Cloud-RAIR: A Cloud Redundant Array of Independent Resources

Abdelhamid Khiat

Research Center on Scientific and Technical Information
CERIST - Algiers- Algeria
Email: a.khiat@dtri.cerist.dz

***Abstract*—Cloud computing is considered as a dynamic distributed environment composed of a large number of resources. The Physical Machines (PM) and Virtual Machines (VM) are two main Cloud components. They collaborate together with other Cloud resources to provide a set of services to the end user, who must be satisfied as soon as possible. Unfortunately, the risk of a PM or VM failure is still inevitable in a Cloud environment. To ensure the end user satisfaction, a power fault tolerance technique must be used to avoid the service failures. In this paper, a new VM and PM fault tolerance management mechanism called Cloud Redundant Array of Independent Resources (Cloud-RAIR) is proposed. The *Cloud-RAIR* solution is based on Redundant Array of Independent Disks (RAID).**

Keywords–Cloud Computing; Fault Tolerance; Redundant Array of Independent Disks.

I. INTRODUCTION

The use of Cloud technology has increased enormously in these last few years. Given the benefits offered by this technology, a significant number of services have been migrated to the Cloud environment, which implies a huge need of used resources, including storage space. With this increase in the amount of used resources in the Cloud environment, the probability to get a PM or VM failure also increases, possibly causing a service interruption. The recovery of such a failed PM or VM can be achieved by using a fault tolerance management solution. The latter must avoid any risk of inability to recover the data after a VM or PM crash. One of the most important parameters to take into account in any fault tolerance solution is the used space storage, which should be minimized as much as possible. Such minimization can also help in optimizing other important parameters like cost and consumed energy .

In this work, we propose *Cloud-RAIR*, a reactive fault tolerance management policy based on the powerful concept known as Redundant Array of Independent Disks (RAID) solution. The latter is widely used by most open source operating systems and usually provided as hardware solutions. *Cloud-RAIR* allows to discover and repair the PM and VM failures. The major contribution of the proposed solution lies in the space storage optimization using a specific level of RAID, namely RAID 6.

The rest of this paper is organized as follows. Section II summarizes the related work. In Section III, the proposed solution is described in detail. An evaluation of our solution is presented in Section V. Finally, a conclusion and future work are given in Section VI.

II. RELATED WORK

Two main standards of fault tolerance are defined for Cloud environments, namely Proactive Fault Tolerance Policy and Reactive Fault Tolerant Policy [1]. The first one envisages to avoid failures, while the second one aims to reduce the effects of occurring faults. Considering the nature of our proposed policy, only some Reactive Fault Tolerant policies will be presented in the rest of this section.

In [2], the authors have discussed some reactive fault tolerance approaches, among which we mention:

- Task Resubmission: this technique is based on task resubmission when a fault is detected. The resubmission processes must be done without interrupting the system workflow.
- Check-pointing/Restart: this technique allows to restart the failed Cloud component (application, VM or PM) from a saved state called checkpoint. It is considered as an efficient fault tolerance technique for high computation intensive applications hosted in the Cloud.
- Replication: this technique consists to keep multiple copies of data or object, which will be used when a fault occurrs. According to [2], the replication technique is a popular solution with many varieties.

A collaborative fault tolerance method based on the Check-pointing technique was proposed in [3]. In this technique, both the service consumer and provider participate to ensure the fault tolerance management. According to authors, application faults can be detected and repaired at the customer level, while VM and hardware faults can be detected and repaired at the Cloud provider level.

In [4], the authors exploit the virtualisation by adding a service layer which acts as a Fault Tolerance Middleware (FTM). The added service is inserted between the computing infrastructure and the applications. Then, the proposed FTM can offer fault tolerance support to each application individually.

A Self-tuning Fault Detection system (SFD) was proposed in [5]. It detects faults in the Cloud computing environment. According to authors, SFD has the advantage of ensuring a better fault detection by adjusting fault detecting control parameters.

A framework called *BFTCloud* was proposed by Yilei Zhang et al. in [6]. The authors have used the dynamic replication technique, in which voluntary nodes are selected based on QoS characteristics and reliability performance. Extensive

 ISBN: 978-1-61208-703-0

experiments on various types of Cloud environments show that *BFTCloud* guarantees robustness of systems when f resources out of a total of $3f + 1$ resource providers are faulty.

