Independent Faults in the Cloud

Rachid Guerraoui and Maysam Yabandeh
School of Computer and Communication Sciences, EPFL, Switzerland
email: `firstname.lastname@epfl.ch`

ABSTRACT

Byzantine fault tolerant (BFT) protocols are replication-based solutions to the problem of tolerating the arbitrary failures of software and hardware components. The essential assumption for replication is *independence* of failures. In this paper, we categorize four different failure independence levels that could be obtained from the cloud. Providing more level of independence comes with the cost of more delays and less bandwidth, and not a single BFT protocol fits all these deployment setups. Using experimental results, we discuss the possible appropriate BFT protocol for each category.

1. INTRODUCTION

A software system is logically created of multiple layers, where each layer implements a particular functionality required by the software. Choosing the right layer for implementing a feature is a design decision, which could affect both correctness and performance of the software. End-to-end argument principle [9] discusses that when a particular functionality is implemented at a lower layer used by multiple upper-layer modules, some of them might not ever use the functionality and yet have to pay for it. Furthermore, the feature, which is now implemented in a general way, might not also correctly satisfy the requirements of the modules which are actually using it. The design of applications in the cloud environment could be re-thought by the help of end-to-end argument.

The cloud is a promising, growing business model that can help reduce the IT costs; the cloud provider takes the administrative load of the IT infrastructures off the cloud clients. The availability and the safety of the service provided by a third party, such as the cloud, can be improved by replicating the service and launching Byzantine fault tolerant (BFT) protocols [8, 3] on top of replicas. [1] The software will then be using BFT as a lower layer module. Alternatively, the safety can be directly addressed by the upper-layer modules; for example, Google file system [5] leaves the unexpected duplicated records in the files to be dealt by each application. Once again, we have a design decision on which the end-to-end argument could apply: when should we use a separate BFT layer to tolerate faults? Of course, giving a definite answer to this question in general is very hard and depends on various parameters. A recent study [11] lists some practical faults that could not be avoided by a naive BFT layer. In this paper, we distinguish the possible faults in the cloud environment. Focused on the ones that BFT layer can correctly tolerate, we discuss that not one single BFT protocol can efficiently cover all the faults.

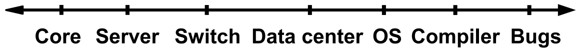

Figure 1: The spectrum of failures.

The main assumption behind the replication-based techniques is the independence of failures. Although the genuine independence is not achievable in reality, depending on the kind of fault, the level of independence can be improved by using some specific replication settings. The spectrum of faults that can risk the availability and safety of a software system (Figure 1) can range from failure of a core inside the processing unit, to programming errors. The more we move towards the programming errors direction, the harder it is to provide failure independence, which is required by BFT protocols; to have real failure independence for programming errors we need multiple independent developing teams, which is very expensive, whereas a simple replication over some servers inside a local area network (LAN) is enough to provide core failure independence.

The failures in the cloud that a BFT protocol could help tolerate can be categorized into the following groups: i) hardware failures, which include faulty core, power failure, storage failure, and failure of connecting switches and routers; and ii) malicious attacks, which include security attacks that compromise the virtual machine (VM) or the entire server [2], as well as Denial of Service (DoS) attacks, which can make a server, switch, or router to saturate and become unavailable. To tolerate hardware failures, the service should be replicated on independent hardware components. The cloud provider can increase the independence of the involved hardware components by assigning the customer nodes to different i) machines, ii) LANs, and iii) geographically distributed data centers.

The attacker that compromises some replicas inside a cloud could

LADIS 2010 Zurich, Switzerland

---

[1] Note that launching BFT is a client-side decision and the cloud vendor does not have to be involved in it.

[2] A physical machine in the cloud is shared between multiple clients via virtualization.

use them as agents of a distributed DoS attack. Therefore, replicas located on the same cloud have a similar risk of failure by a DoS attack. Besides, using multiple clouds could increase the independency of software vulnerabilities if the cloud vendors use separate software components, such as for the host operating system and the virtualization technology. Nevertheless, after a replica is compromised, the replication factor as well as the address of the other replicas can be obtained by the attacker who then can focus its attacks on them, either by trying to compromise them or by directing the DoS attacks towards them. To achieve more independence of replica failures, the address of the other replicas must be unknown to a (potentially compromised) replica. We refer to this technique as *oblivious* clouds.

