the migration, the throughput rate of the Memcached service was counted. As shown in Fig. 11, the throughput decreased after switching to post-copy but returned to normal after the migration was completed at 195 s. The result verifies that when the migration network is disconnected during a post-copy migration, the migration process loads data from the shared storage rather than interrupting the migration. Therefore, VMs can still be migrated even after the migration network is disconnected. In this way, migration failure caused by migration network faults is avoided.

Comprehensive performance

In this section, the comprehensive performance of par-migrate and As-hybrid combined (PA-hybrid) is discussed. We perform the experiment using a VM with a larger memory size and faster write dirty speed. We set the VM memory to 32 GB that run the Memcached load. We allocate 16 GB of cache and 12 processing threads for Memcached. The memaslap concurrent connections were set to 10 with a read/write ratio of 3:7. To improve test efficiency, we set the migration bandwidth to 50 MB/s. The tests were compared with 1-hybrid,

Table 5 Total migration time and transferred data

	1-hybrid	2-hybrid	Dyn-hybrid	PA-hybrid
Total time (s)	605.9	685.4	697.2	465.5
Total data (MB)	30,720	33,792	34,394	22,528

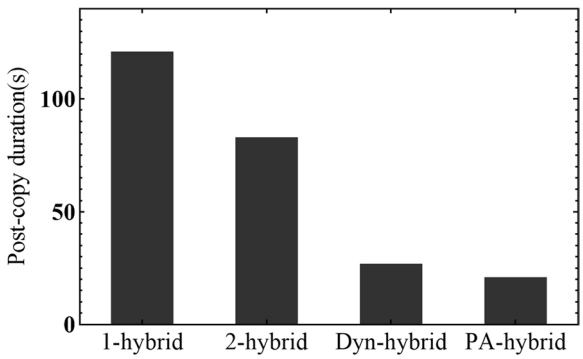

Fig. 12 Post-copy duration

2-hybrid and dynamic hybrid migration (Dyn-hybrid) [14]. The Dyn-hybrid dynamically sets the number of pre-copy based on the memory dirtying rate, the memory size, and migration network bandwidth to obtain the best migration performance, which can effectively solve the problem of memory write-intensive VM migration and reduce the duration of post-copy. Table 5 shows the test results of total migration time and transmitted data. Compared with the 1-hybrid, total migration time and data were reduced by 23.2% and 26.7%, respectively, which were 32.1% and 33.3% lower than 2-hybrid, respectively. Compared with the Dyn-hybrid, total migration time and data were reduced by 33.2% and 34.5%. Therefore, the PA-hybrid demonstrates a more significant performance improvement than the 1-hybrid, 2-hybrid and Dyn-hybrid. Figure 12 shows the post-copy duration during the migration, revealing that the PA-hybrid post-copy duration is the shortest, only 21 s. The second best is Dyn-hybrid which closer to PA-hybrid. However, the maximum duration of 1-hybrid has reached more than 100 s. In addition, we tested the migration performance of lmbench and kernel compilation workloads. As shown in Fig. 13, our comprehensive scheme exhibited the best performance for total migration time, total transferred data, and post-copy duration.

As revealed in Table 6, the migration data and time are larger than those of traditional hybrid migration for As-hybrid which does not contain a parallel mechanism. Because our scheme performs more rounds of pre-copy iteration to find the optimal switching moment compared to 1-hybrid and 2-hybrid, resulting in more data

