Using Virtualization Technology for Fault-Tolerant Replication in LAN

Fernando Dettoni[1], Lau Cheuk Lung[1], Aldelir Fernando Luiz[2]

[1] Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis, Brazil, {fdettoni, lau.lung}@inf.ufsc.br

[2] Departamento de Automação e Sistemas, Universidade Federal de Santa Catarina, Florianópolis, Brazil, aldelir@das.ufsc.br

Abstract. We present an architecture and an algorithm for Byzantine fault-tolerant state machine replication. Our algorithm explores the advantages of virtualization to reliably detect and tolerate faulty replicas, allowing the transformation of Byzantine faults into omission faults. Our approach reduces the total number of physical replicas from *3f+1* to *2f+1*. Our approach is based on the concept of twin virtual machines, where there are two virtual machines in each physical host, each one acting as a failure detector of its twin.

1 Introduction

More and more, computing systems are being used in critical systems and have to operate correctly even under the presence of faults. These faults can be accidental, like crash faults, or arbitrary, called Byzantine [1]. Thus, to ensure that these systems remain available under fault conditions, it is necessary to develop Byzantine fault-tolerant (BFT) mechanisms. One of the most used architecture is the State Machine Replication (SMR)[2], using deterministic state machines to offers a replicated service. Many BFT SMR-based approaches were developed (e.g. [3], [4], [5]). Among these, the PBFT [3] is often considered to be a baseline, being the first practical BFT algorithm and having most of later approaches derived from it.

Another technique consists of using unreliable failure detectors [6]. Despite of supporting at first just crash faults, some proposals were able to extend the idea to support Byzantine faults [7][8]. These fault detectors help the system giving some hints about replicas appearing to be faulty.

The virtualization can also be considered a Byzantine fault-tolerant technique, because it introduces an isolation level between the virtual machines. Several approaches use virtualization to protect some components from others' failure (or intrusion) [9], [10]. Virtualization techniques are widely accepted by industry, and

are largely used, e.g., by cloud computing services as Amazon Web Services and Windows Azure.

The PBFT and many other BFT SMR-based algorithms have a high implementation cost having normally the resiliency of $n \geq 3f + 1$, i.e., need $n > 3f$ replicas to tolerate f faulty replicas. To diminish this cost, some approaches emerged using a trusted component to limit the behavior of faulty replicas using only $n \geq 2f + 1$ replicas [11], [12], [13]. Other approaches were proposed running only $f + 1$ replicas, but keeping more $2f$ replicas waiting in a paused state without consuming any CPU time until it is activation [14].

We present a new efficient BFT SMR-based architecture based on virtualization, called TwinBFT. We reduced the number of required physical machines n from $n \geq 3f + 1$ to $n \geq 2f + 1$ to tolerate f faults. Furthermore, we reduced the number of communication steps in the normal case from 5 (in PBFT) to 3, without the client participation in the agreement. To our knowledge, this is the first algorithm with this number of steps without using a speculative approach [5], [13], involving the participation of the client in the agreement.

The proposed approach consists of use a set of twin virtual machines, executing the same application service in each one of the $N \geq 2f+1$ physical replicas. We use a set with only two virtual machines. Every virtual machine executes the same service, and each set acts as a replica of the state machine replication. The main idea is to use each virtual machine as a failure detector to its twin: upon sending a request to a pair of twin virtual machines, both must provide the same answer, otherwise the whole physical machine hosting the twins is considered faulty and the messages sent by this replica are ignored by the others. This way, each set of twin virtual machines will either act correctly or omit messages. This omission, however, are tolerated by the state machine replication. A crash fault too, is a kind of omission and, therefore, is also tolerated.

The proposed architecture do not intends to offer the ideal solution too all the cases. The use of virtual machines is suitable to companies opened to this kind of technology, as cloud computing companies. It is also recommended when you do not have trusted components or cannot wait for the activation of sleeping replicas in the case of failure. In these cases, an efficient BFT SMR with only $2f+1$ physical replicas ($4f + 2$ virtual machines) using virtualization may be suitable.

