







Global Snapshot File Tracker

Carlos E. Gómez^{1,2}(✉) , Jaime Chavarriaga¹ , David C. Bonilla¹ ,
and Harold E. Castro¹ 

¹ Universidad de Los Andes, Bogotá, Colombia

{ce.gomez10, ja.chavarriaga908, dc.bonilla10, hcastro}@uniandes.edu.co

² Universidad del Quindío, Armenia, Colombia

Abstract. Desktop clouds offer cloud computing services on desktops, simultaneously with users in interactive sessions. Users can affect the virtual machines execution for several reasons. For example, a user can turn-off or reboot the physical machine, or a user can execute demanding applications. A global snapshot of a distributed system is a fault tolerance strategy. In a previous work, we developed the Desktop Cloud Global Snapshot, which obtains the state of the whole system. In case of failure, it is possible to go back to the stored state and resume execution from that point. To recover the system from a global snapshot, we can use the same physical machines or others, if necessary. For this solution it is essential to have a file management system. As global snapshots are created, the number of files that must be handled grows making their management more complex. This article presents the Global Snapshot File Tracker, a software tool that is responsible for maintaining the record of the files that form the state of each virtual machine from its snapshots, and determining what files are required to replicate the state of the virtual machine if it is necessary to resume its execution on another host. The paper includes the background, the problem statement, the proposed solution, the developed solution, and the functionality and evaluation.

Keywords: Global snapshot · Checkpointing · Virtualbox snapshot
Dependability · Reliability · Fault tolerance

1 Introduction

Desktop clouds such as CernVM [5] and UnaCloud [18] run virtual machines on desktop computers distributed along the private campuses. Using these platforms, researchers may execute scientific workflows where computations are performed on virtual machines (VMs) that run at the same time that programs started by the desktop users. Although these platforms take advantage of the idle resources in these desktops, they are susceptible to service failures derived from the presence of the users. Desktop users, for example, can turn-off or disconnect the computers. Therefore, these platform may result not reliable for scientists trying to perform large processing tasks.

Recent researchers have been focused on supporting fault-tolerance strategies in these platforms. Alwabel et al. [3] proposed resource allocation algorithms that consider node failures, Blomer et al. [4] proposed a Global Filesystem for software and data delivery and Gómez et al. [8] have proposed global snapshots and virtual machine migrations as strategies for fault tolerance in desktop clouds. These solutions require a distributed tracking of the states of the virtual machines and of their snapshots. Regretfully, hypervisor software such as Oracle VirtualBox [16] do not support a distributed management of these information.

This paper introduces *Global Snapshot File Tracker (GSFT)*, our tool to manage the snapshot files of a set of VirtualBox machines. Basically, our solution is able to collect information from the virtual machine control files in each desktop, including both machine metadata (e.g., their name, their unique UUID identifier, and the machine configuration) as snapshot definitions. It analyzes and processes the *.vbox* definition file in order to gather in a single data structure the names of all the files that comprise the state of a VM, that is the *.vdi* virtual disk files and *.sav* files with the state of the memory at the snapshot time. The *GSFT* can be used, for instance, to create a *.csv* file with that information, to facilitate the execution of basic management operations on the snapshot files.

Rest of this paper is organized as follows: First, Sect. 2 includes a background for our work and Sect. 3 describes the problem statement and the research questions for this paper. Then, Sect. 4 explains our proposed solution, including the overview of the VirtualBox snapshots, the answers of the research questions and the motivation for the Global Snapshot File Tracker. After that. Section 5 describes the implementation and Sect. 6 reports the evaluation of our application. Finally, Sect. 7 concludes the paper and suggest the future work.

2 Background

Our solution aims to support desktop clouds that use VirtualBox to perform computations in the desktop computers by tracking the snapshots produced in these machines. This section introduced some concepts relevant to our work, namely, Desktop Clouds, Checkpointing and Global Snapshot Algorithms.

2.1 Desktop Clouds

Desktop Cloud (DC) is a form of cloud computing based on desktop computers [2, 17]. It takes advantage of the idle computational resources on these desktops to manage the execution of fully functional VMs with their operating system and applications at the same time that the applications of users of these computers. DCs are designed to run the VMs without the users perceiving a decrease in computer performance or compromising their security [17].

Figure 1 shows a diagram of a DC. It is possible to observe that each desktop has a capacity occupied by the users and an idle capacity that can be used for the execution of VMs, which can be grouped into clusters forming the desktop cloud.