After a brief overview of BFT protocols in Section 2, we categorize the replications setups into four groups: i) multiple machines inside a LAN, ii) separate LANs (Multiple availability zones), iii) separate cloud providers, and iv) oblivious clouds; the BFT protocols in each category are correct assuming existence only a subset of faults. Different system characteristics of replication setups, such as latency and bandwidth, demand different BFT protocols which can perform efficiently in them. Using experimental results in *bftsim* [10], we propose the appropriate BFT protocol for each category.

2. BACKGROUND

BFT protocols are replication-based solutions to the problem of tolerating arbitrary failures of software and hardware components. The protocol can ensure safety and progress of up to f of a particular faulty component, if more than $3f$ replicas of that component are used in replication. For example, if the application is replicated on four separate machines, then the BFT protocol can tolerate at most one faulty hard disk [2]. In this section, we give a brief overview of BFT protocols as well as different BFT setups. The BFT protocols differ in the number of required phases to commit (communication rounds), response latency, and throughput. In general, there is a trade-off between latency and throughput; to have high throughput, the contention between two competing client requests must be avoided by using a primary. The primary orders the client requests and then forwards the ordered requests to the other replicas. Although this offers high throughput, the commit latency increases because of the extra phase of communicating through the primary.

To tolerate malicious attacks, the messages must be authenticated via some cryptographic techniques, such as Public Key Cryptography (PKC), which authenticates a single message, and Message Authentication Code (MAC), which authenticates a single channel (and its messages). PKC could make the BFT protocols much simpler since it is verifiable even after the message is forwarded multiple times, but it is around 100 times slower than MAC. The throughput can be bounded by the number of MAC operations per request performed by the bottleneck replica. This is mostly the case in 0/0 setup, where q/p stands for requests and replies with payload size of q and p KB, respectively.

For large message sizes (4/4 benchmark), the throughput is bounded by the output bandwidth of the bottleneck replica, i.e., the replica that sends/receives more messages per request. An example of such a bottleneck is multicasting the request by the primary. The multicast cost can be remedied in setups such as LAN that support hardware multicast. Nevertheless the cloud vendors might be reluctant in offering the hardware multicast support due to its scalability issues [12].

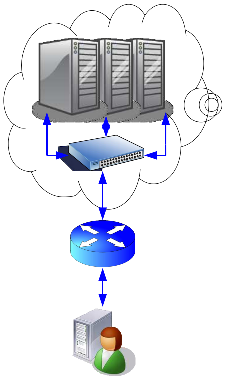

Figure 2: The LAN setup.

In PBFT protocol [3], the client sends the request to all the replicas including the primary to get ordered. The primary forwards the order to other replicas. To detect the faulty primary, all the replicas broadcast the received order to make sure all the replicas have received the same order for the request. Then all the replicas broadcast the ordered request and execute it. This phase is necessary to detect the interference of two primaries, which might happen during view change. The client accepts the replies if they match.

In Zyzzyva [7], the client sends the request only to the primary to get ordered. However, after other replicas receive the requests from the primary, they immediately execute it and send the reply to the client. The client accepts the replies if they all match. Otherwise, either the primary or some of the replicas are faulty. In this case, the first correct client can detect that and demand changing the primary.

Chain [6] also uses a primary to avoid contention. All other replicas are ordered as a chain and each one forwards the request to the next. The last replica sends the reply to the client. This technique increases the end-to-end delay, but the throughput improves as the number of MAC operations by each replica is close to 1, i.e. the theoretical lower bound. The key idea is to partition the replicas into two groups, where one group only verifies the client requests and the other only authenticates the reply. After detecting the failure, the whole protocol aborts and the abort history is used to initialize another instance of BFT protocol. This technique is called *Abortable BFT* [6].