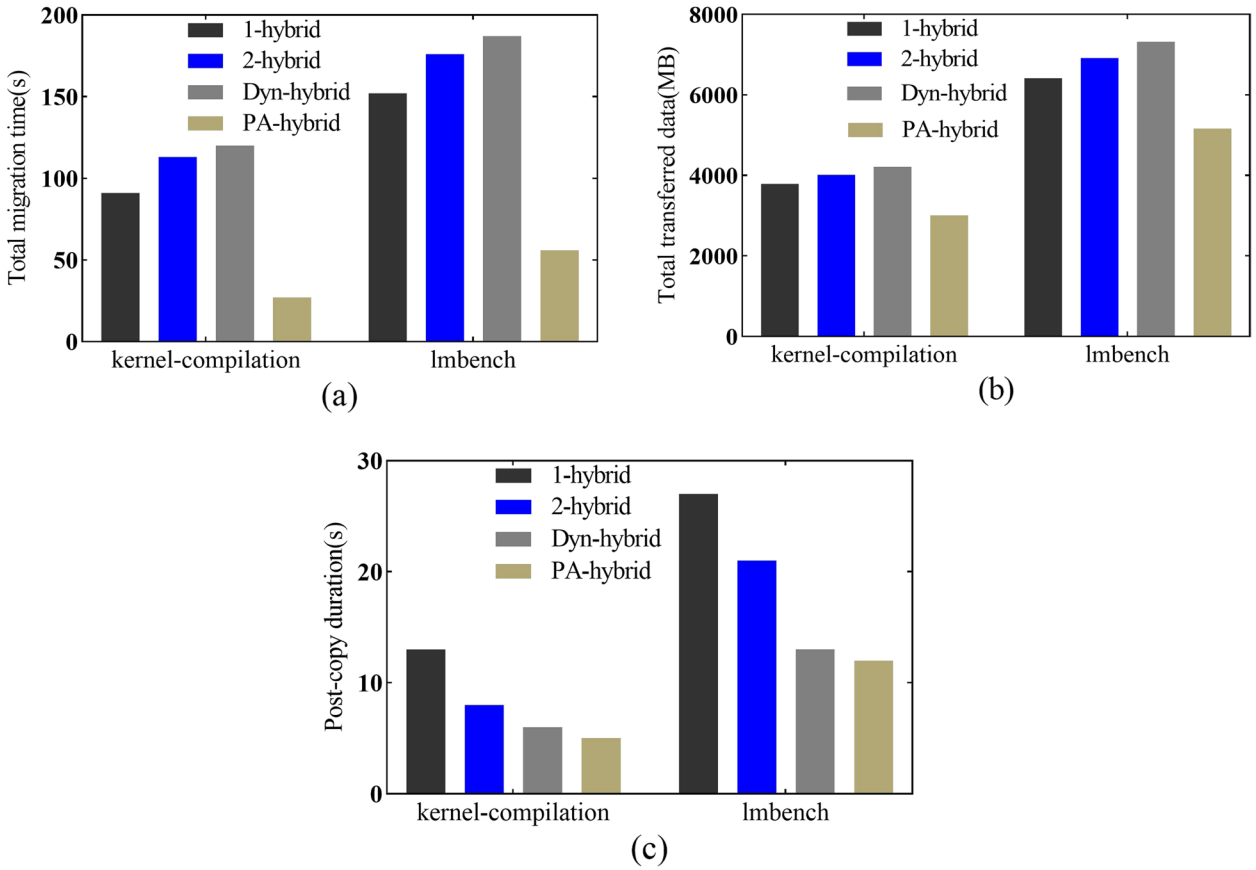

Fig. 13 Performance comparison for different workloads. **a** Total migration time. **b** Total transferred data. **c** Post-copy duration

being transferred. The Heuristic hybrid migration [21] and the Dyn-hybrid [14] also suffers from this problem, although it can enhance the successful migration of memory write-intensive VMs and shorten post-copy migration time. However, our hybrid migration combined with a parallel mechanism can effectively solve this problem. From the analysis of the challenges to migrate a write-intensive VM as know, the more data pre-copy fully copies in phase 1, the more data it transfers in each iteration of phase 2. As a result, traditional hybrid mechanism and Dyn-hybrid mechanism transfer more data during pre-copy and post-copy. However, par-migrate transmits data of phase 1 and phase 2 based on two parallel channels, and these data transfers do not affect each other that greatly improve migration efficiency. In addition, Fig. 6 also shows that the initial transfer data of phase 2 is always kept at a small value. Although multiple rounds of iterations are performed, the data transmitted in each round and the data migrated by post-copy are smaller.

