VMCSnap: Taking Snapshots of Virtual Machine Cluster with Memory Deduplication

Yumei Huang, Renyu Yang, Lei Cui, Tianyu Wo, Chunming Hu, Bo Li
School of Computer Science and Engineering
Beihang University
Beijing, China
{huangym, yangry, cuilei, woty, hucm, libo}@ act.buaa.edu.cn

***Abstract* —Virtualization is one of the main technologies currently used to deploy computing systems due to the high reliability and rapid crash recovery it offers in comparison to physical nodes. These features are mainly achieved by continuously producing snapshots of the status of running virtual machines. In earlier works, the snapshot of each individual VM is performed independently, ignoring the memory similarities between VMs within the cluster. When the size of the virtual cluster becomes larger or snapshots are frequently taken, the size of snapshots can be extremely large, consuming large amount of storage space. In this paper, we introduce an innovative snapshot approach for virtual cluster that exploits shared memory pages among all the component VMs to reduce the size of produced snapshot and mitigate the I/O bottleneck. The duplicate memory pages are effectively discovered and stored only once when the snapshot is taken. In addition, our approach can be not only applied to the stop-copy snapshot but also to the pre-copy mechanism as well. Experiments on both snapshot methods are conducted and the result shows our method can reduce the total memory snapshot files by an average of 30% and reach 63% reduction of the snapshot time compared with the default KVM approach with little overhead of rollback time.**

Keywords —Cloud computing, Virtual Cluster, Snapshot, Memory Deduplication

I. INTRODUCTION

In recent years, Cloud computing has experienced rapid growth as it promises to reduce the maintenance and management costs. Virtualization is prevailing and becomes one of the main technologies used for improving resource utilization with minimal management effort in Cloud datacenters [1]. The applications are encapsulated into virtual machines (VMs) sharing the same underlying physical infrastructure. In particular, with the prevalence of Cloud computing and cluster paradigm [21], VMs distributed on different physical machines can be connected and coordinated within a virtual machine cluster (VMC) by leveraging virtual network construction techniques. In this way, heavy tasks and distributed applications can run on the VMC which provides an isolated, scalable execution environment. For example, Amazon EC2 [18] provides load-balanced web farm which can dynamically add or remove VMs in a VMC in order to maximize the resource utilization. In addition, with the benefits (fast development, re-usability) from SOA, many online applications are composed of a large number of individual services as well as data components encapsulated into isolated VMs. The holistic virtual machine cluster is integrated to provide high-level functionalities. However, there are lots of challenges need to be handled. Due to the frequent failures occurred in the individual components, the system tends to crash frequently. Distributed snapshot [15-16] is a critical technique to improve the system reliability for distributed applications. It saves all the running states of virtual cluster, including the CPU, memory, disk, network and other devices states etc. Once a system failure or error occurs, the system can be resumed to a recorded state according to the snapshot file rather than to the initial state, thus reducing the loss of outage. In particular, the distributed snapshot should be performed periodically during the failure-free execution for the trusted systems. Hence, how to efficiently resume the whole virtual cluster system is a harsh question to be answered [13].

Different approaches have been proposed to produce the snapshot for a distributed virtual cluster. VNSnap [15], Emulab [16] and HotSnap [17] are presented with the aim to support snapshots for a closed VMs network. However, those works focus on how to preserve and coordinate the global consistency of the VM executions and communication states. Additionally, those snapshot approaches are conducted for each individual VM independently without considering shared resources, which is inefficient and produces a great amount of data redundancy. In fact, as the VMs in a VMC are usually deployed with the same OS and applications, many memory pages, libraries and application codes are shared. The same content pages shared by different VMs are typically saved redundantly during each individual VM snapshots. Consequently, the storage cost for snapshot files will increase significantly especially when conducting frequent and periodical snapshots for a large cluster. Furthermore, the storage redundancy may incur extremely fierce disk I/O operations, which leads to a relatively long snapshot time.