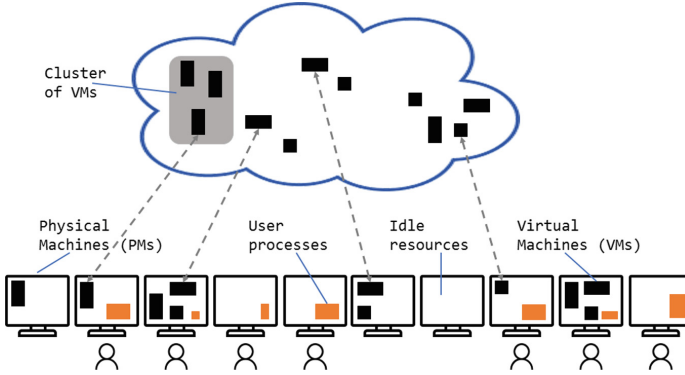


Fig. 1. Overview of a desktop cloud

Although, in a typical campus, the aggregate capacity of the idle resources in the desktops is significant [9], the use of not specialized and dedicated data centers turns desktop clouds more susceptible to failure in the service. Moreover, unlike cloud computing service providers, user presence using physical machines establishes different challenges to overcome: Users of desktops have priority in the use of resources, so their applications can affect the execution of the VMs running on the hosts. In addition, they can even shutdown or reboot the computers interrupting the execution of the VMs. As a consequence, desktop clouds are subject to service failures that do not exist in other platforms.

Recently, many authors have proposed strategies to implement fault-tolerance in desktop clouds: For instance, Alwabel et al. [3] proposed resource allocation algorithms that consider node failures, Blomer et al. [4] proposed a Global Filesystem for software and data delivery and Gómez et al. [8] have proposed global snapshots and virtual machine migrations as strategies for fault tolerance in desktop clouds. These solutions require mechanisms based on *checkpoints* or *snapshots*, to store the state of a VM, to restore its execution in the same or another desktop nodes.

2.2 Checkpointing

Checkpointing saves the state (a snapshot) of a system to resume it from that state when a fail occurs without having to start again from the very beginning. Naturally, if the system is resumed from a checkpoint, nothing performed on the system after the checkpoint was taken is included, so those modifications will be lost. Checkpointing can be full or incremental [1]. The first is the procedure through which every time it is invoked, it saves the complete state of the system. The second saves the system state from modifications after the last checkpoint instead of saving the entire checkpoint. So, it is necessary to keep the files of all previous checkpoints to be possible to resume the execution.

Virtualization-level checkpointing is a functionality (called snapshot) provided by all hypervisors, in which the entire state of a VM is stored in a storage

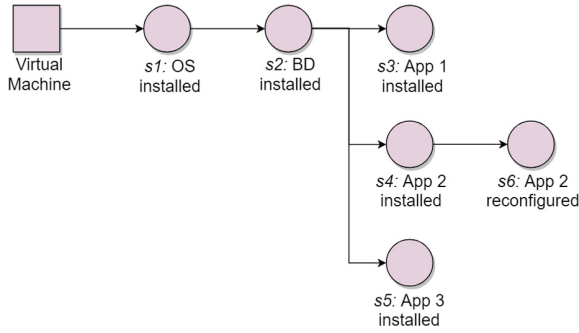


Fig. 2. Sample snapshot tree

medium to form a checkpoint with the possibility of using it later at any time to resume its execution from that state. A VM checkpoint stores the state of all disks attached to the VM, the memory state (if the VM is running when the snapshot is taken), and the metadata needed to configure the VM at the moment of resuming its execution [14].

Hypervisors such as Oracle VirtualBox, VMware Workstation and Microsoft HyperV stores each checkpoint in a *snapshot tree*. In these trees, there is a root snapshot with the initial state of the virtual. The other snapshots are stored with a relationship to their parent snapshot. Each snapshot may be the parent (the base) of one or more other snapshots. The Fig. 2 shows a sample snapshot tree. The root snapshot s_1 has the state when the OS was installed. The s_2 snapshot, based on the s_1 , has the state when a database was installed. Other s_3 , s_4 and s_5 snapshots are based on s_2 and represents the state when different applications was installed. Finally, s_6 keeps the state when one of the applications was reconfigured. When a virtual machine is running, its current state must be based on any of the snapshots in the tree.