In Section 2, we give an overview of some related work showing the state of the art. Then, Section 3 describes our system model and assumptions. A detailed explanation of the algorithm is given in Section 4. Following, the Section 5 presents an evaluation of algorithm and the Section 6 summarize our conclusions.

2 Related Work

Several works were proposed recently in BFT to produce Byzantine fault-tolerant services based on SMR. Being the first practical approach for BFT protocols, PBFT is one of the most successful SMR protocol [3]. Although being practical,

the cost for implementing PBFT is quite high, requiring at least 4 replicas (*3f+1* to *f = 1*) and 5 communication steps. Thus, numerous approaches derive from it with two goals: reduce the number of required replicas and improve the performance.

Many works offered an alternative to improve the resiliency of PBFT reducing the number of replicas. Yin et al. introduces an architecture separating the service in two distinct layers, one responsible for agreement with *3f+1* replicas and another one executing the requests with only *2f + 1* replicas [4]. While still need *3f + 1* replicas, the execution replicas are likely to be much more expensive than agreement replicas.

Correia et al. presented the first solution to execute a BFT SMR with only *2f + 1* replicas, using a trusted distributed component [11]. After this, another work showed the first algorithm of it is kind based in a trusted local component using the abstraction of attested append-only memory [12]. Veronese et al. [13] proposed two algorithms using a trusted component, which supply unique identifiers for each message. The first one, MinBFT, reduced the number of necessary replicas to *2f + 1* and the number of communication steps to 4. The second, a speculative version called MinZyzzyva, reduced the communication steps even further, to 3, keeping the number of replicas in *2f + 1*.

In another approach, Stumm et al. [15] takes advantage of virtualization techniques to reduce the number of required replicas to *2f +1* since the VMM provides a secure communication between the replicas. This approach, however, require all replicas running on the same physical host and, therefore, do not tolerates crash faults on the physical machine, unlike this paper.

Another work, also based on the idea of two replicas watching each other, is presented in [16] using a signal-on-fail approach. This approach needs *4f+2* physical machines and requires a synchronous and trusted communication between each pair of replicas, which is a difficult assumption to be guaranteed in practice. In this same line of thought Inayat and Ezhilchevan presented an optimist multicast BFT protocol with total order based on the signal-on-fail approach. The authors showed that in a normal execution, it is likely to have better performance than other BFT approaches [17].

Several other works presented solutions to improve the performance of PBFT. Kotla et al. presented Zyzzyva, an algorithm able to reduce the number of communication steps in the absence of faults [5]. Instead of trying to reach an agreement before sending the reply to the client, the service replies speculatively. The service needs to execute the request again and reach an agreement only if the replies received by the client differ from each other. This approach takes advantage of most cases of the execution is free of failure but requires the ability to revert operations to ensure consistency in case of failures.

As stated in the Introduction, this paper explores another point of the design space using virtualization to deploy a BFT state machine replication with only *2f + 1* physical replicas (and *4f+2* virtual machines) and only 3 communication steps in the normal case of execution.

Several works use virtualization to isolate components of software. Two of the first use virtualization to protect the intrusion detector from the intruders [10], and

a more recent one uses the same idea to protect a honeypot monitor [9]. Nevertheless, the hypervisor security is mandatory to obtain isolation and some works studied how to improve this security. Murray et al. proposed dissociates the virtualization system as a solution to lower the trusted computing base of the system [18]. The NoHype goes further removing the hypervisor of the way and executing the virtual machines natively [19]. The HyperSafe uses another approach: protects the hypervisor detecting attacks that modify the control flux, as buffer overflows [20].

3 System Model

The architecture of the system is presented in Figure 1. The system is composed by a set of n physical machines (e.g. host) $H = \{h_1, h_2, \ldots, h_n\}$ where $n \geq 2f + 1$ and f is the maximum number of faulty physical machines at any time. Each host of the Figure 1 contains a VMM (*Virtual Machine Manager*) with two virtual machines, called twins virtual machines, running one process each. Both servers $\{s_i, s_i'\}$ execute the same service (with different versions because of software diversity), and communicate between each other to validate each message before send to other processes.