To mitigate the frequent disk I/O and reduce the large amount of reduplicative data stored in the snapshot system, we introduce a collaborative snapshot approach for virtual cluster in this paper. In our approach, the snapshot for each VM is taken by exploiting shared memory pages among all components in VMs. More specifically, a hash-based memory de-duplication method is adopted in which most of the same content pages can be detected and stored for only once. It is beneficial not only to the stop-copy snapshot mechanism but the pre-copy live VM snapshot as well. Moreover, we classify memory pages into different categories and present

978-1-4799-3616-8/14 $31.00 © 2014 IEEE
DOI 10.1109/SOSE.2014.45

corresponding storage strategies respectively with a distributed file storage schema, avoiding the bottleneck problem in single node storage. The redundant data will not be stored twice and the states of virtual cluster can be roll-backed precisely. We perform experiments on KVM to evaluate the efficiency of our work in terms of the snapshot file size, the snapshot time, roll-back time as well as the overhead brought to applications. The experimental results demonstrate that the total memory snapshot files can be reduced by average 30% and at least 63% reduction of the snapshot time compared with the default KVM approach with a little overhead roll-back time. In particular, the main contributions of this paper are:

- A memory de-duplicated approach which can efficiently detect the same memory pages among VMs in virtual cluster.
- An effective distributed snapshot storage mechanism based on different categories of memory page to reduce the size of produced snapshot files.

The remaining sections are structured as follows: Section 2 presents the methodology. Section 3 describes the performed approaches and implementations; Section 4 shows the experimental results; Section 5 discusses related work; finally, Section 6 presents the conclusions and discusses future work.

II. METHODOLOGY

The principle of the memory snapshot is to save the overall memory pages of the VM to the disk. Our main idea is to conduct memory de-duplication while taking snapshots, which guarantee the pages with same content will be stored entirely only once. The whole process is illustrated in Fig. 1. To achieve the memory de-duplication among VMs in a virtual cluster, not only the snapshot procedure but also the storage format should be proposed.

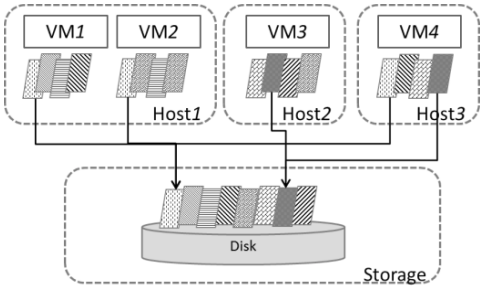

Figure 1. Memory deduplication scenario

A. Memory Pages Identification

In order to de-duplicate the redundant memory pages among all the memory, identical pages need to be initially recognized. However, the total amount of memory pages is significantly large due to the huge impact of the memory size of each VM as well as the number of VMs in the virtual cluster. Therefore, to discover the pages with same content efficiently from a great amount of pages is significantly essential. We firstly compare the hash value of two pages. If their values are the same, the second step is taken for the further comparison of the overall memory page contents byte by byte. Otherwise, the step above can be omitted. The reason for this is because two pages can be determined discrepant if their hash values are different. Although there is a chance that pages with identical hash value may be different, the probability of this phenomenon is quite low, only accounting for less than 1% which can be nearly neglected for the sake of efficiency improvement.

B. Hash Router Mechanism

We design a hash router mechanism implemented by a global hash table shared by all the hosts to store all saved page information. The information includes the hash value and the corresponding file name that actually store the memory page and the offset in the file, through which we can get the page content. Before a page being saved, we calculate the hash value and then search the hash table using the calculated hash value to find whether there has already been an identical value or not. If not found, we update the table by adding the corresponding information.

Furthermore, the size of the hash table will drastically increase with the proceeding of the snapshot. Due to the fact that the hash information is shared by all hosts, if each of these hosts keeps a copy of the hash table, the memory consumed for the hash table will be exorbitant which will bring about additional overhead to keep the consistency among all hash tables. Another alternative is to maintain the hash table on a single host. In this case, all the query and update requests will be sent and accumulated onto this host, giving rise to severe request bottleneck and great resource overhead. To solve the problem, we separate the holistic hash table into several sub-tables distributed on different hosts. Each host shares a Bulletin Board which records the sub-tables distribution information and an example can be illustrated in Table 1.

TABLE I. BULLETIN BOARD EXAMPLE.