Each hypervisor keeps track of snapshots and their dependencies in their own way. For example, Oracle VirtualBox stores the information corresponding to each snapshot in different sections of the `.vbox` file while the VMware Workstation products do it in separate files. Although these hypervisors such can store snapshots of a single VM, they cannot save the state of a system distributed on multiple VMs.

2.3 Global Snapshot Algorithms

A *Global snapshot* is a copy of the state of a distributed system. In desktop clouds, it means a copy of the states of a set of VMs running on different desktops and of the communications among these machines. Creating one of these global snapshots is not a trivial task because it is not possible to assure that all the local snapshots are recorded at the same time [11] and the network communications not necessarily go through a central node [8]. Therefore, to obtain a global state, some coordination among the participants is required [12].

There are many algorithms and techniques used to implement it. For instance, Chandy and Lamport [6] proposed an algorithm to determining global states of distributed systems with FIFO communication channels. Afterward [12], based on Chandy and Lamport algorithm, presented his algorithm for distributed snapshots with non-FIFO channels that, later, Kangarlou et al. [10] simplified it and implemented it using VMs. However, that solution requires customized virtual switches that are not common in desktop clouds settings. More recently, Gómez et al. [8] proposed a global snapshot protocol for distributed systems running on VMs without virtual switches that can be applied to desktops clouds.

Both mentioned solutions, from Kangarlou et al. [10] and Gómez et al. [8], rely on VM snapshots. Any implementation of these solutions in desktop clouds requires to keep a track of the VM snapshots in different physical machines.

3 Problem Statement

In a distributed system, the global state (GS) comprises the set of the *local states* (LS) of each node or process in the system, along with the *channel states* (CS) of the communications among these nodes [6].

Global Snapshot. In a distributed system with n processes, the global snapshot GS comprises the local state LS of each process and the channel state CS of each pair of processes: $GS = \langle \bigcup_{i=1}^n LS_i, \bigcup_{i,j=1}^n CS_{i,j} \rangle$

In a desktop cloud, the local states (LS) may be obtained by requesting to the hypervisor take an snapshot.

3.1 Local State (snapshot) in VirtualBox

As mentioned in the Sect. 2.2, the state of a VM at any given time is represented as a set of files. In VirtualBox, two files are used to create a VM with a single disk: a *.vbox* file with the configuration of the VM and a *.vdi* file with the virtual hard disk. An additional *.vdi* file is required for each additional disk. Each time a snapshot of a VM is taken, the files are created: additional *.vdi* files are used to represent incremental changes on the disks and a *.sav* file is used to store the state of the machine. These files are stored with filenames that do not correspond to the name given to the snapshots.

As an example, Fig. 3, shows the files for a VM when six snapshots. It is not possible to determine which *.vdi* or *.sav* file correspond to each snapshot using only the snapshots. In addition the filenames cannot be used to determine the parent of each snapshot neither. Considering that we are interested on determining the minimal set of files that correspond to an specific snapshot, the file structure do not give use enough information.

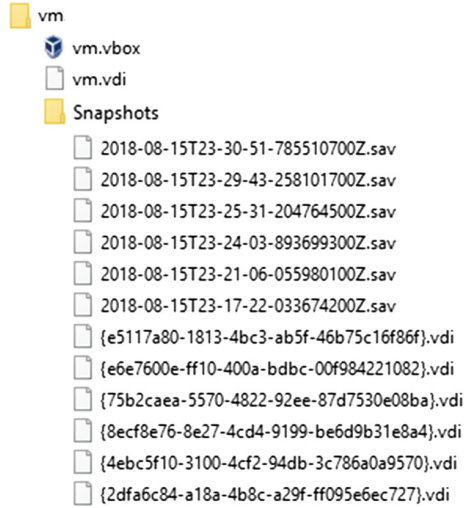


Fig. 3. Files of a virtual machine after six snapshots

3.2 Research Questions

To create tools that support global snapshots on desktop clouds using Virtual-Box, we posed the following research questions:

- RQ1** How to determine and classify the files that make up a virtual machine when multiple snapshots have been obtained?
- RQ2** Of all the files that make up a virtual machine when multiple snapshots have been obtained, which of them are needed to resume the execution of the virtual machine from a specific snapshot?

4 Proposed Solution

The analysis of the *.vbox* file is essential to answer the research questions. This file stores, not only the hardware specifications of the VM, but also the information corresponding to all its snapshots. This section describes the files that are created by VirtualBox when the snapshots are taken and shows the answer to our research questions: how to classify the files of a virtual machine and how to determine the files regarding an specific snapshot.