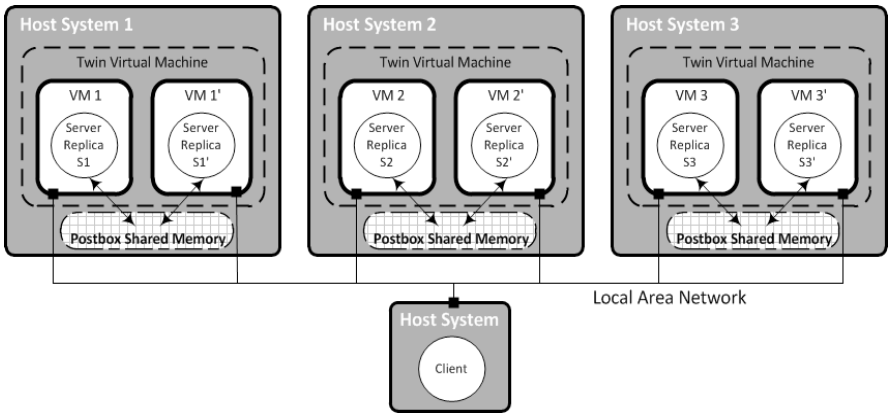

Fig. 1. TwinBFT - Twin virtual machines architecture.

We assume at most f virtual machines acting Byzantine, but no more than 1 is faulty in the same physical host. When a virtual machine is arbitrarily faulty, the validation mechanism transforms this fault to an omission fault. Therefore, we assume up to f physical hosts can be faulty by crash or omission, accidentally or due to a failure in one of it is virtual machines. To substantiate the f faults limit we have to relay in software diversity techniques, i.e., different implementations of the process at each replica in a physical host [21], [22]. This diversity reduces the

chance of more than one virtual machine at the same physical host being attacked simultaneously. We assume the virtualization provides isolation between the virtual machines and the VMM / Hypervisor.

No assumptions are made about the time needed to compute a request. The communication between different *VMs* inside the same host is made through a shared memory space, called postbox. The processes at different hosts communicate through the local network, by message passing only. This local network can fail to deliver messages, delay, or duplicate messages.

Each host can assume two different roles: (1) primary host, which is responsible for defining the order for executing the client requests; and (2) backup host, which executes the requests following the order proposed by the primary. Within a primary host, a process can assume two possible roles: (1) leader, which is responsible for assign the sequence number for client's requests; and (2) follower, which executes the requests following the order defined. All the processes within backup hosts are considered followers. The primary host hi are defined by $i = v \bmod |S|$ where v is the current view. The primary leader process within a server is, by definition, s_i.

We use cryptographic techniques to authenticate messages and ensure authenticity of messages [24]. Each pair of processes share among each other a secret key used to generate a MAC (*Message Authentication Code*) vector [3] with a valid signature for each process.

We assume each physical machine with only two virtual machines (*VMs*) where, for a given input, the replies supplied by both should be the same to the physical machine to not be considered faulty.

4 Algorithm

The service in our algorithm is modeled as a state machine replication, distributed across the service nodes. The replicas move through a succession of configurations called views. In each view, we have one primary replica, which is responsible for defining the message order, and forward the request to all service replicas. As stated by [2], the state machine must be deterministic, and all replicas must start in the same state, otherwise the safety cannot be guaranteed.

In this section we will discuss our proposal of transformation through a well-known approach. In this sense, we present an adaptation of PBFT protocol [23] to our proposed model. In each view, only one replica s_j is the primary (or leader), which is responsible for defining the message order and forward the request to all service replicas. If a message sent by any replica is signed by both VMs in a host we assume this message as correct, since we assume that only one VM can fail at the same time in the same host.

4.1 Properties

As a state machine replication, our algorithm must ensure the following properties to provide a correct service:

- **Total Order** (*safety*): a request is executed sequentially and at the same order on every replica, i.e. despite replication, the operations are atomically executed in order and the system behaves as a centralized one;
- **Termination** (*liveness*): a request issued by a client eventually complete, regardless of failure.

