Supr: Adaptive Byzantine Fault-Tolerant Replication

Maciej Zbierski

Abstract In the last decade, numerous Byzantine fault-tolerant (BFT) replication protocols have been proposed in the literature. However, practically all of these solutions were designed and optimized only for certain, typically very limited set of environment conditions. Despite previous efforts, no existing BFT replication protocol can guarantee stable and reasonable performance in both correct and faulty environments. In this article we attempt to address this problem by introducing Supr, a novel method for effortlessly combining multiple replication protocols into adaptive BFT solutions, which accommodate to a much wider spectrum of environment conditions than the existing BFT systems. Unlike previous approaches, Supr uses a fine-grained mechanism to monitor the parameters of the execution environment, which enables detecting and counteracting arbitrary faults exhibited in the system. To demonstrate its potential, we use Supr to create a sample BFT solution combining three existing replication protocols, each optimized for different conditions. The performed experiments demonstrate that our approach not only significantly outperforms existing solutions in varying environment conditions, but also does not introduce an observable overhead in stable environments.

Keywords Byzantine fault tolerance · State machine replication · Adaptive BFT · Distributed systems · Dependability

1 Introduction

The last decade has observed an increased interest in creating and deploying world-scale, complex distributed systems. As on-line services and cloud environments become more and more widespread, the demand for guaranteeing their correctness and fault-tolerance has nowadays become high as never before. Many contemporary systems use redundancy to guarantee the correctness of the service

M. Zbierski (✉)
Institute of Computer Science, Warsaw University of Technology, Warsaw, Poland
e-mail: m.zbierski@ii.pw.edu.pl

W. Zamojski et al. (eds.), *Dependability Engineering and Complex Systems*,
Advances in Intelligent Systems and Computing 470,
DOI 10.1007/978-3-319-39639-2_50

even when some server machines crash. However, the recurring incidents reported by the major players in the cloud computing market, including Amazon [1] and Google [8], show that this fault model may not be enough in practical deployments. As a result, the overall interest has recently been shifting towards systems providing a correct service even despite Byzantine (arbitrary) faults. Such solutions can not only tolerate server crashes, but also counteract unpredictable faults, or even malicious attackers. In practice, such model is typically provided through Byzantine fault-tolerant (BFT) replication [3]. In such approach the actual service is deployed on multiple nodes, or replicas, and enhanced with a dedicated coordination protocol to guarantee consistency.

Following the publication of PBFT by Castro and Liskov [3], countless Byzantine fault-tolerant replication protocols have been published in the literature (see for instance [5, 10–12] and references therein). However, each of those solutions has been designed to operate only in certain, very limited set of environment conditions. For instance, PBFT [3] achieves reasonable performance only in local area networks, and HQ [5] operates best for large number of replicas. Furthermore, the majority of these protocols have been optimized for the common case, i.e. fault-free environments, and consequently neglect the performance achieved when some nodes become faulty. For instance, it has previously been shown that practically all contemporary BFT replication protocols are prone to a so called MAC attack [4]. Such vulnerability enables faulty clients to indefinitely disrupt the progress of the whole protocol without being detected, simply by appropriately malforming the authentication codes attached to its request. Even though some BFT replication protocols, such as for instance Aardvark [4], have been designed to guarantee acceptable performance despite in faulty environments, they can exhibit more than 10 times worse throughput than the reference solutions when all nodes are correct. Consequently, despite numerous attempts, none of the existing Byzantine fault-tolerant replication protocols provides acceptable performance in both correct and faulty environments.

In this chapter we attempt to fill this gap by proposing Supr, a novel method for constructing Byzantine fault-tolerant replication systems. Supr views a BFT solution as a composition of self-sufficient replication protocols, which can be independently implemented, tested and replaced. A BFT service constructed using Supr actively monitors the properties of the execution environment and processes the client requests using only the protocol most appropriate for the detected conditions. As a result, Supr can be used to create solutions optimized for much broader range of environment conditions than the existing approaches. In order to demonstrate how the method introduced in this article could be used in practice, we use Supr to create Dali, the first BFT replication protocol which provides high performance in correct environments and still guarantees progress during MAC attacks. The conducted experiments demonstrate that Dali not only significantly outperforms existing BFT replication protocols in varying environment conditions, but also does not introduce an observable overhead in stable environments.