Hash Value	Host Address
0x00000000~0x3fffffff	Host1 192.168.1.101
0x40000000~0x7fffffff	Host2 192.168.1.102
0x80000000~0xcfffffff	Host3 192.168.1.103
0xd0000000~0xffffffff	Host4 192.168.1.104

C. An Effective Distributed Storage Format

Without the consideration of de-duplication, the contents of the shared memory page will be separately stored in one or more locations spreading among different VM snapshot files. It will create a great amount of redundant disk space which leads to a relatively low efficiency. Therefore it is necessary to design a novel storage format to store the memory state in order to reduce the size of produced snapshots. Usually the shared memory will only be stored in certain snapshot file and other snapshot files which shares the same memory page just need to store the relevant reference information including both the storage file name and offset address. Through this indexed information, the content can be easily found and a tremendous disk reduction can be reached because the reference information only occupies 16 Bytes in comparison with 4KB holistic memory page size. The proposed storage approach can be illustrated in Fig. 2.

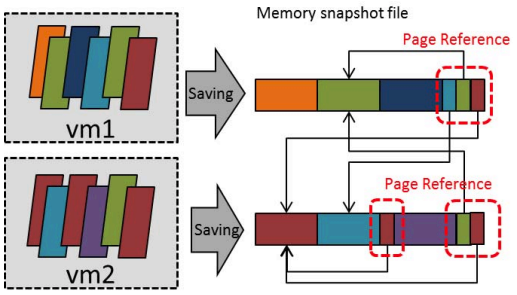

Figure 2. Memory snapshot storage mechanism

Based on our previous observations, we firstly analyze and classify memory pages into three categories: single-byte duplicate pages, ordinary duplicate pages or new pages. The first type is the one in which all the bytes are the repeated contents while the ordinary duplicate page refers to the page whose content can be found the same as a specified page that has been saved before. If the page is neither a single-byte duplicate page nor a duplicate page, it can be marked as a new page. We have designed different storage strategy for different page types. More concretely, only the repeated byte needs to be stored for the single-byte duplicated page. As for the duplicate pages, the index information rather than the page content will be stored which can significantly reduce the disk space consumed. The whole page content has to be saved to the snapshot file with respect to the new page. In addition, the flag which indicates the type of pages is stored together with page content to ensure the correctness and efficiency of roll-back operations. More specifically, the format of storage strategy can be demonstrated in Fig. 3. RAM_SAVE_FLAG_COMPRESS, ADDRESS and PAGE represent the byte-duplicate page, same-content page and new page respectively.

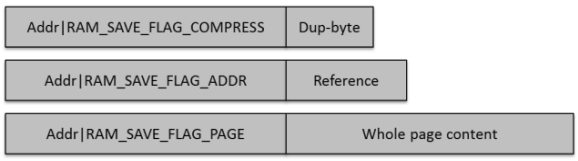

Figure 3. Memory snapshot storage format

Our memory de-duplication approach can be applied to not only stop-copy snapshot but the pre-copy snapshot as well. There are some differences between these two snapshot ways in terms of the storage schema. In stop-copy way, all the VM memory pages are only saved once when taking snapshot. In contrast, in pre-copy snapshot, the dirty pages need to be stored more than once due to the iterations before the dirty pages are eventually synchronized. During this synchronization, the memory page tends to become dirty many times. If the snapshot is performed and written into different file block every iteration round, extremely large disk space and plenty of time have to be spared when saving all the states for each round. Instead, we just conduct memory de-duplication for the first iteration and save the holistic memory page directly for time saving. Once pages modified in the next few iterations, the latest disk space will be overwritten and re-used. In this way, both the time and space consumed will be cut down significantly.

III. IMPLEMENTATION

In this section we present how we implement the proposed methods mentioned in the last section.

We implement a collaborative snapshot prototype based on memory de-duplication by extending KVM snapshot mechanism. The architecture of our snapshot system can be observed in Fig. 4. The virtual cluster is combined with several VMs which spread over different physical machines. The snapshot module and roll-back module are implemented in the QEMU, which runs in the user space.

As shown in Fig. 4, the system is composed of four components: memory snapshot module, memory de-duplication module, communication module and storage module. As the global hash table is distributed among different hosts, query operations for hash table on remote hosts are achieved via socket mechanism. This work is fulfilled by communication module which accepts the hash value passed from the memory snapshot module and subsequently searches the corresponding host. Moreover, the memory de-duplication module determines the data that is needed to be stored in the snapshot file based on the search result send by the communication module. The more detailed process for the distributed snapshot can be illustrated as follows: The first step is that a snapshot synchronization command is sent to all the VMs in the virtual cluster (operation 1) followed by the communication establishment (operation 2). Afterward, the memory snapshot module begins to traverse the whole VM memory pages, computes each hash value and sends the hash value to the communication module (operation 3). After the completion of searching hash table, the result will be send to