Redundant Array of Independent Disks (RAID) [7], is the standardized scheme for the design of redundant multi-unit systems. The RAID systems can be provided either as software solutions or as hardware solutions integrated into the computing system. A RAID system allows to enhance fault tolerance through redundancy. A number of standard schemes (levels) have evolved over the years. In our case, we are interested into the RAID 6 level, which consists in block-level striping with double distributed parity. RAID 6 requires a minimum of four disks and provides fault tolerance up to two simultaneously failed drives.

In [8], the authors have combined DRBD [9] and heartbeat [10] solutions to enhance the high availability of the system in the case of resource failure. The proposed architecture was designed for an OpenNebula [11] based Cloud. DRBD was used to ensure distributed replicated storage, whereas heartbeat was used as a high-availability solution.

A typical replication method called K-fault tolerance strategy was proposed in [12]. According to the authors, the service is not largely affected when no more than k nodes fail. In [13], the authors propose an (m, n)-fault tolerance strategy that can ensure (m, n)-fault tolerance and investigate the optimal virtual machine placement strategy.

In order to minimize the number of QoS violations in a fat-tree data center and continue to support the QoS requirement of an application after data corruption, an optimal data replication approach was proposed in [14]. The solution aims to preserve the quality of service requirements after each data crash.

III. PROPOSED CLOUD-RAIR

The main objective of *Cloud-RAIR* consists to ensure the service continuity by detecting and repairing the physical and virtual machine failures. A failure can be a hard one like a storage disk crash, or a soft one like an operating system crash. Note that an application crash is not considered by the *Cloud-RAIR* solution, the latter reacts only when the fault affects VM, or PM components. *Cloud-RAIR* aims to optimize the total used space storage, taking inspiration from RAID, a powerful technique used in a large number of Open-Source operating systems and in hard storage solutions. Among the set of different RAID levels, the RAID level 6 was chosen, given its optimization of the total space storage used to save the data, and its ability to recover data in case of two simultaneous resource (PM, or VM) failure.

The RAID 6 technique recommends to use disks of same size with a total number of used disks that must be at least four. This recommendation is taken into account in our architecture, by dividing the set of VM and PM into sub-sets of the same size, where a VM or PM is the equivalent of a disk in the basic RAID solution. Two types of resources are considered in the *Cloud-RAIR* architecture. The first one is the VM resource type, while the second one is the PM resource type. The set of resources of a given sub-set must be independent; Two resources of type VM are independent if they are not hosted on the same PM, hence the use of the term *Independent Resources* in the name *Cloud-RAIR* (Cloud Redundant Array of Independent Resources). Note that all the resources of the PM type are independent.

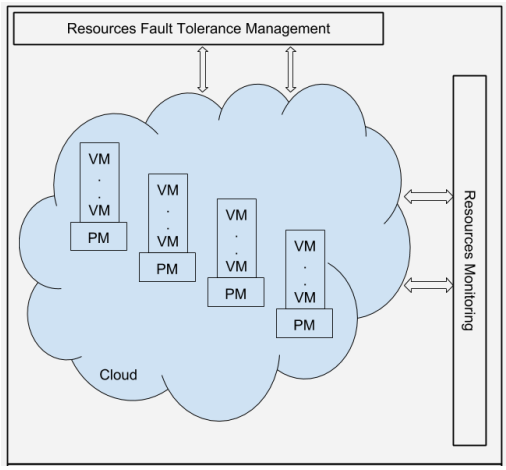

Figure 1. Cloud-RAIR global architecture

As described in Figure 1, *Cloud-RAIR* is composed of two modules that work in parallel and in coordination. The first one ensures a real time Cloud monitoring, in order to detect any event, like resource leaving/joining. The second one ensures the backup management by repairing any resource fault.

From a practical point of view, the two modules can be deployed on any host related to the Cloud. Those modules interact with Cloud components using a Cloud service. Two types of Cloud services are used by the *Cloud-RAIR* solution. The first one is dedicated to provide the informations about the status of a resource, the latter being used by the *Monitoring Resource* module to monitor the status of the resources. Meanwhile, the second one is used to manage the Cloud resources through operations like additions and deletions. This service is used by the *Resource Fault Tolerance Management* module to reconstruct the failed resource.

