Security, privacy, and fault-tolerance in the Cloud, by xxxxxx

Introduction

As Amazon, Microsoft, Google, Rackspace, and other cloud providers race to offer the fastest, most reliable, most scalable, most secure, most private, and least expensive cloud services they will inevitably need to make tradeoffs amongst these goals. Scaling reliably, that is scaling in a fault-tolerant manner, increases equipment and operational costs and tends to lead to lower performance [1]. Extra security and privacy require additional engineering and complicate troubleshooting, both of which increase costs. Encryption also imposes processing overhead for clients and services, reducing the useful work that can be performed [2]. Striving to be the fastest and most reliable, from a network standpoint, drives infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS) to be globally distributed, which leads to higher equipment and operational costs and subjects the services to discovery demands of government agencies in many jurisdictions [3], which very likely has a negative effect on privacy. Additionally, from a statistical standpoint, any effort that increases the number of nodes in the service also increases the likelihood of a node failing.

To complicate matters further, the increased private and public sector use of these cloud services makes the services prime targets for attack and exploitation by attackers, and it increases the potential damage to the service provider, customer, and end user for a successful attack [4] [3]. When combined with lawful, and possibly unlawful, demands of government agencies, the likelihood of insider or malicious attacks means that service providers are not just dealing with arbitrary faults but also the possibility of faults caused by malicious intent. Faults that can't have their cause narrowed between arbitrary failures and malicious attacks are referred to as Byzantine failures [1] [4] [5], named after the "*Byzantine generals problem*" [5]. The majority of research on distributed fault-tolerance is focused on Byzantine fault-tolerance.

Although the "*Byzantine generals problem"* is considered unsolvable when there is a single faulty server in a group of three [6], research has shown that dealing with Byzantine failures becomes easier as the number of servers increases [5]. Byzantine fault tolerance research and systems generally fall in to one of two categories, either agreement-based approaches where requests are processed by every server or quorum based approaches were some subset of the servers must process a request in the same manner in order to reach a quorum [1]. In either case, it is expected that requests arrive at servers that have the same starting state [4]. Agreement-based approaches require massive amounts of server to server communications, which degrades throughput performance as the number of servers increase, but they are able to remain reliable through faults across the majority of the servers that comprise a service, albeit at much lower performance [1]. Quorum based processing is able to increase throughput as the number of servers increase, but they can only handle faults in approximately one-third of the servers in the service [5] [4] [1]. In either case, the systems must exhibit *safety* and *liveness* guarantees, where safety refers to a property always being true and liveness referring to a property that will be true in a finite amount of time [5].

In Byzantine fault-tolerant replicated state machines, creators aim for a safety guarantee of linearizability, which is to say that from the client's perspective there is only a single deterministic [4] server that handles requests in the order they are received, and liveness of eventually consistent [5]. Newer research has provided alternatives to linearizability, such as the significantly weaker *fork model* where conflicting state results in the service forking and maintaining multiple states after each conflict, or the slightly weaker *fork consistency model* where services fork to maintain safety but eventually resolve forks to provide consistency [5].

Encryption and signing of data and commands as they flow through, or are stored by, cloud services offers stronger security and provides an easy mechanism for servers in the service to detect arbitrary or malicious faults in other servers or clients [3], but it requires additional computing power throughout the service and in clients. Additionally, properly implemented encryption combats accidental or intentional data leaks, however it also reduces the value add that a cloud service can provide [5] [2]. For example, a cloud service could not generate thumbnails for images that are uploaded in an encrypted format by

clients. While this limitation might be acceptable for IaaS and PaaS offerings, since the infrastructure and platform layers are not generally interested in the application data, it mostly destroys the ability for SaaS offerings to add value beyond being a simple proxy.

Important Ideas

Castro and Liskov [4] introduce a new algorithm for synchronizing object state among distributed nodes, asynchronously, in a Byzantine fault-tolerant scalable system. The algorithm provides linearizability safety guarantees and eventual liveness guarantees. The algorithm provides both guarantees as long as at least two-thirds of the replicas are functioning properly. When failures extend beyond one-third of the replicas, the algorithm provides safety, but not liveness – that is, clients might not receive responses but faulty requests will not corrupt the synchronized state. This safety guarantee is possible because the algorithm requires signed requests, which makes detection of faulty requests as easy as validating the request's signature. The primary downside to the proposed algorithm is that it requires that node failures are independent of each other, which assumes they are not caused by a malicious insider. In order to make this assumption, the authors point out that nodes should be under different administrative control and run different operating systems and hardware – an assumption that is likely not practical in the real world.

The proposed algorithm works by splitting server nodes in to primary and backup nodes, wherein the client sends requests to the primary, the primary relays the request to the backups, and the backups reply to the client. A client considers a request successful when three times the maximum number of acceptable faulty nodes have responded with the same response. The authors demonstrate their algorithm by augmenting a distributed network file system. They are able to show that their Byzantine fault-tolerant object state synchronization algorithm adds only 3% overhead when compared to the baseline implementation of the distributed network file system.

Amir, Nita-Rotaru, Stanton, and Tsudik [2] provide a detailed overview and comparison of several group communication systems, building up to details on their proposed secure version of the Spread group communication system, deemed *Secure Spread*. The authors describe group communication systems (GCS) as:

> *"…distributed messaging systems that enable efficient communication between a set of processes logically organized in groups and communicating via multicast in an asynchronous environment where failures can occur."* [2]

The GCS described by the authors is derived from Spread. Spread provides mechanisms for managing group membership as well as liveness and deterministic guarantees for messages sent through the system. When servers in the group fail, all of their clients are repartitioned among available servers.

