Replication for Dependability on Virtualized Cloud Environments*

Filipe Araujo
University of Coimbra, Portugal
filipius@uc.pt

Raul Barbosa
University of Coimbra, Portugal
rbarbosa@dei.uc.pt

António Casimiro
University of Lisbon, Portugal
casim@di.fc.ul.pt

ABSTRACT

Execution of critical services traditionally requires multiple distinct replicas, supported by independent network and hardware. To operate properly, these services often depend on the correctness of a fraction of replicas, usually over 2/3 or 1/2. Defying the ideal situation, economical reasons may tempt users to replicate critical services onto a single multi-tenant cloud infrastructure. Since this may expose users to correlated failures, we assess the risks for two kinds of majorities: a conventional one, related to the number of replicas, regardless of the machines where they run; and a second one, related to the physical machines where the replicas run. This latter case may exist in multi-tenant virtualized environments only. We evaluate crash-stop and Byzantine faults that may affect virtual machines or physical machines. Contrary to what one might expect, we conclude that replicas do not need to be evenly distributed by a fixed number of physical machines. On the contrary, we found cases where they should be as unbalanced as possible. We try to systematically identify the best defense for each kind of fault and majority to conserve.

Categories and Subject Descriptors

C.4 [**PERFORMANCE OF SYSTEMS**]: Fault tolerance

General Terms

Reliability,Security

Keywords

Cloud computing, Fault-Tolerance, Dependability, Virtualization

*This work has been supported by the project CMU-PT/RNQ/0015/2009, TRONE — Trustworthy and Resilient Operations in a Network Environment.

MGC2012, December 3, 2012, Montreal, Quebec, Canada.

1. INTRODUCTION

To reduce costs, companies increasingly outsource information technology to cloud providers. However, with this movement, they lose much of the control they could exert on their most critical services. Popular strategies like replication may not work well on the cloud, because providers may take advantage of virtualization techniques [6, 5, 3, 7, 9] to concentrate some of (or all) the replicas in the same physical machine (PM). We could think of a cluster of application servers, responding to HTTP requests, or a remote storage service for instance [2, 4, 1].

Common sense will tell us that one should not use a virtualized infrastructure to run replicas of the same service, because a single fault could tear down many replicas at once. Ideally, each new replica should run on a different PM, preferably at distant physical locations, served by different networks, using diverse software, operating systems, and so on. However, in this paper we assume exactly the opposite: for economical reasons, clients will resort to the same cloud provider and the cloud provider may end up using virtualized resources to place these replicas, sometimes using different PMs, other times, inside the same PM. Taking this as the initial assumption, we qualitatively try to mitigate the problems that such option raises, giving a single defense to the cloud provider: the distribution of the VMs by the PMs it has available. Depending on the service, the defender may care for the number of PMs *or* the number of VMs that stand after each fault. In particular, motivated by many consensus-based algorithms, including Byzantine ones, we count the number of attacks that are necessary before the service loses the majority of machines, physical *or* virtual.

We also assume that the attacker is limited (otherwise, the defense would be helpless). We evaluate two kinds of faults: crash-stop faults, where virtual machines (VMs) or physical machines (PMs) stop due to some hardware or software fault; and intentional faults, where some malicious user may hire a service for the single sake of attacking it. In the latter case, we assume that the attack to a single process or VM compromises the VM, the PM where it runs, all other co-located VMs, and any service in the PM as well. Note that the attack or crash of a PM could start on a VM, but we do not really take that detail into account. We focus on the effect and not on the cause.

While intuition suggests that we should always balance the VMs among the PMs, after a deeper analysis we conclude that this is not always the case. Depending on the kind of attack and majority to keep, the distribution of VMs

might be irrelevant or should even be as unbalanced as possible given the *same* number of PMs. In some cases the defender should concentrate the VMs as much as possible: one VM for each but the last PM, and all the other VMs on the last PM. While this is the best way of confining an attacker to the smallest possible number of PMs, we also show that this does not have any impact on the number of correct VMs, whenever faults on a VM do not affect other co-located VMs.