The global processes of *Cloud-RAIR* are described in the rest of this section.

When a new VM or PM is added to the set of Cloud resources, the addition event is detected by the *Monitoring Resource* module, and an automatic script is triggered by the module, in order to notify the *Resource Fault Tolerance Management* module (RFTM) of their creation. Afterwards, the RFTM module starts by attributing a unique R_id to the new resource and saves all informations about the new added resource. It then uses all saved informations in order to find the best suitable sub-set (SS_id), and subsequently integrates the new resource into that sub-set. Finally, a tuple (R_Type, R_id, SS_id) is constructed as follows:

R_Type: Represents the resource type with several possible values. For example, a type value can be PM to represent PM, VM_Small to represent a Small VM, or VM_Large to represent a Large VM, etc. R_id: Represents a unique id that allows to identify a resource in the system.

SS_id: Represents a unique id used to identify the sub-set that includes R_Id.

VM or PM deletion is another event that can appear. Like a VM/PM addition event, the VM/PM deletion event is detected by the *Monitoring Resource* module, which subsequently launches an automatic script, in order to inform

the RFTM module that a component has been deleted. Then, the RFTM module identifies the deleted resource by its id, removes it from its sub-set and checks if the size of the sub-set after the removal is smaller than four. If that is the case, it deletes the whole sub-set and adds the remaining resources of the sub-set one bye one as new resources without modifying their id.

The third event that can appear is the resource failure. When a resource failed to continue its service, the *Monitoring Resource* module detects the event and notifies the RFTM module to start the process that allows to recover the failed resource. A resource is considered as a failed resource when it does not respond to user requests, which can usually happen if the resource is not reachable.

The proposed *Cloud-RAIR* approach follows the RAID concept. Whenever the Resource Monitoring module detects a PM_i or VM_j failure, *Cloud-RAIR* will not look for a full copy of the lost PM/VM. It will instead reconstruct the lost resource from the sub-set that contains VM_i/PM_j.

In *Cloud-RAIR*, VM and PM are internally coded by a sequence of bits representing an operating system with the users data. Assuming that a sub-set SS_i contains p resources of same size noted $(R_1, R_2, ..., R_p)$, with p equal or greater than four and R_{p-1}, R_p represent the parity values as shown in (1). According to the RAID 6 level, the parity sequences of bits R_{p-1} and R_p represent the data used to recover failed resources and are computed using Formula 1.

$$\begin{cases} R_{p-1} = R_1 \oplus R_2 \oplus ... \oplus R_{p-2} \\ \\ R_p = R_1 \oplus SH^1(R_2) \oplus ... \oplus SH^{p-1}(R_{p-2}) \end{cases} \quad (1)$$

In the formulaes, SH represents the $shift$ function, and a resource R_i is coded as a sequence of bits $R_i = r_0 r_1 r_2 ... r_x$. The corresponding SH function is computed as follows:

$SH^1 = r_1 r_2 ... r_x r_0$, $SH^2 = r_2 r_3 ... r_x r_0 r_1$, etc.

The formulaes 1 and 2 are valid only if $x \geq p$, otherwise, other functions must be applied. In general, $SH^x(R_i)$ represents the shift of R_i by x positions.

For the reconstruction phase, two cases are possible. The first case considers a single resource failure, while the second considers two simultaneous resource failures. In the first case, the reconstruction is done with a simple XOR between all the resources that compose the corresponding sub-set except R_p. Meanwhile, in the second case, the function 2 is applied. In the formulae of the function where k l represent the failed resources. The results of Formula 2 represent a system of $2x$ equations with $2x$ unknowns which uniquely determine the two failed resources R_k and R_l.

$$\begin{cases} R_k \oplus R_l = \bigoplus_{i=0,i!=k,l,p}^{p} R_i \\ \\ SH^{k-1}(R_k) \oplus SH^{l-1}(R_l) = \bigoplus_{i=0,i!=k,l,p-1}^{p} SH^{i-1}(R_i) \end{cases} \quad (2)$$