2 Related Work

Several existing BFT replication protocols have distinguished both fast and recovery sub-protocols in order to improve performance in optimistic case, and guarantee liveness in unfavorable conditions respectively. Zyzzyva [11] applies speculative execution whenever all replicas are correct and switches to a slower variant committing the requests only when some claims from the clients have been received. HQ [5] deploys a lightweight quorum-based protocol in favorable conditions and switches to PBFT when contention is detected. CheapBFT [10] uses an optimistic protocol requiring less replicas, but falls back to MinBFT [12] when faults are detected. However, in all these solutions the sub-protocols are tightly coupled with each other, thus increasing the effort required for proper implementation, testing and maintenance. Contrarily, Supr is the first generic approach for creating BFT solutions where each sub-protocol is entirely independent, and can be implemented, proven and replaced irrespectively of the others.

The concept of constructing BFT solutions comprised of multiple protocol instances has initially been proposed in Abstract [9]. Apart from acting as a traditional BFT protocol, each Abstract instance can decide to abort a client request whenever it cannot provide progress. In such situation, the aborted request is relayed to a different instance according to a predefined order. However, unlike Supr, Abstract does not proactively change the instance processing client requests to improve the achieved performance based on the detected environment conditions.

Bahsoun et al. [2] have recently proposed Adapt, a derivative of Abstract, which does not require a predefined order in which the instances are aborted. Instead, Adapt monitors the performance of the execution environment, such as the obtained throughput or size of incoming requests, and uses these parameters to select the protocol instance optimal for the detected environment conditions. However, contrary to Supr, Adapt struggles to provide acceptable performance under arbitrary faults, as the execution environment monitoring is not directly coupled with protocol instances. Furthermore, both Abstract and Adapt are likely to involve higher implementation and maintenance costs than Supr due to possible hidden dependencies between the sub-protocol instances. Finally, although Adapt does not require a predefined policy, but uses machine learning to determine the most appropriate instance, this approach can also be applied in Supr without an excessive effort.

3 Supr Architecture

This section introduces and describes Supr (sub-protocols), a new approach for effortlessly defining and implementing complex Byzantine fault-tolerant protocols. Each BFT solution created using the proposed method is viewed as a composition of so called Supr instances. Every instance represents an independent state machine replication protocol, which processes client requests and generates responses, much like existing BFT protocols. At any moment exactly one instance is active, which means it is designated to consume the client requests and process them according to the underlying protocol specific for that instance. The non-active instances on the other hand do not receive nor execute new client requests, although they still can communicate with their corresponding instances on other nodes.

In order to oversee execution, Supr deploys on each replica an additional supervisor module, which is shared between all protocol instances. The supervisor acts as a multiplexer, intercepting all incoming messages and forwarding them to the corresponding Supr instance. Apart from that, the supervisor collects the information about the properties of the execution environment, such as the observed throughput, average request size in previous batches, etc. These parameters are reported directly by both active and inactive protocol instances. Additionally, instances report to the supervisor any issues regarding the protocol execution. These can include, but are not limited to, faults suspected in the system, observations about any suspicious behavior or notices concerning insufficient progress. Finally, the supervisor maintains and manages a transition policy, which defines how the observations reported by the sub-protocols should affect the selection of the active instance. The rules included in the policy should typically represent both the design goals and the properties of every Supr instance, such as unfavorable conditions, additional assumptions, etc.

The supervisor constantly monitors the information obtained from the sub-protocols, and by confronting them with the policy, it might decide to change the active instance by initiating a transition. When that happens, the supervisor selects the new active instance based on the transition policy and enqueues any subsequent requests received from the clients, instead of relying them to Supr instances. Immediately after all the requests received by the previously active instance have been processed, the supervisor finishes the transition by relaying the enqueued client messages to the new active instance.

The second module deployed on each replica and shared between all Supr instances is the sequencer, responsible for assigning unique identifiers to incoming requests. Instead of using its own protocol-specific method, every instance is required to query the sequencer module whenever it wishes to assign a client request with a unique identifier. This approach guarantees that the requests are ordered between multiple instances, which facilitates transitions between the protocols. Additionally, the sequencer is responsible for maintaining an identifier cache, enabling detecting duplicate client requests. The communication pattern between Supr modules located on a single node, including n sub-protocol instances, is depicted in Fig. 1. The diagram illustrates the process of message multiplexing performed by the supervisor and communication between the active instance and the sequencer module, according to the description presented above.