4.1 Overview of the VirtualBox Snapshots

Oracle VirtualBox offers functionality to obtain snapshots. Figure 4 shows an example of the VM file structure. When creating a VM with a single hard disk (called *vm* in this case), this is represented by two files, which are stored in the



Fig. 4. Evolution of the virtual machine file structure as snapshots are taken

main folder: (1) a VM configuration file, a *vm.vbox* file, and (2) a virtual disk file, *vm.vdi*, as shown in Fig. 4(a).

Figure 4(b) shows what happens when a first snapshot is created. This snapshot consists of the *vm.vdi* file (the initial state of the disk) and a *2018-08-15T23-17-22-033674200Z.sav* file with the memory state. After the snapshot is taken, the disk file is not modified anymore. When the VM runs, an additional file is created to store any change to the hard disk. In our example, the *{4ebc5f10-3100-4cf2-94db-3c786a0a9570}.vdi* keeps the state of the disk. Each *.vdi* file stores the changes regarding the disk of their parent snapshot. Each of these files has information of its parent.

Figure 4(c) shows the files that represent the state of a VM after having obtained the second snapshot. Note that it is not easy to identify which are the two files (*.vdi* and *.sav*) correspond to each snapshot. In addition, in order to use or move one of the snapshots, it is necessary to determine not only the *.vdi* file of that snapshot but also all their parent disk files too. It is not possible to determine the files required to restore an snapshot by using only the filenames.

4.2 RQ1: How to Determine and Classify the Files that Make up a Virtual Machine When Multiple Snapshots Have Been Obtained?

As shown in Figs. 3 and 4, it is not possible to determine which files belong to each snapshot using the filenames. That information can be gathered from the *.vbox* file of the VM. This is an XML file conforming to a XSD schema definition provided by Oracle [15]. There are 112 different elements (tags) that may be used in that XML.

Listing 1 shows excerpts of the *.vbox* file for the VM described in Fig. 3. The *.vbox* file root is a *VirtualBox* tag. The first child element is a *Machine* tag that describes the virtual machine and shows the files with its current state. Among the descendent elements, there is a *HardDisks* tag that describes all the *.vdi* files. Each *HardDisk* can be a children of another *HardDisk*. That hierarchy of *HardDisk* elements shows, for each disk, which other disk is the parent.

Listing 1. Excerpts of the sample vdi.box file

```

<?xml version="1.0"?>
<VirtualBox xmlns="http://www.virtualbox.org/" version="1.14-windows">
  <Machine uuid="{0909f6e6-5527-4782-9f34-ce3e6988ebfd}" name="vm"
    OSType="Debian_64" currentSnapshot="{1020e46e-14bf-4a86-8ea4-d2d093224d3d}"
    snapshotFolder="Snapshots"
    lastStateChange="2018-08-15T23:32:02Z">
    <MediaRegistry>
      <HardDisks>
        <HardDisk uuid="{4a0e0fa2-1cde-4025-8549-dddedb5ce127}"
          location="vm.vdi" format="VDI" type="Normal">
          <HardDisk uuid="{4ebc5f10-3100-4cf2-94db-3c786a0a9570}"
            location="Snapshots/{4ebc5f10-3100-4cf2-94db-3c786a0a9570}.vdi"
            format="VDI">
            :
          </HardDisks>
        </MediaRegistry>
        :
        <Snapshot uuid="{2a0e6ffb-74bf-4402-9bbb-0fc32ca0f911}" name="snapshot1"
          timeStamp="2018-08-15T23:17:22Z"
          stateFile="Snapshots/2018-08-15T23-17-22-033674200Z.sav">
          <StorageControllers>
            <StorageController name="IDE" type="PIIX4" PortCount="2"
              useHostIOCache="true" Bootable="true">
              :
              <AttachedDevice type="HardDisk" hotpluggable="false" port="0"
                device="0">
                <Image uuid="{4a0e0fa2-1cde-4025-8549-dddedb5ce127}"/>
              </AttachedDevice>
            </StorageController>
          </StorageControllers>
        </Snapshot>
        :
        <Hardware>
        </Hardware>
        <StorageControllers>
        :
        <AttachedDevice type="HardDisk" hotpluggable="false" port="0"
          device="0">
          <Image uuid="{2dfa6c84-a18a-4b8c-a29f-ff095e6ec727}"/>
        </AttachedDevice>
        :
        </StorageControllers>
      </Machine>
    </VirtualBox>

```

Below in the `.vbox` file, there are many `Snapshot` tags describing each snapshot. Among all their children elements there is a `StorageControllers` tag, where each `StorageController` can be attached to some media. The controllers uses the UUID (unique id) of the disks to determine which disk is attached. As the `HardDisk` elements, the `Snapshot` elements can be nested to describe the snapshot tree.