From the above experiments, our proposed migration mechanism can effectively shorten the migration time and improve the migration efficiency for VMs with memory write-intensive load. Moreover, the migration mechanism offers significant stability and reliability as it facilitates the migration of all applications-loaded VMs even with migration network failure. Compared with the conventional hybrid mechanism, the algorithm significantly reduces the post-copy duration and reduces the impact of the migration process on VM performance.

Related works

VM migration technology has been widely studied by scholars globally. Clark et al. [8] proposed the pre-copy migration mechanism in 2005, demonstrating VM data transmission to the target host through multiple iterations. Due to strong reliability, pre-copy migration has been widely used in existing virtualization platforms. For memory write-intensive VMs, pre-copy suffers from migration inefficiencies and iterative convergence problems. Given these problems, many existing studies focus on two aspects: optimization of reducing data transfer and optimization of the migration process.

Optimizations to reduce data transfer

Data compression technology is applied to VM migration to reduce the amount of migration data and improve

migration efficiency. Jin et al. [16] proposed a simple and fast WKdm compression method for memory pages with large similarities. They also use a complex and slow LZO (Lempel–Ziv–Oberhumer) compression algorithm for less similar memory pages. Singh et al. [17] proposed a new model based on geometric programming that dynamically allocated compression rates according to the available network bandwidth. With low available bandwidth using a slow compression algorithm and more bandwidth using fast compression algorithms. Zhang et al. [18] accelerated VM migration based on differential compression technology, using the similarity detection algorithm of HashSimilarityDetector [19]. The algorithm identified memory pages with similar contents in the entire VM memory and addressed space using a differential compression algorithm. The compression technology can effectively reduce the amount of migrated data and improve migration efficiency. Liu et al. [20] optimized pre-copy migration using system checkpoint recovery and record playback techniques. They sent log files to the target node, which recorded VM non-deterministic system events that occurred in the previous iteration migration. Then, the target node plays back the received log files to restore the state of the VM. As smaller log files are sent instead of memory data, migration data is significantly reduced. However, this approach is unsuitable for migrating VM with symmetrical multiprocessor architecture operating systems. Since memory competition between different vCPU for recording and playback is very expensive. Riteau et al. [34] proposed a VM migration method called "Shrinker". This method identifies the memory pages that have been transferred by a service, and when memory pages with the same content are transferred, only their identification is transferred. Thus, this method can effectively reduce the transferred data. The above algorithm only optimizes the amount of data transferred during the migration, which partially improves the migration efficiency. However, the iterative convergence of pre-copy migration is not considered for VMs with memory write-intensive workloads.

Optimization of the migration process

The common solution for the pre-copy iteration convergence problem is to set the maximum number of iterations and force the VM to suspend when the maximum number of iterations is reached. Hines et al. [13] proposed the post-copy mechanism to avoid the pre-copy iterative convergence problem. However, its frequent page fault request increases the VM delay memory access, which has a serious impact on VM performance [30]. Li et al. [21] proposed a heuristic hybrid migration based on pre-copy and post-copy. They compared the number of dirty pages generated and the number of pages transmitted in each iteration. If the number of dirty memory pages generated is not less than the number of transferred memory pages, post-copy migration is enabled. This avoids the convergence problem of pre-copy iteration and reduces the impact of pre-copy migration on VM performance. However, the network failure during a post-copy migration will cause the migration failure. Deshpande et al. [22] proposed a scatter–gather migration method, in which the data to be migrated is distributed to multiple nodes, and these nodes are treated as a relay node. The target node obtains migration data from the relay node and the source node, thereby improving the efficiency of data transmission. Zaw et al. [35] proposed a framework to extend pre-copy based on the pre-processing mechanism. They combine Least Recent Used (LRU) cache and splay tree algorithm to predict the working set and pre-process the pre-copy to reduce the transfer of memory pages.