The process of converting existing implementations of traditional BFT replication protocols into Supr instances involves modifying the protocol to communicate with the modules described above and in most cases is straightforward and easy to perform. Firstly, the code responsible for assigning unique identifiers to incoming

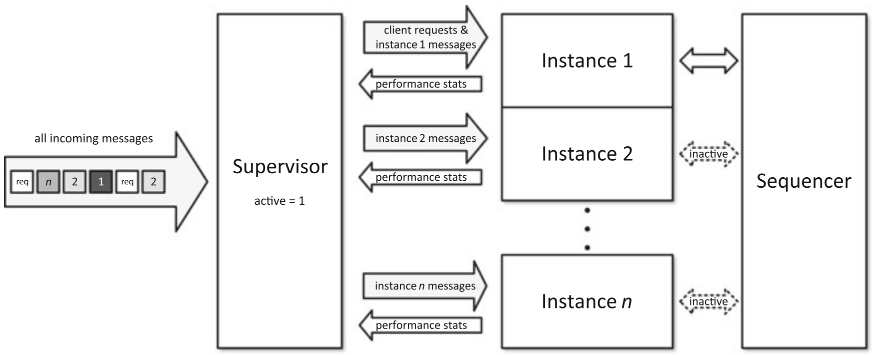

Fig. 1 Communication pattern between Supr modules located on a single replica

requests needs to be replaced with a query to the shared sequencer. Secondly, the implementation of the BFT protocol has to be enhanced to relay progress statistics to the supervisor wherever applicable, either periodically or on demand. The exact location where such functionality should be implemented depends on the properties of the instance and the choice of factors triggering transitions to other instances. Finally, the messages exchanged by each Supr instance should be enhanced with a field unique to that instance. This property is used by the supervisor to forward the received messages to their appropriate destinations, although in some cases this might not be necessary, for instance when the messages exchanged by the adapted protocol already differ somehow from the messages issued by the other instances.

4 Dali Protocol

In order to demonstrate how the approach introduced in this article can be applied in practice, this section uses Supr to create Dali, the first Byzantine fault-tolerant replication protocol which survives MAC attack, and at the same time does not degrade the performance achieved in correct environments. Dali consists of three Supr instances, each implementing an existing BFT replication protocol optimized for different environment conditions. The remainder of this section first provides a brief description of these protocols, and then introduces the corresponding transition policy used by the supervisor.

The first Dali instance uses PBFT [3], a protocol considered as a baseline solution in the field of Byzantine fault-tolerant replication. PBFT designates a single replica to perform the role of the primary, i.e. coordinate protocol execution by imposing a total ordering on client requests. If the remaining servers suspect that the current primary is faulty, they initiate a vote to elect among themselves the replica that will perform the role of the next primary. A similar approach is followed in the large majority of contemporary BFT replication solutions, including the protocols used in the remaining Dali instances. PBFT uses a traditional three-round consensus to guarantee that the identifier assigned by the primary to a client request is correct. This means that PBFT needs to perform three all-to-all communication rounds between the replicas before each client request can be executed. To guarantee communication integrity, the exchanged messages are enhanced with message authentication codes (MAC).

The second Dali instance implements Zyzzyva [11], a protocol decreasing replication costs through speculation. Unlike PBFT, Zyzzyva optimistically executes client requests without prior consensus. When all replicas are correct, this requires only one communication round to be performed between the servers. However, if the clients are unable to receive a consistent response from *all* replicas before an assumed timeout, potentially because some nodes are faulty, they issue claims addressed to the replicas. These are subsequently used to establish a consistent state on all nodes and reach a consensus on a valid response to the corresponding client requests. Consequently, even though Zyzzyva can provide very high throughput and low latency in correct environments, the additional communication rounds and the delay imposed by the clients can significantly degrade the achieved performance whenever some nodes are faulty. Much like in PBFT, the exchanged messages are authenticated with MACs.

Finally, the third Dali instance deploys Aardvark [4], which, unlike both PBFT and Zyzzyva, guarantees progress during MAC attacks. Although Aardvark bases on the same three-round consensus as PBFT, it uses a combination of MACs and public-key signatures to authenticate the exchanged messages. As a result, the clients can no longer generate incorrect message authentication codes without being detected, since public-key signatures provide transferable authentication. However, at the same time this approach degrades the performance achieved by the protocol, since public-key signatures are more than an order of magnitude slower than message authentication codes [3].