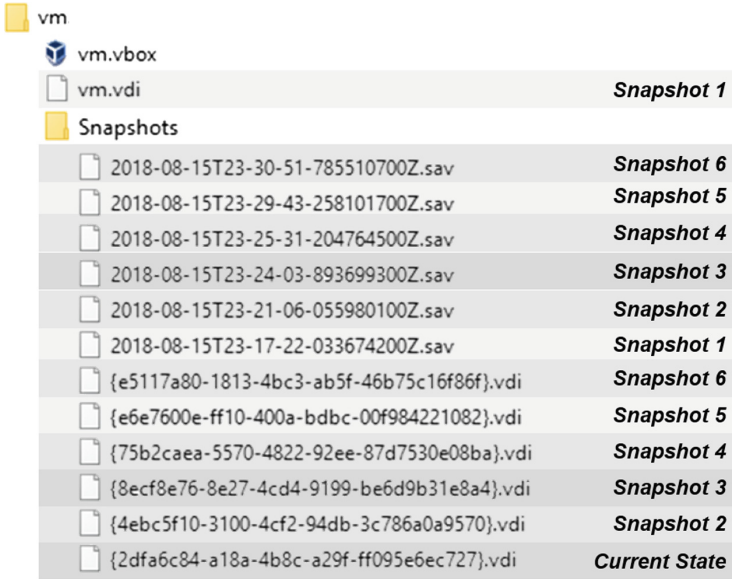


Fig. 5. Meaning of files of a virtual machine after six snapshots

Figure 5 shows the meaning of the files for the sample VM. Although that information cannot be obtained from the filenames, it can be gathered from the *.vbox* file.

4.3 RQ2: Which Files that Make up a Virtual Machine Are Needed to Resume the Execution of the Virtual Machine from a Specific Snapshot?

Not all the files that are part of a VM are needed to resume its execution from the last snapshot:

- .vbox file:** The *.vbox* file maintains the configuration of the hardware for all the snapshots and the current state of the VM. This file is required to resume from any snapshot.
- .vdi files:** Since the *.vdi* files with the disks are differential with respect to its parent, it is necessary to determine all the ancestors in the hierarchy. This applies also for the last snapshot. Considering that the virtual machine may have a snapshot tree with multiple branches, it is possible that many *.vdi* files were not required to restore the last snapshot. Note that the *.vdi* corresponding to the current state of the disk is not required because it stores the changes that occur after the last snapshot.

.sav files: Regarding the state of the memory, the *.sav* file that correspond to the snapshot is the only required. All the other *.sav* files can be discarded or ignored.

Initial state of a VM. The initial state of a VM comprises the *.vbox* file and the *.vdi* files with the main disks: $S_0 = \langle .vbox, .vdi \rangle$

State of a specific snapshot. The state of a specific snapshot comprises the *.vbox* file, the corresponding *.vdi* files, their ancestors *.vdi* files and the corresponding *.sav* file: $S_i = \langle .vbox, .vdi_i, ancestors(.vdi_i), .sav_i \rangle$

5 Implementation

A prototype of the Global Snapshot File Tracker has been developed, which can be found in our repository [7]. It is a stand-alone application, and it is part of the storage system required for the global snapshot solution as a fault tolerance mechanism of a DC. GSFT is responsible for obtaining the necessary information to maintain the record of the names of the files that form the state of a distributed system that runs in a DC. This software can gather all the files required to resume the execution of any snapshots in a VM.

Our solution relies on the Simple API for XML (SAX) in the Java API for XML (JAX) [13] to process the virtual machine definition files. We traverse the structure in the *.vbox* files to obtain relevant and build an object structure with the information of the virtual machines, their snapshots and the corresponding files.