What we would like to know is the best way to distribute v VMs by m PMs ($v > m$). The goal may be to ensure that a majority of v VMs *or* m PMs is correct, to enable operation of some (Byzantine) fault-tolerance protocol, like the ones of Correia *et al.* [11] or Castro and Liskov [10]. Hence, the main contribution of this paper is to define the appropriate strategy to defend a majority from different sorts of faults, including Byzantine. This strategy can be to evenly balance the VMs, to unevenly balance them or, in some cases, there is no best strategy. We think that our study can be useful both for the cloud provider and for the client. For the cloud provider we try to make the limits of the system clearer. This may help the provider to create more precise and safer SLAs. Clients may resort to this study to learn how to distribute their facilities by different cloud providers.

The rest of the paper is organized as follows. Section 2 points to related work. Section 3 presents our assumptions. In Section 4 we do some theoretical analysis regarding the distribution of VMs. Since there is a long and established field using the terms "balls" (VMs) and "urns" (PMs), we often keep these terms. In Section 5 we extend the analysis of the previous section and run Monte Carlo simulations, whenever we are not aware of closed formulas that may help us to find the appropriate distributions. In Section 6 we discuss the best distribution strategies and in Section 7 we conclude the paper.

2. RELATED WORK

Replication is a well-know technique to tolerate faults in distributed computing systems. Distributed object replication can follow a primary-backup approach [**?**], where the primary owns a centralized control of the group; or an active replication, also known as "state-machine approach" [**?**, **?**], where all replicas are similar and control is decentralized. Both approaches require a group of servers running the same application, which cloud providers may be tempted to virtualize. This kind of replication usually accomplishes very strong forms of consistency among the replicas, like linearizability [**?**], or sequential consistency [**?**], to hide the replication from the client. Interestingly, some argue that clouds could never scale using such strongly coupled groups [**?**]. A more scalable use for clouds may consist of growing or shrinking a server farm of stateless servers, without any special concern for consistency. Technologies, like JBoss and GlassFish [**?**, **?**] support this mode of operation, known as "cluster mode". In this kind of scenario, some load balancer will send user requests to some appropriate server instance. Depending on the applications, consistency may be pushed back to the (non-replicated) database level, or may be entirely avoided. We can find similar deployments, both as an infrastructure-as-a-Service, where clients hire additional/fewer machines to enlarge/shrink the cluster, or as a Platform-as-a-Service, where developers just publish the software to the cloud, which is responsible for managing the cluster, depending on demand.

Regarding the specific topic of replication on a cloud, we could mention the very interesting case of Vertical Paxos [**?**], intended for primary-backup replication, which considers the case where the number of replicas is small and grows only as necessary to overcome faulty servers — a process authors call *reconfiguration*. Different clouds to sum matrices. VerticalPaxos

3. MODEL

In this paper we deal with crash-stop and Byzantine faults. We initially assume that malicious users have limited power, due to the money they need to hire services, their inability to pick the right targets, or due to the computational resources necessary for a successful attack. We believe that this adequately reflects the fact that real services may have web-based (or other) interfaces requesting money to let the users in. In particular, we assume that attackers are unable to determine all the cloud provider resources and perform selective attacks to the running services, once they are inside the provider's network. This is the focus of other research work [12]. Hence, we consider that the attacker must go through the cloud provider to access other cloud provider resources, following an apparently normal behavior. Nevertheless, in Section 6 we consider broader classes of attacks.

We consider that the cloud has m PMs. Each of the m machines may run one or more of the v processes that provide some service. We assume that each process runs on its own VM. To make our analysis clearer, we often consider a balls and urns problem: the urn is a PM; the VMs (either guest or host) are the balls. VM failures correspond to drawing a ball from an urn. In some cases, the attacker will put the ball again in the same urn, sometimes he or she will simply remove the ball. One should notice that this is not really the attacker's choice, but a feature of the service. Consider a cluster of HTTP servers running in the cloud. Subsequent requests to the service may end up in the same application server. This would be a case without removal — the attacker may find the same HTTP server on the next attack, or "ball" in the "urns and balls" model. Unlike this if he asks for an extra replica (e.g., an entire VM), then we consider a removal — the attacker cannot find the same VM again.