Protocols that use a primary can take advantage of a technique named *batching*; the primary batches N requests (mostly N=10 [3, 7]) and performs the MAC operation only once on all of them. The increased latency is negligible in high-throughput systems. Q/U [1] does not use a primary and the clients directly communicate with the replicas. If two clients are accessing the replicas at the same time, the protocol requires more number of phases to resolve the contention. Although it has the advantage of optimal communication rounds in non-contention cases, it requires more number of replicas to operate, i.e., $5f$+1. Moreover, increasing the number of clients could increase the contention. Quorum [6], an abortable version of Q/U, also suffers from the contention problem.

3. MULTIPLE MACHINES

A LAN setup, in which the replicas are connected via a switch, offers the best performance for replication because of the low la-

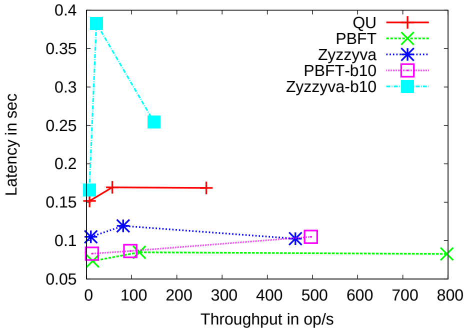

Figure 3: 0/0 benchmark, same LAN.

tency (1 ms) and high bandwidth (1 Gbps) between the replicas. The clients of the cloud are always connected through a wide area network (WAN), which implies 10-30 ms latency and higher packet loss rate. The setup is depicted in Figure 2.

On the other hand, the LAN setup offers the lowest failure independence among the possible setups. Upon the failure of a switch connecting the replicas, the protocol cannot progress any more. Furthermore, a LAN can spread only over a limited local area. It is likely, therefore, that some failures, such as main power failure, fire, and earthquake, affect all the replicas. In the particular case that the VMs are placed inside the same physical machine or rack, the failures are even more dependent since the failure of the physical machine or the rack power source, respectively, will affect all the replicas.

In theory, the BFT protocols are usually compared based on the required number of phases to commit a value. In practice, however, the low latency of the LAN makes the difference between the overall latency of the BFT protocols negligible. Instead, leveraging the high available bandwidth of LAN becomes more appealing. For small payloads (0/0 benchmark) where the bottleneck is the CPU, Chain [6] is the best candidate since it performs only $1 + \epsilon$ MAC operations; the ϵ parameters gets closer to zero, by using higher degree of batching. Figure 3 compares the throughput of the state-of-the-art BFT protocols (excluding chain), using *bftsim*. The batching factor is always 10 and is specified by a "b10" suffix. The timeout values in bftsim are set to 0.204 s, 0.210 s, and 0.206 s for Q/U, PBFT, and Zyzzyva, respectively. The client-replica and replica-replica latency is set to 60 ms and 0.08 ms, respectively. In all the experiments the loss rate of client-replica communications is set to 0.05%, client-replica bandwidth is 1 Mbps, and the number of clients is 1, 10, and 50. As we expected, the low latency of LAN, makes PBFT have the best performance. Since we did not have any implementation of Chain in *bftsim*, we use the reported numbers at [6], to compare Chain with the other protocols. Figure 4 shows that Chain outperforms all the other protocols in high throughputs. Because the Chain uses fewer client-replica messages compared to PBFT and Zyzzyva, we expect that the same conclusion applies to the setup where the clients are behind a WAN.

Figure 5 presents the experimental results of 4/4 benchmark [3]. The

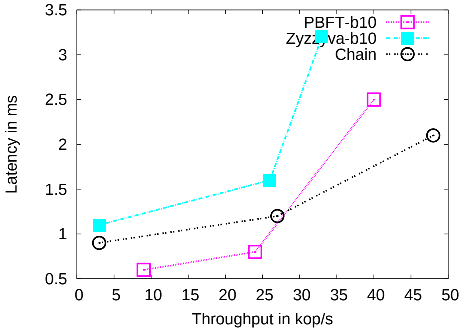

Figure 4: 0/0 benchmark [6].