Our algorithm provides both safety and liveness, assuming that no more than $f = (n - 1)/2$ hosts are faulty and there is at least one correct process s in each host. To ensure that all replicas will execute the requests in the same order, all the replicas follow the order defined by the leader and the leader can be assumed correct if both processes at the primary host sign the order proposed by the leader. Our protocol ensures safety regardless of timing but to ensures liveness we need to make some assumptions about synchrony and message loss.

To ensure that all replicas execute the same requests in the same order, all replicas follow the order defined by primary leader. A consensus algorithm is not necessary because we can trust the order defined by the primary leader, provided that our previous assumptions are not violated. This is because after the primary leader defines the order, this order will not be accepted by the other replicas without the agreement of both processes in primary host.

4.2 Description of Algorithms

In this section, we will discuss our algorithms in detail. First, we show in Figure 2 a diagram-based view of our main algorithm, to make easy to understand it. This architecture assumes $f = 1$, where three hosts are required, each one with two VMs. Each pair of VMs inside the same host communicates between each other through a trusted FIFO channel called postbox. The postbox can be faster than the network by using a shared memory abstraction provided by the VMM.

The algorithm works basically as follows:

1. Client issues a request to both VMs in primary host;
2. The primary leader s_i define a sequence number and post an “ORDER” message on postbox;
3. The primary follower s_i' reads the message from postbox, gets the sequence number and posts on postbox an “ORDER” message with the sequence number received;

4. Both VMs sign the “ORDER” read from twin and send to backup replicas;
5. As soon as each VM inside a backup replica receive the message “ORDER”, they execute the operation, and post a signed “REPLY” on the postbox;
6. When a VM reads a “REPLY”, it compares with the one computed locally and if all fields matches, attaches its own signature to the message and send to client;
7. If the client receives at least *f + 1* correctly signed (by both twin VMs) replies from distinct physical replicas, it accepts the result.

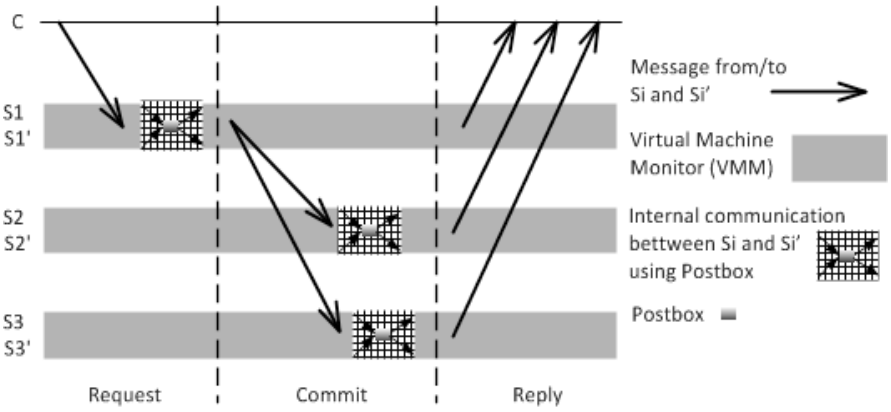

Fig. 2. Algorithm’s steps sequence in normal case operation with f = 1.

In a normal execution, a client sends a request to the service and waits until receives at least *f + 1* valid replies from distinct replicas. The request message has the form <REQUEST, *c*, *seq*, *opi*>$_{\sigma c}$ where *c* is the client id, *seq* is a request id on a client, and *op* is the operation to be executed on the service. If the client does not receive *f + 1* messages soon enough, it multicast the request to all replicas.

4.3 Normal Case Operation

The algorithm, at normal case operation, running in each one of the replicas has two concurrent tasks. The Task 1 is responsible for reading the messages received through network. Task 2 is responsible for reading the messages from postbox, posted by its twin. The state of each process is composed by the state of the service, a message buffer and the current view. This state is shared among the tasks.