When a VM crashes, we assume that this does not affect other VMs or the hosting PM, as long as other co-located VMs remain active. In the Byzantine case, we assume that as soon as an attacker takes over a VM, the entire PM, along with its VMs, become compromised, and the cloud provider must no longer rely on them. Depending on the assumption and on the service, we may want to ensure a majority of correct VM *or* PM replicas. In fact, some consensus algorithms, as the one in [11], use special hardware devices, such as a Trusted Platform Module (TPM), to improve the tolerance to Byzantine nodes. While appropriate when there is a one-to-one correspondence between machines and hardware, the existence of multiple VMs sharing the same hardware raises a problem to these algorithms, because they use the notion of round and the TPM may only sign a single message per round. It will thus not work with more than one VM. On the other hand, demanding one TPM per VM seems unreasonable as this would defeat the purpose of virtualizing the infrastructure. It is therefore a natural step to virtualize the TPM itself, as in vTPM [8]. Other possibilities exist

as well: e.g., a single proxy per PM could represent all the co-located VMs. The point is that whenever a PM owns a single function, device or process that is shared by all the VMs, it becomes important to ensure a majority of correct PMs.

Overall, we have crash-stop and Byzantine faults; we also consider Byzantine faults where the attacker can pick his or her targets (3 kinds of faults); faults may occur at the VM or PM level (2 levels); and we may want to preserve VM or PM majority (2 majorities). This makes for a total of $3 \times 2 \times 2 = 12$ cases of interest that we discuss along the paper and wrap up in Section 6.

4. THEORETICAL ANALYSIS

4.1 Independent VM failures

We start our theoretical analysis by considering that faults are independent: the failure of a VM does not affect any other co-located VM. This may fit a crash-stop fault. We also assume that to participate in some replicated protocol, the PM needs to have at least one operational VM, which is to say that the PM becomes unusable as soon as the last VM fails. I.e., Given the failure of $n \leq v$ VMs, we can calculate the probability that a PM with v_i VMs fails. We compute this value in Equation 1, for $v_i < n \leq v$, using the Hypergeometric distribution. $P_{f_i}(n)$ is the probability that PM i fails, after all its v_i VMs have failed.

$$P_{f_i}(n) = \frac{\binom{v_i}{v_i}\binom{v-v_i}{n-v_i}}{\binom{v}{n}} = \frac{n \times (n-1) \times \ldots (n-v_i+1)}{v \times (v-1) \times \ldots (v-v_i+1)} \quad (1)$$

Since this function is convex, the *expected* number of failed PMs is a minimum when $P_{f_1} = P_{f_2} = \ldots = P_{f_m}$. This results from Jensen's inequality [13, Exercise (3.1)]. This means that if the VMs fail independently of each other, one should uniformly distribute them by the PMs.

4.2 VM failure contaminates other VMs

We now consider Byzantine attacks, where a malicious user may take over all the co-located VMs, once he or she successfully attacks the first one. In mathematical terms, we can treat this problem as an urns and balls problem, and use known results, such as [13]. Urns represent a PM, while balls represent the VMs (or processes). To distinguish between correct and compromised VMs we may assign colors to balls: white balls represent correct machines, whereas black balls represent compromised machines. The objective of our analysis in this section is to show that there is a proper way of initially distributing white balls, which minimizes the number of urns that end up having black balls as a result of malicious actions successively changing the color of balls from white to black. This corresponds to letting the cloud provider select a distribution of VMs by the PMs. We care about the number of urns that have no black balls, which we denote by the random variable X. $P(X \geq k)$ is the probability that k or more urns have no black balls.

THEOREM 1. *Assume that we have v white balls distributed by $1 < m < v$ urns, such that each urn has at least one ball. Assume that the attacker works in successive turns, picking one ball at a time from a urn, and always putting back a black ball in the same urn. The attacker does not select the ball or the urn. $\forall\ k \in \{1, \ldots, m\}$, $P(X \geq k)$ is maximized when $m-1$ urns have 1 ball, and the remaining urn has the remaining $v-m+1$ balls.*