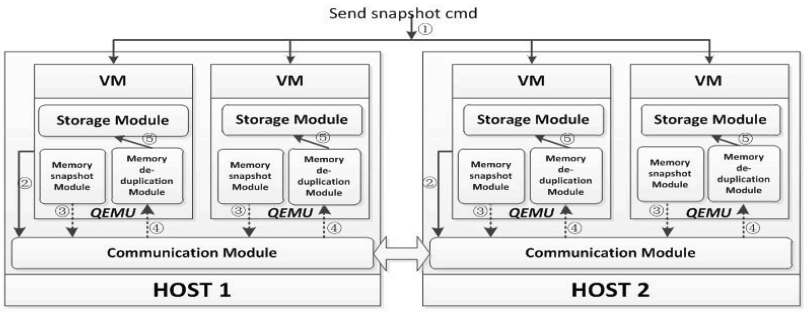

Figure 4. System Architecture

the memory de-duplication module which will produce the data according to the result (operation 4). The memory de-duplication module will subsequently send the data to its storage module to complete the storage for the page (operation 5).

As the hash table is distributed on multiple hosts mentioned in Section 2, the query requests for the hash table are sent through socket mechanism. For each hash sub-table, there is a socket server thread serving for it. The socket server thread receives the search request and returns the result after searching the local hash sub-table.

According to the range of hash value and the number of hosts, the entire hash table is divided into several sub-tables with equal size. The Bulletin Board records the distribution of the hash table which has been discussed in Section II. Upon receiving a query request, the thread in communication module will look up the bulletin board to find the destination host where the sub-hash table is maintained.

Based on our snapshot design and mechanism, the roll-back of the virtual cluster can be conducted when system resuming. As all the raw page data is stored in its snapshot file, the roll-back daemon read the snapshot file sequentially. According to the format illustrated in Figure 3, we can extract the page address, type flag as well as the content from the file. For a single-byte duplicate page, the page is loaded by filling the whole page. But for the duplicated page, we firstly get the page reference by reading the next 16 bytes and calculate corresponding location. Afterwards, the page content can be loaded by reading the next 4096 bytes.

IV. Experiments and Evaluation

Our approach can be applied to both stop-and-copy snapshot and live snapshot. We conducted a set of experiments to demonstrate the efficiency of the system design.

A. Experimental Environment

We conduct the experiments on three physical servers and each of them is configured with 8 core Intel(R) Core(TM) i7 CPU 2.93GHz processor and 16GB memory. The servers are connected via switched Gigabit Ethernet. All the servers are used to be a nfs server where the VM images and snapshot files is located. Two of them are utilized to run VMs and take snapshot. The operating system on physical servers is debian6.0 with 2.6.32-5-amd64 kernel. In addition, we allocate 1GB memory for each virtual machine and adopt qemu-kvm-0.12.5 as the virtual machine manager. We set up the experiments in different scales: 2, 4, 6, 8, and 10 virtual machines in a virtual cluster over the servers respectively. Each VM has its isolated memory snapshot file stored in nfs servers.

Three typical workloads are capsulated into the virtual machines and used as the benchmarks to measure the performance of different snapshot mechanisms:

Idle workload means the VM does nothing except for the tasks inside OS itself after it booting up.

Kernel Compilation is a development workload involving memory and disk I/O operations. We compile the Linux 2.6.32 kernel along with all modules.

Distcc [14] is a compilation tool that distributes the compilation tasks across the VMs in the virtual cluster. It is composed of one Distcc client and a couple of servers. The client distributes the tasks to servers, which will complete the tasks and send the results to client. We use Distcc to compile the Linux kernel with all modules.

We compare our stop-and-copy snapshot method with the default snapshot of KVM, and compare the live snapshot method with VNSnap[15]. The comparisons are made in the following three aspects: the size of snapshot files, snapshot time and roll-back time.

B. Impact on Stop-copy Snapshot

In this experiment, we compare our method with the default snapshot of QEMU/KVM. The virtual cluster snapshots are conducted when the cluster is idle and running a Distcc compilation tasks. The time to complete the snapshot and the memory snapshot file size of each VM are recorded. We perform these experiments several times to collect and calculate the average value in order to reduce error margins.