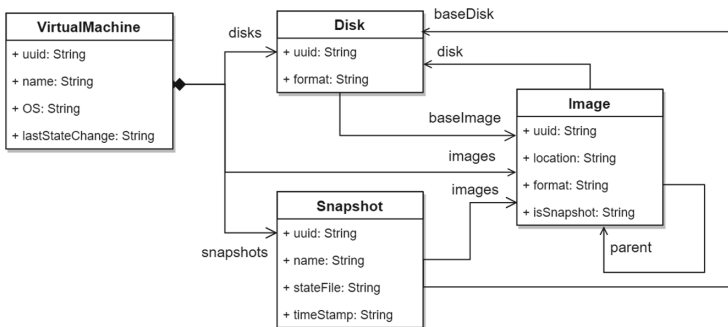


Fig. 6. Excerpt of the class diagram

Figure 6 shows an excerpt of the class diagram. The *VirtualMachine* represents each VM. It has a collection of *Disk* and *Snapshot*. The *Image* class

represents each *.vdi* file. It has relationships that describes which **Disk** and **Snapshot** it belongs. In addition, it has a relationship to their parent **Image**.

We implemented a set of functional tests by processing multiple *.vbox* files. These tests cover multiple scenarios using VMs with different types of snapshot trees and diverse numbers of hard disks.

6 Functionality and Evaluation

This section shows a functionality test and reports the results of our preliminary evaluation. First of all we want to show the functionality of the developed software tool and then we will briefly talk about the time it takes to execute it.

6.1 Functionality

Experiment Design. Given a VM running on host A, we get six snapshots called *SnapshotN*, where *N* is the number corresponding to each snapshot. We want to migrate this VM to two different hosts. In host B, we want to recover the execution of the VM in the last obtained snapshot, while, in host C, we will resume the VM in a specific snapshot, the *Snapshot4*. Figure 7 illustrates the functional test of our system.

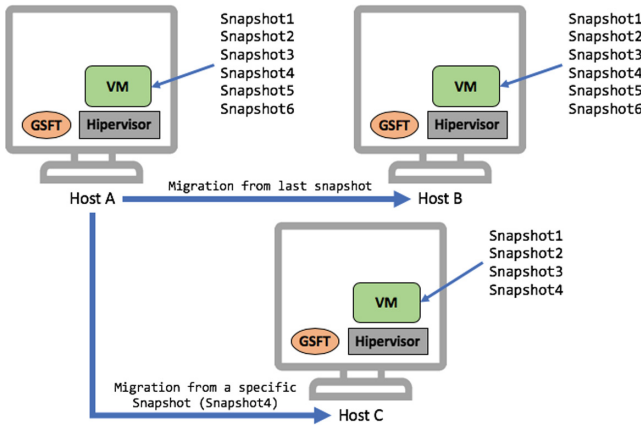


Fig. 7. Functional test of the GSFT System

To correctly resume the execution of the VMs from the mentioned snapshots, we need to know which are the files that should be migrated to the respective host. For this, we are going to use the GSFT system that we have developed.

```
java -jar gsft.jar vm\vm.vbox -f output01.csv
```

Fig. 8. GSFT command to obtain the file structure of a VM

Execution of the Experiment. The GSFT was executed on the *vm.vbox* file, which contains all the information of the VM. To obtain the file structure that make up the virtual machine, we use the command shown in the Fig. 8.

When executing the command, the file *output01.csv* is created with the file structure. In this file we can easily identify the meaning of each file within the composition of the VM, grouping each snapshot into a line. The Fig. 9 shows the result obtained.

Timestamp	Snapshot	Disk	Memory State
2018-08-15T23:17:22Z	Snapshot1	vm.vdi	Snapshots/2018-08-15T23-17-22-033674200Z.sav
2018-08-15T23:21:06Z	Snapshot2	Snapshots/{4ebc5f10-3100-4cf2-94db-3c786a0a9570}.vdi	Snapshots/2018-08-15T23-21-06-055980100Z.sav
2018-08-15T23:24:03Z	Snapshot3	Snapshots/{8ecf8e76-8e27-4cd4-9199-be6d9b31e8a4}.vdi	Snapshots/2018-08-15T23-24-03-893699300Z.sav
2018-08-15T23:25:31Z	Snapshot4	Snapshots/{75b2caea-5570-4822-92ee-87d7530e08ba}.vdi	Snapshots/2018-08-15T23-25-31-204764500Z.sav
2018-08-15T23:29:43Z	Snapshot5	Snapshots/{e6e7600e-ff10-400a-bdbc-00f984221082}.vdi	Snapshots/2018-08-15T23-29-43-258101700Z.sav
2018-08-15T23:30:51Z	Snapshot6	Snapshots/{e5117a80-1813-4bc3-ab5f-46b75c16f86f}.vdi	Snapshots/2018-08-15T23-30-51-785510700Z.sav
2018-08-15T23:32:02Z	{current}	Snapshots/{2dfa6c84-a18a-4b8c-a29f-ff095e6ec727}.vdi	null