However, the above studies fail to consider VMs' parallel migration in the optimization of data transfer reduction and the optimization of the migration process. Song et al. [23] proposed a time-limited parallel migration method, using data and pipeline parallelism to parallelize the migration process. However, the parallel data transmission in the same network increases the competition for network resources. Akiyama et al. [24] focused on VM migration based on shared storage, revealing that the normal pages and the cached pages of VMs were transmitted in parallel. The normal pages were transmitted through the migration network, and the target host loaded the cached pages from the shared storage. However, Akiyama et al. [24] need to modify client operating system, which limited to the open-source Linux system and not applicable to the Windows system.

In addition, this paper studies VM migration based on shared storage. However, in a WAN scenario with non-shared storage, VM storage data needs to be transferred. Zheng et al. [36] proposed a storage migration scheduling algorithm. They use the history of VM disk I/O operation to predict the I/O locality characteristics of the migrated workload. Based on this feature, migration scheduling is carried out. This effectively solves the problem that VM I/O performance deteriorates during storage migration. Wood et al. [37] proposed an effective migration framework to efficiently migrate VM in WAN based on VPN. The framework reduces the cost of transferring VM storage and memory data over a WAN. Migration of non-shared storage scenarios is not the focus of this paper, but will serve as our future research in VM migration.

Conclusion

This study proposes an efficient and reliable online migration for memory write-intensive VMs. Firstly, this paper proposes a parallel migration mechanism to improve the efficiency of VM migration. We analyze the existing problems associated with the memory write-intensive VMs' online migration. To solve these problems, we implement an efficient parallel transmission mechanism that migrates data based on two channels and receiver threads. Our proposed scheme shortens migration time and improves migration efficiency. Secondly, this paper proposes an adaptive hybrid migration to improve the reliability of VM migration. The adaptive hybrid migration is implemented based on parallel transmission. We select the optimal switching time based on the proposed iterative convergence factor when pre-copy iterations fail to converge to achieve a smooth completion of migration and minimize the performance impact of post-copy migration on VMs. The strategy helps to improve the performance and reliability of the migration. To improve network reliability and avoid shortcomings associated with post-copy migration, the network fault tolerance mechanism is added after the adaptive hybrid migration is switched to post-copy. Finally, the validity of this study is verified experimentally. The proposed migration mechanism is evaluated and reveals that the mechanism has obvious performance advantages for VMs migration.

This study mainly focuses on online migration in VMs based on shared storage for memory write-intensive load in a LAN. Future work will focus on a WAN scenario without shared storage, with low bandwidth, high latency, and other problems. Future work will also explore a strategy to effectively shorten the migration time and improve migration efficiency and reliability.

Abbreviations

VMs	Virtual machines
KVM	Kernel-based virtual machine
vCPU	Virtual central processing unit
SAN	Storage Area Network
NAS	Network Attached Storage
TCP	Transmission Control Protocol
SBitmap	Static data Bitmap
DBitmap	Dynamic data Bitmap
TVM	Target VM
SVM	Source VM
par-migrate	Parallel migrate
As-hybrid	Adaptive switching of hybrid migration
1-hybrid	Traditional hybrid migration with 1 round of pre-copy
2-hybrid	Traditional hybrid migration with only 2 rounds of pre-copy
PA-hybrid	Adaptive switching of hybrid migration of par-migrate and As-hybrid combined
SLAv	Service-level agreement violation
Dyn-hybrid	Dynamic hybrid migration

Acknowledgements

The authors would like to thank the anonymous referees for their valuable comments and suggestions.

Authors' contributions

Pingping Li put forward the main ideas, contributed to the modeling, conducted the experiments, performed the data analysis and wrote the manuscript; Jiuxin Cao guided the research and made suggestions for the article. All authors read and approve the final manuscript.