Fig. 5 describes the reduction ratio of the snapshot file under different cluster scales in idle status and Distcc benchmark running scenarios respectively. Apparently, a significant reduction on snapshot size can be achieved using our approach and the reduction rate is enlarged with the increase of VM number. The reason for this is because the larger the VM number of the cluster is, the higher probability of finding the same-content pages in other VMs will have. The

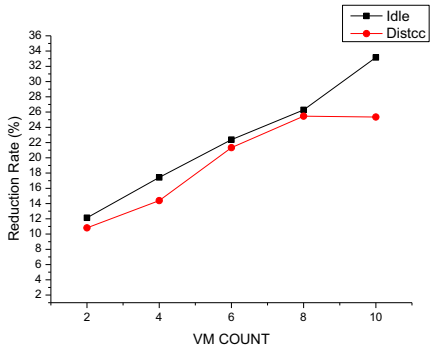

Figure 5. the size reduction of stop-copy snapshot file

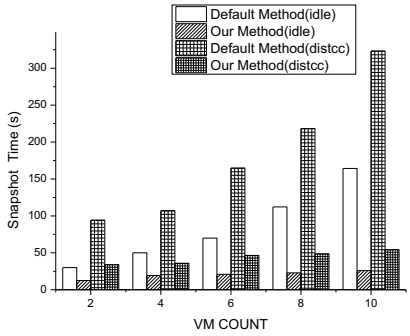

Figure 6. the total snapshot time of stop-copy snapshot whenVMs are idle and running Distcc

reduction percentage can even reach about 34% when ten VMs are co-allocated. In addition, the reduction degree in idle case is larger than that of the workload executed case in all VM scales. This is due to the fact that a large amount of free pages contribute to efficient snapshot compression. This also indicates that it is better to take snapshot when the whole cluster is in idle for the space reduction consideration.

As shown in the Fig. 6, the snapshot time taken by our method is reduced by at least 63% compared with that by default approach no matter which scenario the experiment is conducted and the reduction rate even increases very dramatically when the virtual cluster scales up. There are several reasons to explain this phenomenon. Firstly, the default method saves the memory state in its image file. Due to the qcow2 format used in the VM image, two levels of tables will be traversed when data is written to the image file, producing a non-negligible time overhead. In contrast, in our proposed approach, the VM memory state is saved in a separate binary file where the data can be stored directly. Furthermore, the data stored in memory snapshot file by our snapshot method is much less than the default method owing to the de-duplication strategy, which will eventually result in the less I/O disk operations. The overhead brought by extra memory de-duplication can be omitted considering the tremendous time saving.

Fig. 7 shows that our method has an slight overhead on rollback time than the default method but it can be limited into few seconds. This overhead is mainly brought by locating the reduplicate pages. When the VMs number increases, the rate of reduplicate pages becomes higher, resulting in a slightly obvious rollback overhead.

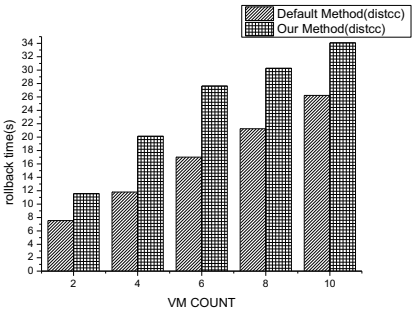

Figure 7.the total rollback time of stop-copy snapshot when VMs are idle

It can be also observed that there are still some discrepancies between the cluster in idle and the cluster running Distcc, although they share an extremely similar trend. Obviously, both the snapshot time and rollback time are longer when running the Distcc benchmark. It is mainly because that free pages of VMs when running application are much less and the size of its snapshot file is larger compared with the one in idle case which will bring about a longer snapshot and rollback time. Additionally, we can draw a conclusion that the effect of memory de-duplication in idle cluster outperforms the cluster running Distcc. This suggests that snapshot should be taken when VMs are idle if possible.

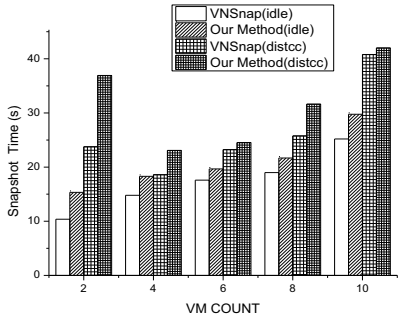

Figure 8. the total snapshot time comparison of live snapshot whenVMs are idle and running Distcc

C. *Impact on Live Snapshot*

To demonstrate the efficiency of our method on live snapshot, we implement the strategy of VNSnap and compare our method against it with respect to the time it consumes. We also take the live snapshots to the virtual cluster in two scenarios: the idle cluster and the cluster running a Distcc compile task.