The proposed security architecture builds security services on top of a contributory group key agreement mechanism, wherein each group is assigned a unique key for server-to-server communications in order to minimize the number of nodes that a key must be distributed to, thus overcoming scalability issues of prior approaches. Client-to-server communications are protected using keys that are different from the server-to-server keys, and are specific to each client-server connection. While the authors are able to demonstrate impressive performance characteristics on their Secure Spread implementation, the system does not address attacks from insiders.

Abd-El-Malek, Ganger, Goodson, Reiter, and Wylie [1] introduced a highly efficient protocol for synchronizing object state among nodes in a Byzantine fault-tolerant scalable system, which they refer to as the *Query/Update protocol*. The Query/Update protocol is an attempt to reduce server to server communications in a distributed Byzantine fault-tolerant system by moving away from the primary node with backup nodes model presented by Castro and Liskov [4] to a quorum based model. In the quorum based model, clients are free to talk to any server node in the service, and each server node belongs to a quorum group. When requests are received, they are only synchronized among the quorum group prior to sending a response to the client. Once the client response has been sent, synchronization happens across quorum groups to provide an eventually consistent state throughout the service. Like the other systems discussed, the Query/Update protocol uses message signature validation to identify and ignore faulty messages.

The authors demonstrate, through empirical testing of a prototype library merged in to a network file system, that this new protocol consistently outperforms existing agreement based solutions, even as the number of failed nodes increases. The tradeoff for this increased performance, however, is that systems using the described protocol require roughly 40% more

nodes. The authors also provide psuedocode and a correctness proof for their protocol.

Chun, Maniatis, Shenker, and Kubiatowicz [5] propose the use of trusted abstractions of various components within existing Byzantine fault-tolerant state synchronization systems and algorithms in order to improve the node failure limits of existing fault-tolerance mechanisms. The authors present one such abstraction which provides an externally verifiable log of sequential events that can be used by existing systems or algorithms to provide proof to other parts of their system that the sequence was not tampered with. More specifically, the proposed *Attested Append-Only Memory* (A2M) abstraction provides a defense against *equivocation*, that is, a faulty client remaining undetected while providing different faulty data to each client or server it communicates with. A2M is able to achieve this by maintaining a single shared historical view of node interactions, where each new interaction is appended to the end of a chain and then the entire chain, including the latest interaction, is signed. This mechanism ensures that it is not possible to repeat a prior sequence with different data and that it is not possible to retroactively change an interaction.

The paper explores the use of this newly proposed abstraction within two existing protocols and a local storage mechanism, all three of which show promising results. Additionally, the paper demonstrates two variants to prior Byzantine fault-tolerant protocols where liveness and safety are achieved with up to two-thirds of the nodes in a faulty state.

Feldman, Zeller, Freedman, and Felten [3] present a generic framework for making use of untrusted cloud resources in group collaboration efforts, while maintaining the privacy and security of the communications, especially from potential abuse by the untrusted cloud systems or administrators. The authors propose a key exchange and management mechanism for securing communications, which is different from the prior papers reviewed. The previously discussed papers only made use of cryptographic hashes to validate payloads, whereas the present paper uses the exchanged keys both to encrypt communications and to verify them. The keys used for server to server communications are rotated as members join and leave server groups. The authors also propose an operational transformation mechanism that allows nodes in the service to simply fork when a conflict is identified, since the operational transformation mechanism generates transformation functions that can be applied in any sequence by a service node trying to bring itself up to date on operations.

In order to validate their proposal, the authors implement a collaborative text editor and key-value store. The collaborative text editor provides a similar user experience to Google Wave, where multiple users can edit the same document simultaneously. The key-value store simply allows clients to store and retrieve key value pairs, with proper handling for multiple clients writing to the same key simultaneously. Both services demonstrated that the authors' framework is highly scalable and fault-tolerant.

Conclusion

Commercially available cloud hosted distributed services have come a long way over the past ten years, but they still have a lot of room to grow. As systems grow in size, the likelihood of arbitrary component, network, or software failures increases rapidly. As systems add customers, end users, and valuable data, they become increasingly valuable targets to those with malicious intent. Resilience to faults, arbitrary or intentional, will continue to be an area that receives significant research efforts. Similarly, as rogue system administrators, hackers, and government agencies vie for access to the data stored in and transiting the cloud, transport encryption, payload encryption, and encryption of data at rest will continue to be a top priority for both researchers.

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," in *the twentieth ACM symposium on Operating systems principles (SOSP '05)*, New York, NY, USA, 2005.

[2] Y. Amir, C. Nita-Rotaru, J. Stanton and G. Tsudik, "Secure Spread: An Integrated Architecture for Secure Group Communication," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 3, pp. 248-261, July 2005.

[3] A. J. Feldman, W. P. Zeller, M. J. Freedman and E. W. Felten, "SPORC: group collaboration using untrusted cloud resources," in *the 9th USENIX conference on Operating systems*

design and implementation (OSDI'10), New York, NY, USA, 2010.

[4] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *the third symposium on Operating systems design and implementation (OSDI '99)*, New Orleans, Louisiana, USA, 1999.

[5] B.-G. Chun, P. Maniatis, S. Shenker and J. Kubiatowicz, "Attested Append-only Memory: Making Adversaries Stick to Their Word," in *twenty-first ACM SIGOPS symposium on Operating systems principles (SOSP '07)*, Stevenson, Washington, USA, 2007.

[6] L. Lampport, R. Shostak and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382-401, 1982.