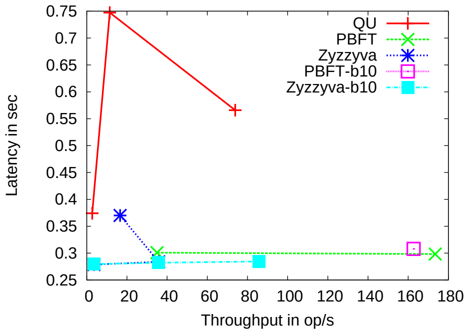

Figure 5: 4/4 benchmark, same LAN.

change in the message size does change the performance, but PBFT is still leading. Zyzzyva performs worse than PBFT, since it is more sensitive to the replica-client message delay and loss; not receiving all the replies before timeout, makes Zyzzyva to fall into the two phase recovery operation. We could not justify the appeared knee in the performance of Q/U by increasing the number of clients. The same pattern was observed in Figure 8 of bftsim simulation results [10], but that also was left unexplained.

We also ran experiments having multicast enabled for both 0/0 and 4/4 benchmarks. Surprisingly, the throughput of the protocols dropped after enabling multicast. Since the multicast feature in bftsim is simulated by computing less delay for multicasting the messages, we could not interpret the drop in the throughput.

4. MULTIPLE AVAILABILITY ZONES

Availability zones are distinct locations inside the same geographical region that are engineered to have independent failures via separate power-lines, being isolated in separate rooms not to allow a fire to propagate to them, and probably using separate gateway routers[4]. The setup is depicted in Figure 6. Because they are located in the same region, the latency between them is still low (1-5 ms) and the available bandwidth is still high (100-1000 Mbps). The higher latency makes the BFT protocols with less number of required phases more appealing. Multicast feature might be available (although less likely) depending on the engineering of the zones.

Figures 7 and 8 compare the performance of the protocols in 0/0 and 4/4 benchmarks, respectively. The timeout values in bftsim are

---

[3] The missing numbers in the figures imply that the corresponding experiments never finished. This is either a liveness bug in the protocol or the bftsim implementation. In the case of Zyzzyva, we observed the similar behavior in the experiments with the C++ implementation released by its authors.

[4] Since Amazon does not offer this information, we do not know for sure how isolated the availability zones in EC2 [4] are.

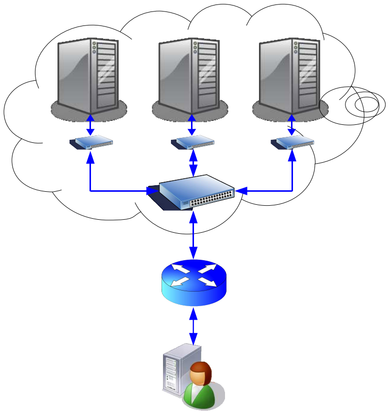

Figure 6: The setup for multiple availability zones.

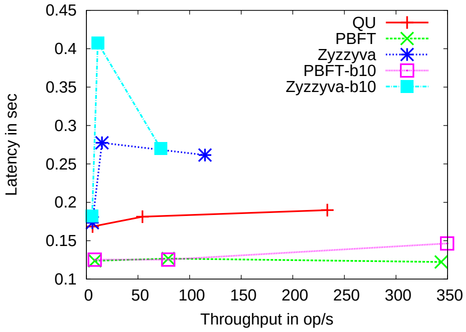

Figure 7: 0/0 benchmark, multiple zones.

set to 0.212 s, 0.242 s, and 0.222 s for Q/U, PBFT, and Zyzzyva, respectively. The client-replica and replica-replica latency is set to 60 ms and 10 ms, respectively. The replica-replica bandwidth is set to 1 Gbps. Both the throughput and the latency of Zyzzyva-b10 are improved by increasing the number of clients from 10 to 50, perhaps because the increase in number of requests makes the batching technique more effective. Again, the sensitivity of Zyzzyva to the WAN delays makes the PBFT to be still the best option. However, for low number of clients in 4/4 benchmark, Zyzzyva outperforms PBFT. It can be explained by the fact that in PBFT the clients have to send the requests to all the replicas, while in Zyzzyva it is sent to only the primary. For large messages, this overhead can have a negative impact on PBFT's performance. For larger number of clients, Zyzzyva gets over-saturated and we observe a decrease in throughput as well as an increase in latency. The over-saturation had been observed in other experiments with bftsim [10].