The snapshot time consumption can be observed in Fig. 8. The VMs running applications usually have a longer snapshot time than the idle VMs because of the longer memory iteration copy time to synchronize the memory states. In particular, in each scenario, as the live snapshot mechanism of both our method and VNSnap approach is based on pre-copy live migration, the snapshot time is mainly determined by the memory iterate copy time which leads to a very similar time consumption. Nevertheless, a marginal overhead is generated by adopting our approach when taking snapshot time because duplicated pages discovery and page contents replacement will take some time. However, this slight overhead can be tolerated considering the greatly reduced disk capacities by data de-duplication in comparison with the VNSnap schema.

D. *Impact on Performance*

Figure 7 depicts the compilation time during continuous distributed snapshot as the number of VMs in the virtual cluster increases. It is observable that the completion time increases if the snapshot is conducted. The increment is derived from the snapshot overhead such as I/O operations and CPU utilization as well as the VM downtime brought by taking snapshot. Compared with VNSnap approach, the

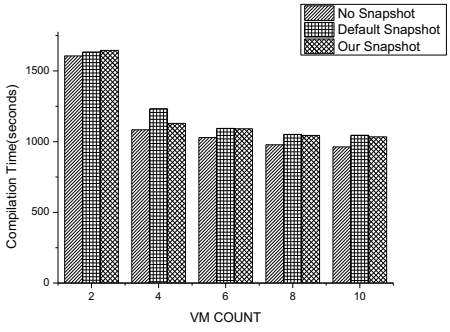

Figure 9. the size reduction of stop-copy snapshot file

compilation time is slightly reduced in our approach despite of the marginal decrement.

V. Related Work

Since the emergence of virtualization technology, as one of the most important features of VMs, snapshot has received a lot of attention in the research community. There has been a large amount of work on virtual VM snapshot. The Linux Logical Volume Manager [9] provides a limited form of copy-on-write snapshots of a block store. Parallax [3] significantly improves on this design by providing limitless lightweight copy-on-write snapshots at the block level. The Andrew File System [7] allows one snapshot at a time to exist for a given volume. In addition, distributed snapshot has been widely studied in the previous works. The earlier works mainly focus on the distributed snapshot protocol to preserve the consistency of system. VNSnap[15] leverages Xen's live migration function to minimize system downtime when taking snapshots, and instantiate a distributed snapshot algorithm to enforce causal consistency across the VM snapshots. Dejavu[12] uses a new runtime mechanism for transparent incremental check-pointing that captures the least amount of state needed to maintain global consistency and provides a novel communication architecture that enables transparent migration. In contrast to optimize the snapshot protocol, we implement a collaborative snapshot approach for virtual machine cluster with memory de-duplication.

Eunbyung Park[5] presents a technique for fast and space-efficient check-pointing of virtual machines by eliminating redundant data and storing only a subset of the VM's memory pages. At a checkpoint, these pages are excluded from the checkpoint image. Lei [17] proposed a HotSnap which focuses on system downtime and TCP backoff duration reduction. Traditional snapshot technique for single VM eliminates the redundant data just by compressing all the single-byte duplicate pages. Lots of work about memory redundancy and sharing had been conducted [6, 8, 10, 11, 19] but the traditional distributed snapshot does not consider the memory similarity across the virtual machines and the snapshot for each VM is proceeded independently. Renyu [20] discussed the performance interference without solving the memory sharing problems in co-allocated virtual cluster. In our work, a memory de-duplication operation is added to the above traditional snapshot procedure.

VI. Conclusions and Future Work

This paper presents a virtual cluster snapshot approach that considers shared resources in order to reduce redundant data and mitigate the IO overhead. Our approach identifies redundant memory pages across the entire virtual cluster and maintains references to duplicated pages using a hash router mechanism to perform memory de-duplication. By doing memory de-duplication when taking snapshots of VMs in the cluster, the total size of memory snapshot files are reduced effectively. We implement the approach on QEMU/KVM platform with not only stop-copy snapshot method but also pre-copy snapshot method. The experiment results reveal that we can reduce the total memory snapshot files by average 30% and reach 63% reduction of the snapshot time compared with the default KVM approach. As future work, we are planning to optimize the pre-copy snapshot by memory locality prediction in order to conducting snapshot intelligently while further improving the performance.