IV. Description of the Algorithms

Two algorithms can ensure the *Cloud-RAIR* proper system functioning. The first one allows to manage the sub-sets (Figure 2). The second one (Figure 3) allows to recover the failed resource.

```
Input  : VM_i, PM_j
Output: SS_id
Order SS according to Size;
for SS_i ∈ SS do
    SB: for VM ∈ SS_i do
        if VM hosted on PM_j then
            break SB ;
        else
        end
    end
end
if found SS_i then
    Add(VM_i to SS_i)
else
    Create new SS;
    migrate two others VM to the new SS;
    add VM_i to new SS;
end
```

Figure 2. Resource addition algorithm

The algorithm presented in Figure 2 allows to manage the sub-sets over two main setups. In the first setup, the set of sub-sets is sorted in ascending order according to their size, aiming to insert the new VM into the smallest possible sub-set in terms of size. The second setup consists to search the sub-set that does not contain any VM hosted in the same PM with the new VM; if no sub-set is found, a new sub-set is created and two VM are randomly chosen and migrated to the new created sub-set, provided that these two VM are not hosted in the same PM as the new VM. Finally the backup of the new sub-set and the two other altered sub-sets are restarted.

```
Input  : R_1, R_2, ...., R_p
Output: R_failed
if R_failed is PM then
    for R_i ∈ PM do
        Reconstruct(R_i)
    end
else
    Reconstruct(R_failed)
end
function Reconstruct (R_failed)
find SS_id with R_failed in SS_id
Reconstruct R_failed
end function
```

Figure 3. Resource recovering algorithm

Figure 3 presents the algorithm used to recover the failed VM or PM once the failure has occurred. Two cases of failure can appear, the first one consists into a VM failure (VM_{failed}), while the second one consists into a PM failure. In the first case, *Cloud-RAIR* has to identify the sub-set that contains VM_{failed}, then, VM_{failed} is reconstructed using the recovery process defined in Section 1. For the second case, all the VM that have been hosted on the failed PM are identified

and the set of VM are reconstructed, similarly to the case of VM failure.

V. EVALUATION

For the evaluation phase, the JAVA, SHELL and R languages were used to develop a simulator designed to make our experiments. The different experiments have been done by simulation on a personal computer equipped with an Intel Core i5 processor and 6 GB of RAM, using Ubuntu 16.04 as an operating system. *Cloud-RAIR* was implemented using the JAVA programming language, and evaluated by simulation. Our approach was compared with the replication policy.

The replication technique used in the evaluation phase was also implemented in JAVA and deployed with *Cloud-RAIR* in the developed simulator. The implementation of the used replication technique was done as follows: assuming that we have a set of n VM hosted under a set of m PM, the replication consists to make only one copy of each VM (VM_i) on a PM different from the PM that hosts VM_i. Then the OS and data of each PM are copied on another PM.

The following Cloud model was used for the evaluation phase: assuming that the Cloud is composed of a set of m PM and n VM. The whole set of resources (VM and VM) is connected by a private network. The term R is used as a common term to denote a PM or a PM. Each VM is characterized by a tuple ($TypeVM_i$, $SizePM_j$).

A table called $Type\_VM = [\text{"}Type1\text{"}, \text{"}Type2\text{"}, ..., \text{"}Typek\text{"}]$ contains the different types that can be taken by a VM. The variable denoted $TypeVM_i$ is used to designate the type of VM_i, and $TypeVM_i$ is defined at the creation of VM_i and can not be changed during the VM_i life cycle. An associative table denoted $Size\_VM = [\text{"}Type_1\text{"} => Size_1, ..., \text{"}Type_p\text{"} => Size_p]$ contains the sizes of the different VM types, where the size of a VM_i is calculated as follows: $SizeVM_i = Size\_VM[TypeVM_i]$.

The variable $CStorageSize$ denotes the total space consumed by the Cloud excluding the storage space used for the backup. Meanwhile, the variable $CBackupSize$ represents the space used for the backup. The variable $CTotalSize$ represents the total space used by the Cloud ($CTotalSize$=$CStorageSize$ + $CBackupSize$).

For the replication technique, the storage size consumed by the backup ($CBackupSize$) is equal to $CStorageSize$, since, each VM has exactly one copy. Subsequently, the size of each copy of VM_i is exactly equal to $SizeVM_i$. Similarly to VM, each PM haw only one copy, and the PM_j copy size is exactly equal to $SizePM_j$. The total space consumed by the Cloud ($CTotalSize$) is equal to $2 * CStoeageSize$.