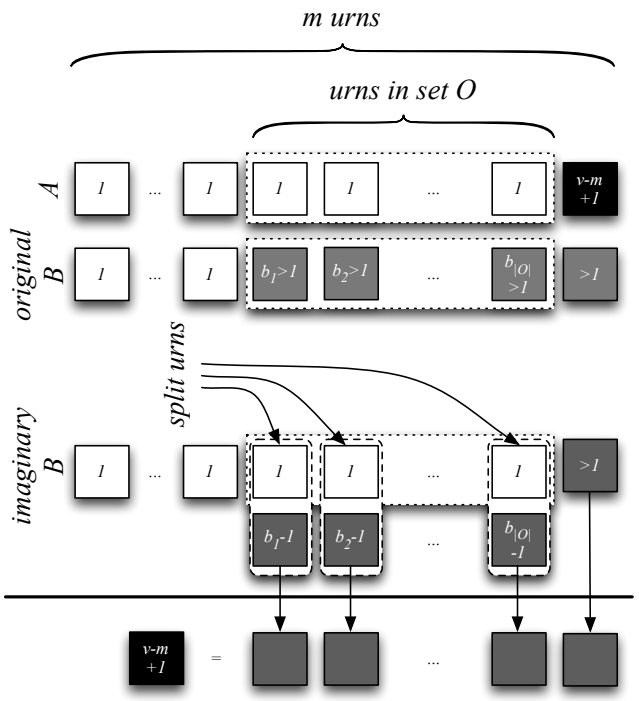

Figure 1: Distributions of balls by urns in settings A and B

PROOF. Refer to Figure 1. Note that white urns have only 1 ball, gray urns have at least 1, but less than $v-m+1$ balls, and black urns have $v-m+1$ balls. Consider setting A, where only 1 urn has more than 1 ball. I.e., $m-1$ urns have 1 ball, whereas the last urn in the figure has $v-m+1$ balls. Setting B represents any other case, with the same number of urns, m, but with a different distribution of balls. Let us match pair-wise the urns in setting A with the urns in setting B (original), starting by the 1-ball urns (on the left side of the figure). After this first set of urns, we define another set O, with the urns that have more than 1 ball in B, but only one ball in A. This definition excludes the first urns that have 1 ball in A and B, as well as the last urn of A, which has more than 1 ball. Note that $1 \leq |O| \leq m-1$.

We now resort to an artificial division of set O in B. We split each one of the $|O|$ urns in B into two other urns: one with 1 ball, and the other with the remaining balls. This makes for a total of $m-1$ urns with only 1 ball, just like in setting A. The remaining $v-m+1$ balls are spread over $|O|+1$ urns (in gray in the "imaginary B"), each having one or more balls. In the rest of our reasoning, we should refer to settings A and "imaginary B", with m and $m+|O|$ urns, respectively. In both cases there are $m-1$ urns with only 1 ball. We first observe that these urns have exactly the same probability of having a black ball. What about the singe black and $|O|+1$ gray urns in A and "imaginary B", respectively? Since the number of balls is the same,

$v - m + 1$, the probability of having one or more black balls is exactly the same in both settings. However, these black balls only "contaminate" (or exist) in a single urn in setting A, whereas in "imaginary B" they may spread over multiple urns. As a consequence, $P(X \geq k)$ in setting A must be the same or greater than in setting B. □

To calculate $P(X \geq k)$, we can use a result from [13, Equation (3.5)], which we restate here:

$$P(X \geq k) = \sum_{\mathbf{a}}^{(k)} \left(1 - \sum_{j=1}^{k} p_{a_j}\right)^n - \\ - \binom{k}{k-1} \sum_{\mathbf{a}}^{(k+1)} \left(1 - \sum_{j=1}^{k+1} p_{a_j}\right)^n + \\ + \binom{k+1}{k-1} \sum_{\mathbf{a}}^{(k+2)} \left(1 - \sum_{j=1}^{k+2} p_{a_j}\right)^n - \\ \ldots \\ + (-1)^{m-k} \binom{m-1}{k-1} \sum_{j=1}^{m} {p_j}^n \qquad (2)$$

The m urns define a set of integers $\{1, 2, \ldots, m\}$. The variable p_j is the probability that we assign a black ball to urn j. From this set, we define subsets with k elements $\{a_1, a_2, \ldots, a_k\}$. $\sum_{\mathbf{a}}^{(k)}$ denotes a summation over all these subsets. Thus, there are $\binom{m}{k}$ terms in this sum. For a better understanding of this formula, we should realize that summation $\sum_{\mathbf{a}}^{(k)}$ concerns the probability of having k urns without black balls when n black balls have been dropped in the urns. The summation that follows in the formula considers the probability of having $k+1$ empty for the same n balls and so on.