Funding

Project is supported by the National Natural Science Foundation of China (62172089,61972087,62172090) Forward-looking and Key R&D Project of Jiangsu Province (SBK2019022870). Key Laboratory of Network and Information Security Project of Jiansu Province (BM2003201).

Availability of data and materials

The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request.

Declarations

Competing interests

The authors declare that they have no competing interests.

Received: 8 May 2022 Accepted: 26 March 2023
Published online: 06 April 2023

References

1. Wan X, Zhang X, Chen L (2012) An improved vtpm migration protocol based trusted channel. In: Proceedings of 2012 International conference on systems and informatics. Yantai, pp 870–875
2. Dong Y (2013) Efficent migration of virtual functions to enable high availability and resource rebalance. US Patent 8:533–713
3. Cao R, Tang Z, Li K, Li K (2021) HMGOWM: a hybrid decision mechanism for automating migration of virtualmachines. IEEE Trans Serv Comput 14(5):1397–1410. https://doi.org/10.1109/TSC.2018.2873694
4. Shen H, Chen L (2020) A resource usage intensity aware load balancing method for virtual machine migration in cloud datacenters. IEEE Trans Cloud Comput 8(1):17–31. https://doi.org/10.1109/TCC.2017.2737628
5. Mireslami S, Rakai L, Wang M, Far BH (2021) Dynamic cloud resource allocation considering demand uncertainty. IEEE Trans Cloud Comput 9(3):981–994. https://doi.org/10.1109/TCC.2019.2897304
6. Kherbache V, Madelaine É, Hermenier F (2020) Scheduling live migration of virtual machines. IEEE Trans Cloud Comput 8(1):282–296. https://doi.org/10.1109/TCC.2017.2754279
7. Le T (2020) A survey of live virtual machine migration techniques. Comput Sci Rev 38(11):100304
8. Clark C, Fraser K, Hand S (2005) Live migration of virtual machines. In: Proceedings of the 2nd conference on symposium on networked systems design and implementation. IEEE Piscataway NJ USA 2:273–286
9. Nelson M, Lim B H, Hutchins G (2005) Fast transparent migration for virtual machines. In: Proceedings of the USENIX annual technical conference. Anaheim, pp 391–394
10. Nitin SM, Rajesh BI (2020) Optimizing the topology and energy-aware vm migration in cloud computing. International Journal of Ambient Computing and Intelligence 11(3):42–65
11. Kostenko VA, Chupakhin A (2020) Live migration schemes in data centers. Program Comput Softw 46(5):312–315
12. Jin H, Gao H, Wu W, Shi S, Wu X, F. Zhou F, (2011) Optimizing the live migration of virtual machine by CPU scheduling. J Netw Comput Appl 34(4):1088–1096
13. Hines MR, Deshpande U, Gopalan K (2009) Post-copy live migration of virtual machines. Operating systems review 43(3):14–26
14. Altahat MA, Agarwal A, Goel N, Kozlowski J (2020) Dynamic hybrid-copy live virtual machine migration: Analysis and comparison. Procedia Computer Science 171:1459–1468
15. Sahni S, Varma V (2012) A hybrid approach to live migration of virtual machines. In: Proceedings of IEEE International Conference on Cloud Computing in Emerging Markets (CCEM). Bangalore, pp 1–5