Cloud-RAIR assumes that we have p sub-sets denoted SS_i. Following to the concept of *Cloud-RAIR*, all resources of the same SS_i have the same size denoted $RSize_i$. Then the total size of the Cloud is computed using Formula 3.

$$CStorageSize = CStorageSize + (2 * \sum_{i=1}^{p} RSize_i) \quad (3)$$

In order to evaluate and compare *Cloud-RAIR* with the replication policy described above, we assume that we have 5 types of VM according to their size, which are tiny, small, medium, large, and xlarge VM. It is considered that there is no constraint on the available storage space on the PM, that is, the space is sufficient for all VM and backup. For evaluation purpose, the number of PM is varied inside the following set $\{5, 10, 50, 100, 250, 500, 1000, 2500, 5000, 10000, 20000\}$. The number of hosted VM on each PM is randomly generated, and each PM can host between 1 and 20 VM. The evaluation metrics are, the storage space consumed by each policy and the percent of space saved by the *Cloud-RAIR* approach compared with the replication policy.

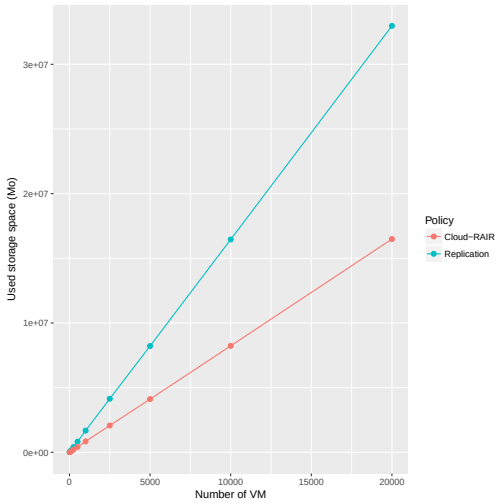

Figure 4. Used space comparison

As previously explained, for the replication approach, the total size of space used for VM backup equals exactly the sum of the sizes of all VM. In contrast, for *Cloud-RAIR*, the storage space depends on the total number of VM and PM, as well as on the way in which VM are distributed on PM. The total space used for backup storage is calculated for each case, as shown in Figure 4.

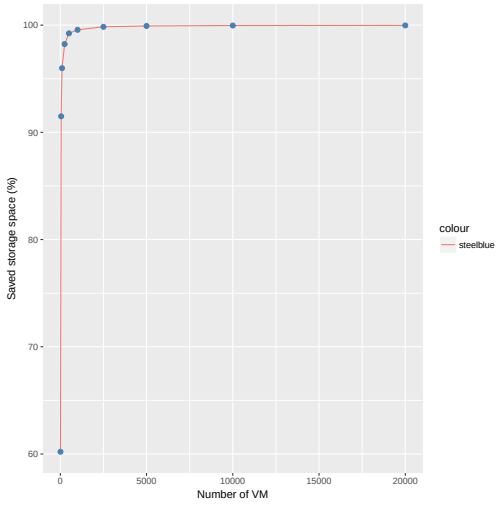

Figure 5. Cloud-RAIR saved backup storage space

Figure 5 shows the percentage of backup space saved using *Cloud-RAIR* compared with the replication method. The saving

becomes more important when the Cloud size in terms of total number of PM and VM becomes large enough. The space saved using *Cloud-RAIR* compared with the replication policy peaks around 99%, where the Cloud size becomes more important. This shows the usefulness of *Cloud-RAIR* when the Cloud is large.

The overall results show the good properties of *Cloud-RAIR* from a theoretical point of view, when the different times taken by the operations are not considered. However, in practice, it is necessary to take into account other parameters, such as communication time. These parameters can potentially have an impact on *Cloud-RAIR* efficiency, particularly when the number of resources is large.