Acknowledgments

The work in this paper has been supported in part by the National Basic Research Program of China (973) (No. 2011CB302602), China 863 program (No. 2011AA01A202), National Nature Science Foundation of China (No. 61170294, 91118008, 61272165), and HGJ Program 2010ZX01045001-002-004. We also thank Ismael Solis Moreno from the University of Leeds for his kind suggestions.

References

[1] James E.Smith, et al. "The Architecture of Virtual Machines", IEEE Computer Society, Volume 38, Issue 5, May 2005 Page(s):32 – 38

[2] Snapshot. http://www.snia.org/education/dictionary/s/#snapshot, retrieved November 8,2012

[3] D. T. Meyer, et al. "Parallax:Virtual Disks for Virtual Machines" In Proceedings of the 3th ACM European conference on Computer systems(EuroSys' 08). ACM, 2008

[4] N. Garimella, "Understanding and exploiting snapshot technology for data protection", IBM developer Works, IBM, 2006, http://www.ibm.com/developerworks/tivoli/library/t-snaptsm1/index.html retrieved November 8,2012

[5] E. Park, et al. Fast and space-effcient virtual machine checkpointing. In Proceedings of the 7th International Conference on Virtual Execution Environments(VEE' 11), pages 75–86, Newport Beach, USA, 2011.

[6] D. Gupta, et al. Difference engine: Harnessing memory redundancy in virtual machines. In Proceedings of the 8th symposium on Operating systems design and implementation (OSDI'08), 2008

[7] Howard, et al. Howard, John H., et al. "Scale and performance in a distributed file system." ACM Transactions on Computer Systems (TOCS) 6.1 (1988): 51-81.

[8] S. Barker, et al. An Empirical Study of Memory Sharing in Virtual Machines. In Proceedings of the 2012 conference on USENIX Annual technical conference (ATC'12) USENIX Association, 2012

[9] Lvm2. http://sources.redhat.com/lvm2/.

[10] Zhao W, et al. Efficient LRU-based working set size tracking[J]. Michigan Technological University Computer Science Technical Report, 2011

[11] Miłós, Grzegorz, et al. "Satori: Enlightened page sharing." In Proceedings of the 2009 conference on USENIX Annual technical conference (ATC'09). USENIX Association, 2009.

[12] J. F. Ruscio, et al. "Dejavu: Transparent user-level checkpointing, migration, and recovery for distributed systems." In proceedings of the IEEE International Parallel and Distributed Processing Symposiumv(IPDPS'07). IEEE, 2007.

[13] Cully, et al. "Remus: High availability via asynchronous virtual machine replication." In Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI'09). 2008.

[14] Distcc.http://code.google.com/p/distcc/

[15] Kangarlou, et al. "VNsnap: Taking snapshots of virtual networked environments with minimal downtime." In IEEE/IFIP International Conference on Dependable Systems & Networks(DSN'09). IEEE, 2009.

[16] Burtsev, et al. "Transparent checkpoints of closed distributed systems in Emulab." In Proceedings of the 4th ACM European conference on Computer systems(EuroSys' 09). ACM, 2009

[17] Lei Cui, et al. "HotSnap: A Hot Distributed Snapshot System for Virtual Machine Cluster". In Proceedings of the 27th Large Installation System Administration Conference(LISA'13), 2013

[18] Amazon elastic compute cloud (amazon ec2). Http://aws.amazon.com/ec2/

[19] Yunkai Zhang,et al "CloudAP: Improving the QoS of Mobile Applications with Efficient VM Migration." In Proceedings of the 15th IEEE International Conference on High Performance Computing and Communication(HPCC 2013), pp. 1375-1381. IEEE, 2013.

[20] Renyu Yang, et al. "An Analysis of Performance Interference Effects on Energy-Efficiency of Virtualized Cloud Environments" in Proceeding of the 5th IEEE International Conference on Cloud Computing Technology and Science(CloudCom 2013), pp. 113-119. IEEE 2013

[21] Xicheng Lu, et al. "Internet-based Virtual Computing Environment: Beyond the data center as a computer." Future Generation Computer Systems (FGCS 2011).