16. Hai J, Li Deng Wu, Song, (2009) Live virtual machine migration with adaptive memory compression. Proceeding of the IEEE International Conference on Cluster Computing and Workshops. Los Alamitos, USA, pp 1–10
17. Singh G, Singh AK (2021) Optimizing multi-VM migration by allocating transfer and compression rate using geometric programming. Simul Model Pract Theory 106:102201
18. Zhang X, Huo Z, Ma J (2010) Exploiting data deduplication to accelerate live virtual machine migration. In: Proceedings of the IEEE International Conference on Cluster Computing and Workshops (CLUSTER). Heraklion, pp 88–96
19. Gupta D, Lee S, Vrable M (2008) Difference engine harnessing memory redundancy in virtual machines. In: Proceeding of the 8th USENIX Symposium on Operating Systems Design and Implementation. Berkeley, pp 309–322
20. Haikun L, Hai J, Xiaofei L (2009) Live migration of virtual machine based on full system trace and replay. Proceeding of the 18th ACM International Symposium on High Performance Distributed Computing. ACM, New York, pp 101–110
21. Li C, Feng D, Hua Y, Qin L (2019) Efficient live virtual machine migration for memory write-intensive workloads. Future Gener Comput Syst (FGCS) 95(1):126–139
22. Deshpande U, Chan D, Chan S (2018) Scatter-gather live migration of virtual machines. IEEE Transactions on Cloud Computing 6(1):196–208
23. Song X, Shi J, Liu R (2013) Parallelizing live migration of virtual machines. In: Proceedings of the ACM SIGPLAN/SIGOPS international conference on Virtual Execution Environments (VEE). Houston, pp 85–95
24. Akiyama S, Hirofuchi T, Takano R (2016) Fast live migration for IO-intensive vms with parallel and adaptive transfer of page cache via SAN. IEICE Trans Inf Syst 99(12):3024–3034
25. Choudhary A, Govil MC, Singh G, Awasthi LK, Pilli ES, Kapil D (2017) A critical survey of live virtual machine migration techniques. J Cloud Comput 6(23). https://doi.org/10.1186/s13677-017-0092-1
26. Li H, Zhu G, Zhao Y, Yu D, Tian W (2017) Energy-efficient and QoS-aware model based resource consolidation in cloud data centers. J Cluster Computing 20(7):1–11. https://doi.org/10.1007/s10586-017-0893-5
27. Singh S, Chana I, Buyya R (2020) STAR: SLA-aware autonomic management of cloud resources. IEEE Transactions on Cloud Computing 8(4):1040–1053. https://doi.org/10.1109/TCC.2017.2648788
28. Kivity A, Kamay Y, Laor D, Lublin U, Liguori A (2007) KVM: The Linux virtual machine monitor. Proceedings Linux Symposium 15:225–230
29. Abe Y, Geambasu R, Joshi K (2016) Urgent virtual machine eviction with enlightened post-copy. In: Proceedings of the ACM SIGPLAN/SIGOPS international conference on Virtual Execution Environments (VEE). Atlanta, pp 51–64
30. Jalaei N, Safi-Esfahani F (2021) virtual CPU scheduling for Post-copy live migration of virtual machines. Int J Inf Technol 13(5):239–250
31. Li H, Zhu G, Cui C, Tang H, Dou Y, He C (2016) Energy-efficient migration and consolidation algorithm of virtual machines in data centers for cloud computing. J Computing 98(3):303–317. https://doi.org/10.1007/s00607-015-0467-4
32. Li H, Zhao Y, Fang S (2020) CSL-driven and energy-efficient resource scheduling in cloud data center. J Supercomputing 76(1):481–498. https://doi.org/10.1007/s11227-019-03036-9
33. (2018) Amazon EC2 X1e Instances. https://aws.amazon.com/cn/ec2/instancetypes/x1e/. Accessed 17 Oct 2018
34. Riteau P, Morin C, Priol T (2011) Shrinker: improving live migration of virtual clusters over WANs with distributed data deduplication and content-based addressing. In: Proceedings of the 17th International Conference on Parallel Processing and Distributed Computing. Bordeaux, pp 431–442
35. Zaw EP, Ni LT (2012) Improved live VM Migration using LRU and splay tree algorithm. Int J Comput Sci Telecommun J 3(3):1–7
36. Zheng J, Ng T, Sripanidkulchai K (2011) Workload-aware live storage migration for clouds. In: Proceedings of the ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. ACM, New York, pp 133–144
37. Wood T, Shenoy P, Ramakrishnan K, Merwe J (2015) CloudNet: Dynamic pooling of cloud resources by live WAN migration of virtual machines. IEEE ACM Trans Netw 23(5):1568–1583

Publisher's Note