As has been already observed in the literature, each of the protocols presented above performs best in different environment conditions. Zyzzyva provides the best throughput and latency in correct environments, but can be expected to perform worse than PBFT in unfavorable conditions, i.e. when some replicas or clients are faulty [6]. Furthermore, while only Aardvark is resistant to MAC attacks, it provides the worst performance in correct environments due to the additional cost of signature verification [4]. Consequently, when designing Dali, we have reflected these properties in its transition policy. As a result, Dali supervisor select Aardvark whenever a MAC attack is detected, chooses Zyzzyva if no faults are observed, and in remaining situations switches to PBFT.

Each Dali instance monitors the protocol execution and informs the supervisor about the observed anomalies. Dali transition policy uses two main factors to determine which instance should be activated by the supervisor. The first one is the frequency of Zyzzyva-specific claims received from the clients and their relation to the overall number of recently executed requests. When no client claims have been received for an assumed amount of time, all replicas can be considered correct, and an optimistic execution of Zyzzyva can be used to increase the overall performance.

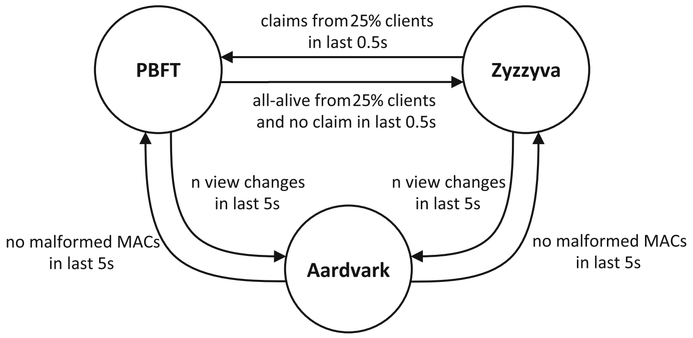
PBFT
Zyzzyva
Aardvark
claims from 25% clients in last 0.5s
all-alive from 25% clients and no claim in last 0.5s
n view changes in last 5s
n view changes in last 5s
no malformed MACs in last 5s
no malformed MACs in last 5s

Fig. 2 A transition graph for Dali

The second decision factor is the frequency of primary changes triggered by an incorrectly authenticated request. This is because frequent and recurring primary changes typically indicate a MAC attack. The transition policy used in Dali is presented in Fig. 2. The quantitative parameters of the policy have been selected experimentally to minimize the amount of time required to react to changes in environment conditions, and at the same time prevent premature transitions to less resilient instances in situations where the faults are not exhibited only temporarily. Additionally, after each observed premature transition the time intervals used by the policy are additionally doubled to prevent such situations in the future.

5 Experiments and Evaluation

In order to verify our approach we have created stand-alone implementations of PBFT, Zyzzyva and Aardvark, and subsequently adapted them as Dali instances using the Supr approach introduced in this article. Additionally, each implemented protocol has been enhanced to take advantage of the cost-aware batching optimization introduced by the author [13]. The supervisor module has been setup to select active instances according to the transition policy described in the previous section. The test environment consisted of four servers, each equipped with a 3.4 GHz processor, 4 GB RAM and a Gigabit Ethernet controller, plus an additional machine hosting the client processes, all connected to the same local area network.

The performed experiments compare the throughput achieved by Dali and its reference protocols and analyze how Dali reacts to changes in environment conditions. Throughout the tests we consider throughput as the maximum number of client requests per second that could be processed by the service replicated using the respective protocol. The experiments do not directly compare Supr to the other existing adaptive BFT approaches, such as Abstract [9] or Adapt [2], as they would have to be additionally modified to implement a protocol with a functionality similar to Dali, since none of them is capable of sufficiently tolerating faults of malicious origin.

The goal of the first experiment was to analyze the performance achieved by Dali running PBFT and Zyzzyva instances, and estimate the cost of performing transitions between them. The experiment started in a correct environment, where all replicas processed incoming requests according to respective protocols. At a certain moment, one of the replicas was manually suspended to simulate a node failure. After some time, the failed replica was restarted, and it subsequently resumed processing client requests. The throughput achieved by the considered protocols throughout the duration of the experiment is presented in Fig. 3. While Dali started the experiment in Zyzzyva instance, the supervisor initiated a transition to PBFT after the replica failure has been detected, i.e. immediately after enough client claims have been received. Similarly, after the failed replica was restarted and the supervisor was no longer receiving client claims, it changed the active instance back to Zyzzyva.