4.3 Keeping a Majority of Virtual Machines

The next question we consider is the impact of machine failures on the number of VMs (or processes) that stay alive in a correct state. In many cases, this number might be more important than the number of different PMs where the replicas run. One may ask whether concentrating many processes in the same (or few) machine(s) could reduce the average number of processes that survive (other) PM crashes. Interestingly, the answer is *no*. If we assume that all PMs have the same characteristics, at any given time t, the average number of VMs running is the same, independently of their distribution by the *same* set of PMs. I.e., if processes themselves do not fail, we have:

$$E[Z(t)] = \sum_{i=1}^{m} v_i \cdot P[Y_i(t) = 1] = P[Y(t) = 1]v \qquad (3)$$

where $Z(t)$ is a random variable that represents the number of VMs that are running at time t. Variable v_i is the number of VMs running in machine i, $Y_i(t)$ is a random variable that assumes the value 0 if machine i is off at time t or 1 if the machine is on. Since we assume that this probability is the same for all machines, we can remove the subscript and simply write $P[Y(t) = 1]$. Equation 3 shows that the distribution of VMs (or processes) by the PMs is not relevant from the point of view of the average. However, one should notice that other metrics may be relevant as well: on the few occasions when the most loaded PM fails, much fewer replicas will be available. I.e., the unbalanced setting will most of the times keep a few more processes running, but sometimes, it will have much less.

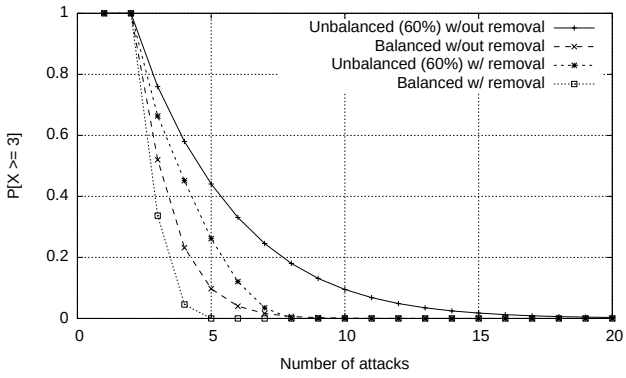

Figure 2: Probability of conserving majority for 5 physical machines

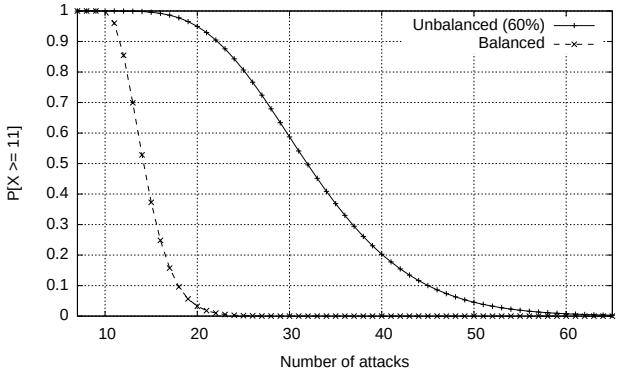

Figure 3: Probability of conserving majority for 21 physical machines

5. EXPERIMENTAL EVALUATION

Since we cannot evaluate all the interesting scenarios using analytical expressions, in this section we partially resort to simulation. To evaluate the likelihood of keeping a majority of correct PMs, we start with $m = 5$ PMs and $v = 10$ VMs in Figure 2. We considered four cases: in one of them, all the machines (urns) have the same probability of receiving an attack (black ball) and this probability remains constant. We call this case "balanced without removal". We also consider the case where a single machine has a probability of 60% of receiving an attack, while the remaining 40% probability is equally distributed among the remaining machines. This corresponds to running 6 VMs in a single PM, while the remaining 4 VMs run in the other 4 PMs, in a one-to-one correspondence. To this case we call "unbalanced (60%) without removal". We also consider removals of the VMs once the attacker dominates them. This corresponds to a scenario where the attacker is successively requesting for some instance of a service from a limited set. Each time it gets a new instance, the attacker will not get the same,

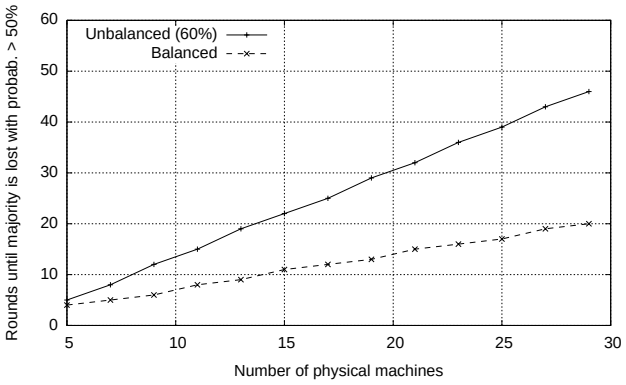

Figure 4: Evaluation of the rounds the attacker needs until it probabilistically gets the majority as a function of the number of physical machines