VI. Conclusion and future work

In this paper, a new fault-tolerance policy for managing both virtual and physical machine failures in a Cloud environment has been proposed. As described in this paper, *Cloud-RAIR* inherits its concepts from the RAID 6 technique, consequently, inheriting the advantages provided by RAID 6, particularly, in terms of storage space optimization compared with the standard backup systems, and in terms of the number of simultaneous failures that can be recovered. *Cloud-RAIR* being based on RAID 6 level, it can be useful to adapt the policy of sub-set management defined in this paper with other RAID levels, in order to study the impact of *Cloud-RAIR* solution on the consumed storage space when changing the RAID level.

It will also be useful to introduce the communication time between resources, in order to compute the minimum allowed time between two successive failures. This will allow to predict the maximum allowed failures per unit time. It can also be useful to study the efficiency of *Cloud-RAIR* if the sub-set size is limited, (although this will increase the backup storage size, it will probably allow to reduce the repairing time). Another potential future work to consider, consists in making a real implementation of the *Cloud-RAIR* solution on a real Cloud environment and study its performance in terms of cost and energy consumption. In the proposed solution, the VM and PM failures are considered, but not the application crashes. It could be useful to consider adding the fault management of application level faults.

References

[1] K. Ganga and S. Karthik, "A fault tolerent approach in scientific workflow systems based on cloud computing," in Pattern Recognition, Informatics and Mobile Engineering (PRIME), 2013 International Conference on. IEEE, 2013, pp. 387–390.

[2] A. Ganesh, M. Sandhya, and S. Shankar, "A study on fault tolerance methods in cloud computing," in Advance Computing Conference (IACC), 2014 IEEE International. IEEE, 2014, pp. 844–849.

[3] A. Tchana, L. Broto, and D. Hagimont, "Approaches to cloud computing fault tolerance," in Computer, Information and Telecommunication Systems (CITS), 2012 International Conference on. IEEE, 2012, pp. 1–6.

[4] R. Jhawar, V. Piuri, and M. Santambrogio, "A comprehensive conceptual system-level approach to fault tolerance in cloud computing," in Systems Conference (SysCon), 2012 IEEE International. IEEE, 2012, pp. 1–5.

[5] N. Xiong, A. V. Vasilakos, J. Wu, Y. R. Yang, A. Rindos, Y. Zhou, W. Song, and Y. Pan, "A self-tuning failure detection scheme for cloud computing service," in 2012 IEEE 26th International Parallel and Distributed Processing Symposium, May 2012, pp. 668–679.

[6] Y. Zhang, Z. Zheng, and M. R. Lyu, "Bftcloud: A byzantine fault tolerance framework for voluntary-resource cloud computing," in 2011 IEEE 4th International Conference on Cloud Computing. IEEE, 2011, pp. 444–451.

[7] D. A. Patterson, G. Gibson, and R. H. Katz, A case for redundant arrays of inexpensive disks (RAID). ACM, 1988, vol. 17, no. 3.

[8] C.-T. Yang, J.-C. Liu, C.-H. Hsu, and W.-L. Chou, "On improvement of cloud virtual machine availability with virtualization fault tolerance mechanism," The Journal of Supercomputing, vol. 69, no. 3, 2014, pp. 1103–1122.

[9] P. Pla, "Drbd in a heartbeat," Linux Journal, vol. 2006, no. 149, 2006, p. 3.

[10] D. Bartholomew, "Getting started with heartbeat," Linux Journal, vol. 2007, no. 163, 2007, p. 2.

[11] D. Milojičić, I. M. Llorente, and R. S. Montero, "Opennebula: A cloud management tool," IEEE Internet Computing, vol. 15, no. 2, 2011, pp. 11–14.

[12] F. Machida, M. Kawato, and Y. Maeno, "Redundant virtual machine placement for fault-tolerant consolidated server clusters," in Network Operations and Management Symposium (NOMS), 2010 IEEE. IEEE, 2010, pp. 32–39.

[13] A. Zhou, S. Wang, C.-H. Hsu, M. H. Kim, and K.-s. Wong, "Virtual machine placement with (m, n)-fault tolerance in cloud data center," Cluster Computing, 2017, pp. 1–13.

[14] J. Lin, C. Chen, and J. M. Chang, "Qos-aware data replication for data-intensive applications in cloud computing systems," IEEE Transactions on Cloud Computing, vol. 1, no. 1, Jan 2013, pp. 101–115.

 ISBN: 978-1-61208-703-0