5. MULTIPLE CLOUD PROVIDERS

To provide failure independence against regional power failures and disasters, such as earthquake and tsunami, the replicas must be replicated over separate geographical regions. Geographical distribution of replicas also provides availability against tier-1 router failures. Amazon already offers computing units distributed over three separate regions. However, using separate cloud provider also offers failure independence against vulnerabilities of a particular cloud provider. For example, if the replica maintained by a cloud provider is compromised because of vulnerability in the used VM technology, the other replicas can still progress since the corresponding cloud providers probably use a different technology for vitalization.

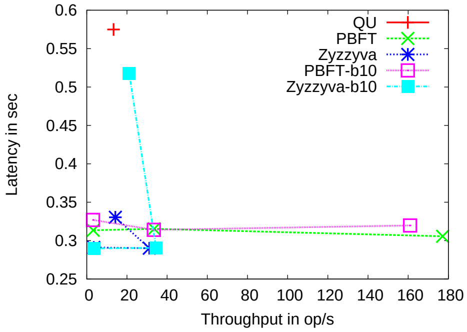

Figure 8: 4/4 benchmark, multiple zones.

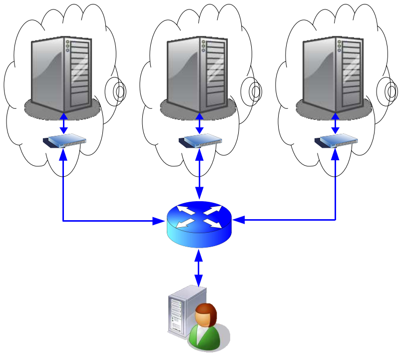

Figure 9: The setup for multiple clouds.

The setup is depicted in Figure 9. The WAN that connects the clouds has high latency (10-30 ms) and low bandwidth (1-10 Mbps). Moreover, the multicast feature is not available, which has a negative impact on the performance of protocols such as PBFT and Zyzzyva. Not existence of multicast increases the overhead on the primary, which is in charge of sending the proposals to all replicas.

Figures 10 and 11 compare the performance of the protocols in 0/0 and 4/4 benchmarks, respectively. The timeout values in bftsim are set to 0.262 s, 0.442 s, and 0.322 s for Q/U, PBFT, and Zyzzyva, respectively. The client-replica and replica-replica latencies are set to 60 ms. The replica-replica bandwidth is set to 1 Mbps. The loss rate of replica-replica communications is set to 0.05%. The high latency between the replicas, make the performance of PBFT and Zyzzyva to drop very quickly. In contrary, the Q/U's performance surprisingly scales very well with the number of clients. This is in contradiction with the intuition that more clients increases the contention in the Q/U and hence lowers the throughput. We can explain this as following: because of the high WAN delays, the throughput is in general low anyway, and hence the interference between the client's requests occurs rarely in Q/U.

6. OBLIVIOUS CLOUDS

Although the distribution of the service over multiple could providers offers a high availability against non-malicious attacks, the failure of a compromised replica will still increase the odds of failure of other replicas. In other words, the failures of replicas are not completely independent. The reason is that after a replica is compromised, the location of the other replicas can be obtained from the

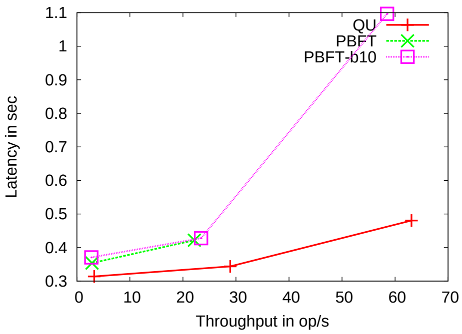

Figure 10: 0/0 benchmark, multiple clouds.