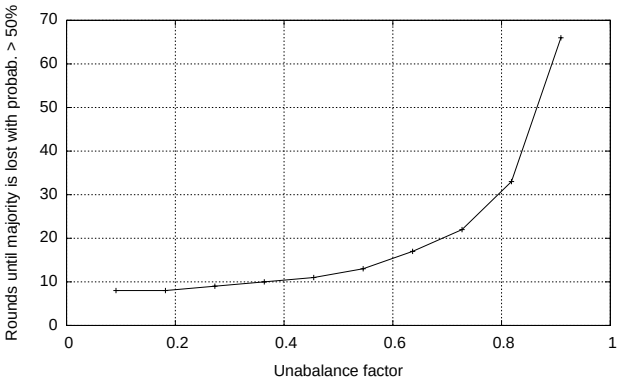

Figure 5: Evaluation of the rounds the attacker needs until it probabilistically gets the majority as a function of the unbalance factor

but a new one (e.g., a new VM). These are the two cases with removal.

Still in Figure 2, in the plots without removal, we use Equation 2 to compute the probability that at most 2 out of 5 PMs have black balls, as the number of attacks grows. We can see a clear difference between the balanced and unbalanced cases. The chances of keeping a majority significantly improve for the latter case. For instance, after 5 attacks, there is still more than 40% chances of conserving the majority in the unbalanced case, while in the balanced case, this probability is below 10%. To plot the lines with removal we resorted to Monte Carlo simulation. As we expected, removal makes it easier for the attacker to reach a larger number of different PMs, thus negatively affecting the probability of keeping a majority of correct PMs. The four plots of the figure actually depict two extreme pairs of cases: the one without removal approximates a scenario where we have a very large number of VMs, or where the same VM can be handed to the attacker. The pair of lines with removal corresponds to the other extreme case, where we have a small number of VMs (10) for the available PMs (5).

Next, in Figure 3, we try a larger number of PMs, $m = 21$, to compare against the smaller set of 5. For this larger set, the advantage of unbalancing the distribution of VMs is even more visible in the case without removal, given by Equation 2. For the balanced option, after a little more than 20 attacks, there is nearly no chance that the majority of the PMs is still sane. This takes more than 60 attacks in the case of the unbalanced scenario.

We can now determine the number of attacks that are necessary until the attacker (probabilistically) holds the majority of the machines. We do this evaluation in Figure 4, for the balanced and unbalanced (60%) cases, without removal, for a varying number of PMs. In both cases, the growth seems to be approximately linear, but the slope is much higher in the unbalanced case, thus making it much more difficult to break.

Finally, in Figure 5 we evaluate the unbalance factor. We use 11 machines and make the balance change from $1/11$ (balanced) to $1 - 1/11 \approxeq 91\%$ (most unbalanced) in steps of $1/11$. For all these unbalanced factors, we plot the number of attacks until the attacker gets the majority of machines with probability greater than 50%. There is no removal.

6. PLACEMENT STRATEGIES OF THE DEFENSE

We now try to identify the best strategy to distribute the VMs by the PMs. Should the defender use a balanced approach or an unbalanced one? We consider six sorts of attacks, crash-stop, limited Byzantine and unlimited Byzantine, to VMs, or directly to the PMs (in fact, an attack to a PM could start from a VM), and two different goals, keeping a majority of PMs or VMs. Based on these twelve combinations, and on our evaluation of Sections 4 and 5, in Table 1 we recommend the best placement for the VMs.

Let us start from the simplest cases: crash-stop faults. In the first line, which concerns crashes that affect the entire PM (e.g., some software component that stops all the VMs or the hypervisor and other relevant services), the distribution of the VMs is irrelevant to the kind of majority the defender might want. This is evidently so in the case of a PM majority, whereas in Section 4.3 we demonstrate that the same holds for a VM majority. If the crashes occured at the VM level, and, if these crashes did not affect the PM or their services, then, again, the distribution of the VMs would be irrelevant. However, we think that it is more likely that the PM ceases to be useful for a service once the last VM residing there crashes. Therefore, in this case, we would like to avoid as much as possible the crash of all the VMs. In Section 4.1 we showed that balancing the VMs is the best option.