When any of processes {s_i, s_i'} in primary host receives a request from client, s_i generates a new sequence number *n* and create a message <<ORDER, *si*, *v*, *n*, *dm*>$_{\sigma pi}$>, m> where *v* is the current view number, and *dm* is the digest of message

m. As soon as s_i' reads the *si* "ORDER" message from the postbox and have the "REQUEST" message in the message buffer, it gets the sequence number proposed by s_i, creates an "ORDER" message and posts on the postbox. When each one reads an "ORDER" message from postbox, it verifies if all parameters correspond to the ones computed locally and, if yes, add it is own signature to the twin message and multicast to backup replicas.

To any "ORDER" message received, the replicas will consider it valid if:

- Message is correctly signed, i.e., if received from network both twin *VM* machines on the replica must sign it, and if received from postbox, its own twin machine must sign it;
- The view in the message is the current view;
- Has not accepted another "ORDER" message with the same sequence number for a different request.
- The sequence number is between a low and high water marks *h* and *H* (in practice, if this verification is made when the primary follower reads the "ORDER" message from postbox, a backup replica will never receive a message outside these water marks).

Upon the receiving of "ORDER" message by both twin processes on a physical host, each one of $\{s_i, s_i'\}$ verifies if the message is valid and, if yes, it executes the operation and create a message $<\text{REPLY}, si, v, seq, c, res>_{\sigma pi}$ where *res* is the result of executing the operation, and posts on the postbox. Once the "REPLY" has been read from the postbox, it compares each parameter of the message and, if all parameters are identical to the ones locally computed then sign the message generated by its twin and send to client.

When the client receives a "REPLY" it accepts as a valid message if the following conditions are true:

- Is signed by both processes $\{s_i, s_i'\}$ on sender host;
- It has not accepted yet a valid message from any of the twin processes on the sender host.

The client waits until have received at least $f + 1$ valid messages from the replicas to accept the result. If it could not receive these messages, soon enough, it multicast the "REQUEST" to all replicas.

5 Analytic Evaluation

In Table 1 we can see a comparison between our approach and state-of-the-art BFT algorithms in the literature. All numbers considers only gracious executions. The benefits of using a twin machines approach are visible on the number of replicas and communication steps. While our approach has the lowest number of replicas, along with [11], [12], [13], it has the same number of communication steps of the speculative algorithms [5], [13] even in case of faults, this is an interesting achievement. Speculative algorithms, however, require more communication steps

in case of faults, additionally involving the client in the protocol, leaving behind the transparency of the protocol.

	Number of Replicas	Number of Processes	Number of Physical Machines	Communication Steps (latency)	Speculative / Optimist
PBFT[3]	3f+1	3f+1	3f+1	5	no
Zyzzyva[5]	3f+1	3f+1	3f+1	3 / 5	yes
TTCB[11]	2f+1	2f+1	2f+1	5	no
A2M-PBFT-EA[12]	2f+1	2f+1	2f+1	5	no
MinBFT[13]	2f+1	2f+1	2f+1	4	yes
TwinBFT	2f+1	4f+2	2f+1	3	no

Table 1. Comparison of Evaluated BFT Algorithms

As our approach uses two virtual machines at each replica, we have a bigger number of processes despite of the number of required physical machines being the same as [11], [12], [13].

6 Conclusions

By exploring some virtualization techniques, we proposed a less expensive alternative algorithm to BFT. We show that is possible to implement a reliable *2f + 1* SMR algorithm in an asynchronous environment. Despite of relying in a secure communication between each pair of virtual machines, we believe that virtualization is widely available today and can provide a good isolation between the replicas and the external world. Moreover, we reduced the number of necessary communication steps, reducing the cost of communication.

References

[1] Lamport L, Shosta, R, Pease, M (1982) The Byzantine generals problem. ACM Trans. Program. Lang. Syst. 4(3):382-401.

[2] Schneider F B (1990) Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput. Surv. 22(4):299-319.

[3] Castro M, Liskov B (1999) Practical Byzantine fault tolerance. In: Proc. of the 3rd OSDI, Berkeley, CA, USA, USENIX Association. 173-186.