The throughput achieved by Dali has demonstrated to be practically indistinguishable from its reference protocols in both analyzed environment conditions, with the maximum difference below 5 %, and an average around 1 %. Additionally, Dali has achieved the highest average throughput of all protocols despite the change in environment conditions, with an exception of Zyzzyva for requests with large payloads. However, what cannot be observed in the figure, is that in such case Zyzzyva provided around 50 % higher request latency than Dali. While we have configured Dali policy to remain in PBFT in these conditions, in environments where the high throughput is the most important factor, the policy could be further modified in favor of Zyzzyva. The time required to perform a transition from Zyzzyva to PBFT has been dominated by the delay imposed by the clients and has demonstrated to be very similar to the one observed for Zyzzyva alone. Finally, although Dali requires more time to switch back to Zyzzyva in order to prevent

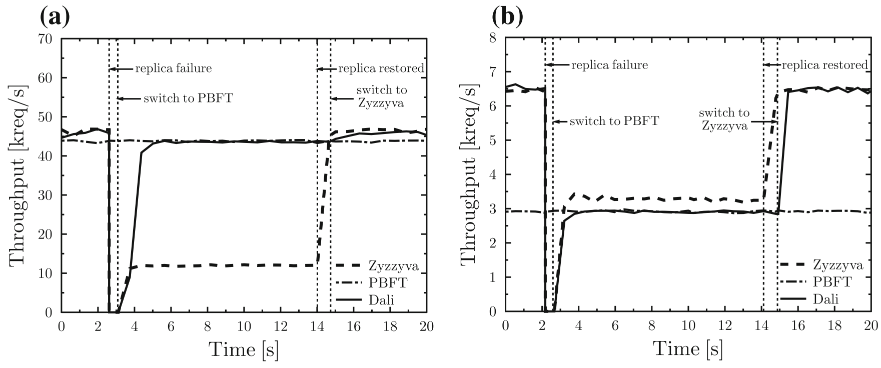

Fig. 3 Comparison of changes in throughput imposed by a single faulty replica. **a** Payload = 0 kB, **b** payload = 4 kB

premature transitions when the replica is still faulty, the switching itself does not exceed 1 s and still remains at an acceptable level.

The second experiment aimed at analyzing the behavior exhibited by Dali and the reference protocols during a MAC attack. The test started in a fault-free environment, and at a certain moment a single client started periodically issuing requests with malformed MACs [4]. After some time the faulty client has been turned off and the execution environment again became correct. Figure 4 presents the throughput achieved by the analyzed protocols throughout the test. While Dali started the experiment in Zyzzyva instance, as soon as its supervisor has detected the MAC attack, exhibited by four consecutive primary changes, it initiated a transition to Aardvark. Subsequently, the supervisor switched back to Zyzzyva after no malformed requests had been received for 5 consecutive seconds, as defined by the transition policy.

During the MAC attack both Zyzzyva and PBFT were unable to process any incoming requests. This is the result of recurring primary changes triggered by the replicas due to incorrect message authentication codes generated by the faulty client. Analogically, while Aardvark maintained progress during MAC attack, it exhibited even more than 10 times worse throughput than the remaining protocols whenever the environment was correct. On the other hand, Dali both provided high throughput in the fault-free environment and exhibited performance similar to Aardvark during the MAC attack. Although the time spent on performing transitions was slightly larger than in the previous experiment, it oscillated around 1 s and still be considered acceptable. Finally, as described earlier, the additional 5 s delay has been introduced by the policy after the end of the MAC attack to provide a trade-off between the achieved performance and the cost of a premature transition. However, this delay could be reduced to the level observed for the other transitions, provided the supervisor is additionally enhanced with appropriate client blacklisting mechanisms [4].

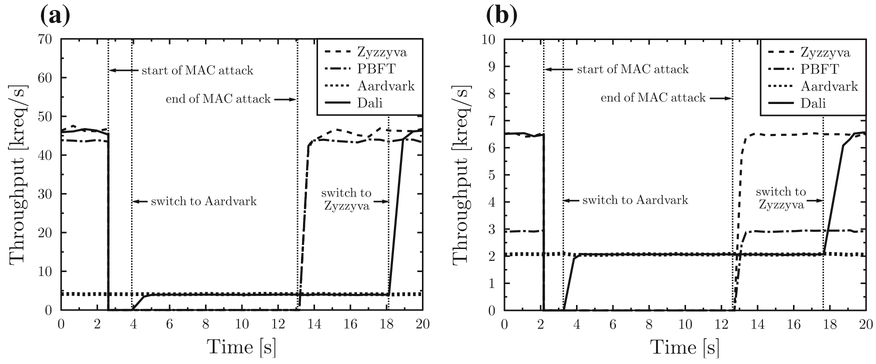

Fig. 4 Comparison of changes in throughput imposed by a faulty client issuing malformed MACs. **a** Payload = 0 kB, **b** payload = 4 kB