Fig. 9. File structure of a VM

To resume the execution of the VM starting from the last snapshot, in host B, we need to know the list of files that must be migrated to be efficient in the use of the network. The software allows us to obtain it by means of the command of the Fig. 10.

```
java -jar gsft.jar vm\vm.vbox -f output02.csv -migrate
```

Fig. 10. GSFT command to obtain the list of files to migrate a VM in the last snapshot

Figure 11 shows the list of files needed to resume execution from the last snapshot. The result of the execution is stored in the file *output02.csv*.

Finally, for the migration to Host C and later resumption in the specific “Snapshot4” snapshot, we can use the command in the Fig. 12.

Again, the result is stored in a *.csv* file, which stores the list of files needed to be able to resume the execution of the VM in the specified snapshot (Fig. 13).

This test allows us to show the service provided by the developed software tool, answering the three research questions mentioned in the Sect. 3.

vm.vbox
Snapshots/{e5117a80-1813-4bc3-ab5f-46b75c16f86f}.vdi
Snapshots/{e6e7600e-ff10-400a-bdbc-00f984221082}.vdi
Snapshots/{75b2caea-5570-4822-92ee-87d7530e08ba}.vdi
Snapshots/{8ecf8e76-8e27-4cd4-9199-be6d9b31e8a4}.vdi
Snapshots/{4ebc5f10-3100-4cf2-94db-3c786a0a9570}.vdi
vm.vdi
Snapshots/2018-08-15T23-30-51-785510700Z.sav

Fig. 11. List of files to migrate a VM in the last snapshot

```
java -jar gsft.jar vm\vm.vbox -f ooutput03.csv -migrate -s Snapshot4
```

Fig. 12. GSFT command to obtain the list of files to migrate a VM in a specific snapshot

vm.vbox
Snapshots/{75b2caea-5570-4822-92ee-87d7530e08ba}.vdi
Snapshots/{8ecf8e76-8e27-4cd4-9199-be6d9b31e8a4}.vdi
Snapshots/{4ebc5f10-3100-4cf2-94db-3c786a0a9570}.vdi
vm.vdi
Snapshots/2018-08-15T23-25-31-204764500Z.sav

Fig. 13. List of files to migrate a VM in a specific snapshot

6.2 Preliminary Evaluation

The evaluation was carried out in order to determine the performance of the tool. The experiment was performed on a host that is usually used by a DC to run VMs. The experiment consisted of processing 10 .vbox files with one to ten snapshots, respectively. We measure the time it took for the system to generate the .csv file with the information of each snapshot, as shown in the Fig. 14

Figure 14 shows the analysis time of .vbox files, which had from one to ten snapshots. The average time for the analysis varies between 15 and 35 ms, with a linear behavior.

7 Conclusions and Future Work

In this tool paper we have presented the *Global Snapshot File Tracker*, a software tool developed to facilitate the management of the files that correspond to all the virtual machines running in all the computers in a desktop cloud. This software determines and classify the files that make up a virtual machine when multiple snapshots have been obtained. This facilitates the recovery of the execution of the virtual machines in the last snapshot or in a specific one. It can be used to

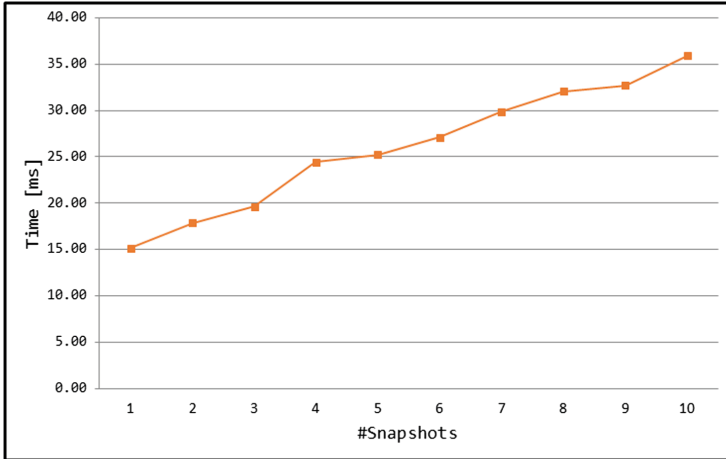


Fig. 14. Average time of analysis of .vbox files

migrate virtual machines by copying the minimal set of files required for a specific snapshot and implement other fault tolerance solutions for desktop clouds.