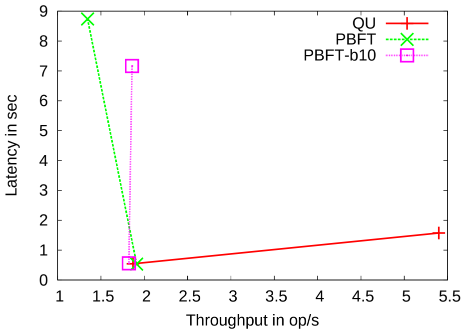

Figure 11: 4/4 benchmark, multiple clouds.

compromised replica [5]. The attacker can then focuses its attacks on other replicas. For example, it is very difficult to avoid a distributed DoS attack when the attacker has enough motivation to do so and also knows the location of the target victim.

To keep the address of the replicas anonymous, there should not exist any replica-replica communication. Therefore, all the communications must be performed through the clients. The setup is depicted in Figure 12. In this scheme, the cloud providers (and consequently the replicas) do not even know that we are running a replication protocol. The BFT protocols that fall into this category are Q/U [1] and Quorum [6]. Quorum is basically an abortable version of Q/U, which also uses less number of replicas ($3f+1 < 5f+1$). It is worth noting that Q/U specification allows replica-replica communication for object synchronization. Hence, to be able to use Q/U in the oblivious clouds setup, the corresponding optimization in Q/U must be disabled.

7. CONCLUSION

In this paper, we have categorized the different levels of failure independence that can be offered to the BFT protocols by different setups in the cloud. Providing higher failure independence comes with the performance penalty due to the increased latency and the decreased bandwidth. Because the clients are connected via a WAN, the protocols that are sensitive to the replica-client delay and loss do not perform well. Chain achieves the best throughput when the replicas are connected via a LAN. For availability zones, where the latency between the replicas is higher, PBFT offers the best performance. Q/U performed the best for the setup that the replicas are geographically distributed, because of using no communication between the replicas. To achieve the highest failure independence in the oblivious clouds setting, the only available options are Q/U and its abortable version, Quorum.

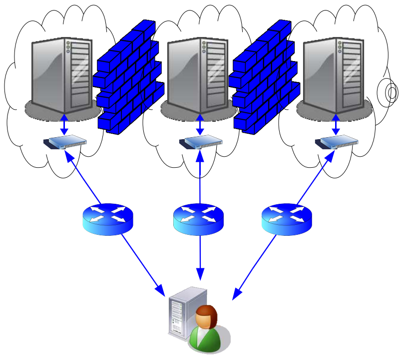

Figure 12: Oblivious clouds setup.

8. REFERENCES

[1] M. Abd-El-Malek, G.R. Ganger, G.R. Goodson, M.K. Reiter, and J.J. Wylie. Fault-scalable Byzantine fault-tolerant services. *ACM SIGOPS Operating Systems Review*, 39(5):59–74, 2005.

[2] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)*, 32(4):824–840, 1985.

[3] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[4] Amazon ec2. http://aws.amazon.com/ec2/.

[5] S. Ghemawat, H. Gobioff, and S.T. Leung. The Google file system. *ACM SIGOPS Operating Systems Review*, 37(5):43, 2003.

[6] R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić. The next 700 BFT protocols. In *Proceedings of the 5th European conference on Computer systems*, pages 363–376. ACM, 2010.

[7] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: speculative byzantine fault tolerance. *ACM SIGOPS Operating Systems Review*, 41(6):45–58, 2007.

[8] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Com. of the ACM*, 21(7):558–565, 1978.

[9] J.H. Saltzer, D.P. Reed, and D.D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)*, 2(4):288, 1984.

[10] A. Singh, T. Das, P. Maniatis, P. Druschel, and T. Roscoe. BFT protocols under fire. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 189–204. USENIX Association, 2008.

[11] Y.J. Song, F. Junqueira, and B. Reed. BFT for the skeptics (WIP). In *SOSP*, 2009.

[12] Y. Vigfusson, H. Abu-Libdeh, M. Balakrishnan, K. Birman, R. Burgess, G. Chockler, H. Li, and Y. Tock. Dr. multicast: Rx for data center communication scalability. In *Proceedings of the 5th European conference on Computer systems*, pages 349–362. ACM, 2010.

---

[5] An attacker that is based inside the cloud can also obtain this information by monitoring the network traffic of a replica.