6 Conclusion

In this article we have presented Supr, a new method for creating adaptive Byzantine fault-tolerant replicated services. Unlike previous approaches, Supr determines the properties of the execution environment based on the notifications obtained directly from sub-protocol instances. This way the resulting solutions can react to more subtle environment changes and counteract less evident performance degradation caused by arbitrary faults. This makes Supr the first generic method for creating Byzantine fault-tolerant solutions which guarantee high performance in both correct and faulty environments. Furthermore, not only transforming existing protocols to Supr instances requires far less effort than in previous approaches, but also the resulting solution is easier to modify and reason about.

Additionally, to demonstrate that the proposed method can be used in practice, we have used Supr to create Dali, the first Byzantine fault-tolerant solution which provides performance competitive with contemporary BFT replication protocols in correct environments, and at the same time survives the MAC attack [4]. In the performed experiments Dali has demonstrated to easily outperform the other analyzed protocols under varying environment conditions. Additionally, while operating within a single sub-protocol instance, Dali has introduced no observable overhead over its reference protocols.

However, it is worth noting that Dali is by no means the only adaptive BFT solution that could be created using Supr. In the future, we plan to extend Dali with additional sub-protocols to make it adapt even better to different environment conditions. Additionally, we intend to utilize Supr to create a commercial-grade BFT replicated service and use it to enhance the scope of performed experiments, including the additional tests with contemporary fault injection techniques [7].

References

1. Amazon: Summary of the Amazon DynamoDB service disruption and related impacts in the US-East region. https://aws.amazon.com/message/5467D2/. Accessed September 2015
2. Bahsoun, J.P., Guerraoui, R., Shoker, A.: Making BFT protocols really adaptive. In: 2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 904–913. IEEE (2015)
3. Castro, M., Liskov, B.: Practical Byzantine fault tolerance. In: Proceedings of the Third Symposium on Operating Systems Design and Implementation, OSDI 1999, pp. 173–186. USENIX Association, Berkeley (1999)
4. Clement, A., Wong, E., Alvisi, L., Dahlin, M., Marchetti, M.: Making Byzantine fault tolerant systems tolerate Byzantine faults. In: Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, pp. 153–168, NSDI'09 (2009)
5. Cowling, J., Myers, D., Liskov, B., Rodrigues, R., Shrira, L.: HQ replication: a hybrid quorum protocol for Byzantine fault tolerance. In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation, pp. 177–190, OSDI'06. USENIX Association, Berkeley, CA, USA (2006)

6. Duan, S., Peisert, S., Levitt, K.N.: hBFT: Speculative Byzantine fault tolerance with minimum cost. IEEE Trans. Dependable Sec. Comput. **12**(1), 58–70 (2015)
7. Gawkowski, P., Rutkowski, T., Sosnowski, J.: Improving fault handling software techniques. In: 2010 IEEE 16th International On-Line Testing Symposium, pp. 197–199. IEEE (2010)
8. Google: Today's outage for several Google services. https://googleblog.blogspot.com/2014/01/todays-outage-for-several-google.html. Accessed January 2014
9. Guerraoui, R., Knežević, N., Quéma, V., Vukolić, M.: The next 700 BFT protocols. In: Proceedings of the 5th European Conference on Computer Systems, EuroSys 2010, pp. 363–376 (2010)
10. Kapitza, R., Behl, J., Cachin, C., Distler, T., Kuhnle, S., Mohammadi, S.V., Schröder-Preikschat, W., Stengel, K.: CheapBFT: resource-efficient Byzantine fault tolerance. In: Proceedings of the EuroSys 2012 Conference (EuroSys 2012), pp. 295–308 (2012)
11. Kotla, R., Clement, A., Wong, E., Alvisi, L., Dahlin, M.: Zyzzyva: Speculative Byzantine fault tolerance. In: Symposium on Operating Systems Principles, SOSP (2007)
12. Veronese, G.S., Correia, M., Bessani, A.N., Lung, L.C., Verissimo, P.: Efficient Byzantine fault tolerance. IEEE Trans. Comput. **62**(1), 16–30 (2013)
13. Zbierski, M.: Cost-aware request batching for Byzantine fault-tolerant replication. In: Theory and Engineering of Complex Systems and Dependability. Proceedings of the Tenth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX, June 29–July 3 2015, Brunów, Poland, pp. 583–592. Springer (2015)