This paper reports a few functional tests for this application. Here we present the migration of the execution of a VM with six snapshots to other hosts resuming from the last snapshot and from a particular one. In addition, it reports some performance tests. Our public repository [7] has a larger set of functional tests.

As future work we intend to use the global snapshot file tracker within the system for managing files in our global snapshot solution. In addition, we are developing a solution to consolidate snapshots and reduce the number of files that must be copied and move to other machines, Finally, we are using our experience with VirtualBox to support other hypervisors used in desktop clouds, namely VMware Workstation and Microsoft HyperV.

References

1. Agarwal, H., Sharma, A.: A comprehensive survey of fault tolerance techniques in cloud computing. In: 2015 International Conference on Computing and Network Communications (CoCoNet), pp. 408–413. IEEE (2015)
2. Alwabel, A., Walters, R., Wills, G.: A view at desktop clouds. In: International Workshop on Emerging Software as a Service and Analytics (ESaaS 2014), pp. 55–61 (2014)
3. Alwabel, A., Walters, R.J., Wills, G.B.: A resource allocation model for desktop clouds. In: Web-Based Services: Concepts, Methodologies, Tools, and Applications, pp. 356–376. IGI Global (2016)
4. Blomer, J., Buncic, P., Charalampidis, I., Harutyunyan, A., Larsen, D., Meusel, R.: Status and future perspectives of CERNVM-FS. *J. Phys. Conf. Ser.* **396**(5) (2012)

5. CernVM: cernVM software appliance (2018). <https://cernvm.cern.ch/>. Accessed 20 Apr 2018
6. Chandy, K.M., Lamport, L.: Distributed snapshots: determining global states of distributed systems. *ACM Trans. Comput. Syst. (TOCS)* **3**(1), 63–75 (1985)
7. Gomez, C., Chavarriaga, J., Bonilla, D., Castro, H.: Desktop cloud global snapshot (2018). <https://github.com/dc-gs/gsoft>
8. Gómez, C.E., Castro, H.E., Varela, C.A.: Global snapshot of a distributed system running on virtual machines. In: 29th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2017, pp. 169–176. IEEE Computer Society (2017)
9. Gómez, C.E., Díaz, C.O., Forero, C.A., Rosales, E., Castro, H.: Determining the real capacity of a desktop cloud. In: Osthoff, C., Navaux, P.O.A., Barrios Hernandez, C.J., Silva Dias, P.L. (eds.) CARLA 2015. CCIS, vol. 565, pp. 62–72. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26928-3_5
10. Kangarlou-Haghighi, A.: Improving the reliability and performance of virtual cloud infrastructures. Ph.D. thesis, Purdue University (2011)
11. Kshemkalyani, A.D., Singhal, M.: *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, Cambridge (2011)
12. Mattern, F.: Efficient algorithms for distributed snapshots and global virtual time approximation. *J. Parallel Distrib. Comput.* **18**(4), 423–434 (1993)
13. Oracle: JAXP-SAX API documentation. <https://docs.oracle.com/javase/tutorial/jaxp/sax/parsing.html>. Accessed 20 Apr 2018
14. Oracle: Oracle VM VirtualBox: User Manual Version 5.0.20. Oracle Corporation (2016)
15. Oracle: VirtualBox XSD schema definition (2017). <https://www.virtualbox.org/browser/vbox/trunk/src/VBox/Main/xml/VirtualBox-settings.xsd>. Accessed 20 Apr 2018
16. Oracle: Oracle virtualbox (2018). <https://www.virtualbox.org/>. Accessed 20 Apr 2018
17. Rosales, E., Castro, H., Villamizar, M.: Unacloud: opportunistic cloud computing infrastructure as a service. In: *Cloud Computing*, pp. 187–194 (2011)
18. UnaCloud: Unacloud: Opportunistic cloud computing platform (2018). <https://sistemasproyectos.uniandes.edu.co/iniciativas/unacloud/>. Accessed 20 Apr 2018