[4]. Yin J, Martin J P, Venkataramani A et al (2003) Separating agreement from execution for Byzantine fault tolerant services. SIGOPS Oper. Syst. Rev. 37 253-267.

[5] Kotla R, Clement A, Wong E et al (2008) Zyzzyva: speculative Byzantine fault tolerance. Commun. ACM 51 86-95.
[6] Chandra T D, Toueg S (1996) Unreliable failure detectors for reliable distributed systems. J. ACM 43(2) 225-267.
[7] Doudou A, Garbinato B, Guerraoui R et al (1999) Muteness failure detectors: Specification and implementation. In: Proc. of the 3rd EDCC, Springer-Verlag 71-87.
[8] Kihlstrom K P, Moser L E, Melliar-Smith P M (2003) Byzantine fault detectors for solving consensus. The Computer Journal 46.
[9] Jiang X, Wang X (2007) Out-of-the-box monitoring of VM-based high-interaction honeypots. In: Proc. of the 10th International Symp. on Recent Advances in Intrusion Detection.
[10] Garfinkel T, Rosenblum M (2003) A virtual machine introspection based architecture for intrusion detection. In: Proc. of the Network and Distributed Systems Security Symposium.
[11] Correia M, Neves N F, Verissimo P (2004) How to tolerate half less one Byzantine nodes in practical distributed systems. In: Proc. of the 23rd IEEE SRDS. 174-183.
[12] Chun B G, Maniatis P, Shenker S et al (2007) Attested append-only memory: making adversaries stick to their word. In: Proc. of the 21st ACM SOSP. 189-204.
[13] Veronese G S, Correia M, Bessani A N et al (2013) Eficient Byzantine fault tolerance. IEEE Transactions on Computers 62(1):16-30.
[14] Wood T, Singh R, Venkataramani A et al (2011) ZZ and the art of practical BFT execution. In: Proceedings of the 6th ACM SIGOPS/EuroSys European Systems Conference. 123-138.
[15] Stumm V, Lung L C, Correia M et al (2010) Intrusion tolerant services through virtualization: A shared memory approach. In: Proc. of the 24th IEEE AINA. 768-774.
[16] Mpoeleng D, Ezhilchelvan P, Speirs N (2003) From crash tolerance to authenticated Byzantine tolerance: A structured approach, the cost and benefits. In: Proceedings of the IEEE/IFIP 33rd International Conference on Dependable Systems and Networks. 227-236.
[17] Inayat Q, Ezhilchelvan P (2006) A performance study on the signal-on-fail approach to imposing total order in the streets of byzantium. In: In Proc. IEEE DSN. 578-587.
[18] Murray D G, Milos G, Hand S (2008) Improving Xen security through disaggregation. In: Proceedings of the 4th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. 151-160.
[19] Szefer J, Keller E, Lee R B et al (2011) Eliminating the hypervisor attack surface for a more secure cloud. In: Proceedings of the 18th ACM Conference on Computer and Communications Security. 401-412.
[20] Wang Z, Jiang X (2010) HyperSafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In: Proc. of the IEEE Security and Privacy Symposium. 380-395.
[21] Bessani A, Daidone A, Gashi I et al (2009) Enhancing fault / intrusion tolerance through design and configuration diversity. In: Proceedings of the 3rd Workshop on Recent Advances on Intrusion-Tolerant Systems.
[22] Gashi I, Popov P T, Strigini L (2007) Fault tolerance via diversity for o#-the-shelf products: A study with SQL database servers. IEEE Transactions on Dependable and Secure Computing 4(4):280-294.
[23] Castro M, Liskov B (1999) Authenticated Byzantine fault tolerance without public-key cryptography. Technical report, Cambridge, MA, USA.
[24] M. S. Wangham, L. C. Lung, C. M. Westphall,and J. da Silva Fraga, “Integrating SSL to the JACOWEB security framework: Project and Implementation,” in IM’01, 2001, pp. 779–792.