The cases with Byzantine faults are slightly more elaborate. In the "Limited Byzantine" attacks, in lines three and four of the table, the attacker randomly gets a VM or a PM when he/she hires a service. He or she cannot pick the VM/PM that better suits the attack. When the attacker is provided an entire PM[1], the distribution of VMs by the PMs is not relevant, regardless of the majority the defender wants to preserve. Since the defender loses the PM, VMs do not have any role in a majority of PMs, whereas Section 4.3, once again, applies to the case of a VM majority. Therefore, in line three, we have a pair of "Irrelevants". If, on the other hand, the fault starts on a VM and may extend to the entire machine (line four), the unbalanced option is the best to keep a PM majority (Sections 4.2 and 5). However,

[1]Another way of putting this case is to say that the attacker may reach all PMs with the same probability.

Table 1: Distribution of VMs by PMs as a function of the fault and the objective to achieve by the defender

Attack/Objective	PM Majority	VM Majority
Crash-Stop on PMs	Irrelevant	Irrelevant
Crash-Stop on VMs	Balanced	Irrelevant
Lim. Byzantine PMs	Irrelevant	Irrelevant
Lim. Byzantine VMs	Unbalanced	Balanced
Unlim. Byzantine PMs	Irrelevant	Balanced
Unlim. Byzantine VMs	Irrelevant	Balanced

unbalancing is negative to keep a majority of VMs, and, if this is the goal of the defender, he or she should balance the VMs by the PMs. This does not contradict the findings of Section 4.3, because in this case the most heavily loaded machine is *more likely* to suffer an attack, while in that Section the probabilities where the same.

The final cases are identical: if the attacker may choose the resource to attack, he/she may prefer one with many VMs (lines five and six). Unbalancing is therefore not relevant for a PM majority, but is negative for a VM majority.

7. CONCLUSION

Clouds are changing the surface of information technologies. However, the concentration of resources in the same region, network, or even machine pose an evident challenge to designers of dependable systems. In this paper we evaluate exactly this problem. We adopt the perspective of the cloud provider that needs to distribute processes or VMs by a given fixed set of PMs. While intuition could perhaps suggest that a balanced distribution of VMs would make a more dependable system, for many scenarios this is not the case: the distribution is either irrelevant or should be extremely unbalanced.

We think that the reasoning that follows our approach is to change the defender from the cloud provider to the user of the service. The user may anticipate himself to the cloud provider by distrusting it. In this case, the user might be tempted to hire the services of multiple providers to balance the VMs or the processes at his own will. In this case, despite the advantages of building a more robust system based on competing providers, the user would need to have a broker of cloud services, with the corresponding increase in complexity and costs.

8. REFERENCES

[1] Google app engine — google developers. `https://developers.google.com/appengine/`. Retrieved on August 5, 2012.

[2] Heroku | cloud application platform. `http://www.heroku.com`. Retrieved on August 5, 2012.

[3] Papers — oracle vm virtualbox. `https://www.virtualbox.org/wiki/Papers`. Retrieved on August 5, 2012.

[4] Ruby on rails and php cloud hosting paas | managed rails development | engine yard platform as a service. `http://www.engineyard.com`. Retrieved on August 5, 2012.

[5] Technical white papers — vmware. `http://www.vmware.com/resources/techresources/`. Retrieved on August 5, 2012.

[6] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 164–177, New York, NY, USA, 2003. ACM.

[7] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association.

[8] Stefan Berger, Ramón Cáceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn. vtpm: virtualizing the trusted platform module. In *Proceedings of the 15th conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, pages 305–320, Berkeley, CA, USA, 2006. USENIX Association.

[9] F.L. Camargos, G. Girard, and B. des Ligneris. Virtualization of linux servers. In *Proceedings of the Linux Symposium*, pages 63–76, July 2008.

[10] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, November 2002.

[11] Miguel Correia, Giuliana S. Veronese, and Lau Cheuk Lung. Asynchronous byzantine consensus with 2f+1 processes. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 475–480, New York, NY, USA, 2010. ACM.

[12] Y. Deswarte, L. Blain, and J.-C. Fabre. Intrusion tolerance in distributed computing systems. In *Research in Security and Privacy, 1991. Proceedings., 1991 IEEE Computer Society Symposium on*, pages 110 –121, may 1991.

[13] Norman L. Johnson and Samuel Kotz. *Urn Models and Their Application — An Approach to Modern Discrete Probability Theory*. John Wiley & Sons, Inc., 1977.