Scalable BFT for Multi-Cores: Actor-Based Decomposition and Consensus-Oriented Parallelization*

Johannes Behl
TU Braunschweig
behl@ibr.cs.tu-bs.de

Tobias Distler
FAU Erlangen-Nuremberg
distler@cs.fau.de

Rüdiger Kapitza
TU Braunschweig
rrkapitz@ibr.cs.tu-bs.de

Abstract

To pave the way for Byzantine fault-tolerant (BFT) systems that can exploit the potential of modern multi-core platforms, we present a new parallelization scheme enabling BFT systems to scale with the number of available cores and to provide the performance required by critical central services. The main idea is to organize parallelism around complete instances of the underlying multi-phase BFT agreement protocols, and not around single tasks (e.g., authenticating messages), as realized in state-of-the-art systems. We implemented this *consensus-oriented parallelization scheme* on basis of a BFT prototype that permits flexibly configured parallelism by relying on an *actor decomposition*. In an early evaluation conducted on machines with twelve cores, the consensus-oriented parallelization achieved over 200% higher throughput than a traditional approach while leaving the potential to utilize even more cores and exhibiting a significantly greater efficiency in a single-core setup.

1 Introduction

The sheer complexity of today's hard- and software systems makes it virtually impossible to prevent the occurrence of errors during production operation. Indeed, many central, heavily-used services [6, 10] are deployed with multiple replicas instead of single instances to ensure their availability. So far, these replicated systems are primarily meant to withstand merely crashes of system components or machines; with respect to the vast range of possible malfunctioning and the risk of malicious attacks, this is insufficient for a lot of applications.

A logical consequence would be the use of Byzantine fault-tolerant (BFT) state-machine replication [4, 18], which can be employed to immunize systems against a bounded number of arbitrary faults. But although substantial progress has been made in this research area over the last 15 years [4, 11, 15, 21], a common adoption of this technique is still put on hold. One reason is that most works addressed the theoretical foundations and general feasibility of the underlying *BFT agreement protocols* used to reach consensus among replicas; the actual implementation of these protocols, however, was considered secondary at most. In fact, we are only aware of a single ongoing project that directly addresses their concrete realization [3]. Accordingly, most BFT prototypes settle for a plain implementation and neglect concurrency and parallel execution within each replica, two issues that tend to introduce considerably more complexity.

State-of-the-art implementations follow a straightforward approach where the agreement protocol is decomposed into several tasks like sending messages, authenticating messages, and managing clients, and where some of these tasks are executed concurrently in different threads [2, 3, 4, 5]. This *task-oriented parallelization scheme* has one important drawback: With the degree of parallelism being bound to the number of defined protocol tasks, there are only limited possibilities to adapt to to the number of available processor cores. As a result, state-of-the-art BFT systems cannot make efficient use of modern processors drawing their computational power more from an increasing number of cores than from single-core performance. Considering the high computational resources BFT agreement protocols require [4], this deficiency restricts the performance of contemporary BFT systems significantly, and thereby their practical use especially for critical, heavily-utilized services.

Following this, we present a parallelization scheme for BFT agreement protocols that organizes parallelism around instances of these multi-phase protocols and not around tasks carried out for each instance. This *consensus-oriented parallelization scheme* allows scaling the possible throughput of replicated systems with the demand and the number of processor cores offered by the executing platform. In conjunction with a flexible implementation decomposed on basis of actors the scheme leads to an easier development of replicated systems that can exploit the potential of modern multi-cores.

*This work was partially supported by the German Research Council (DFG) under grant no. KA 3171/1.

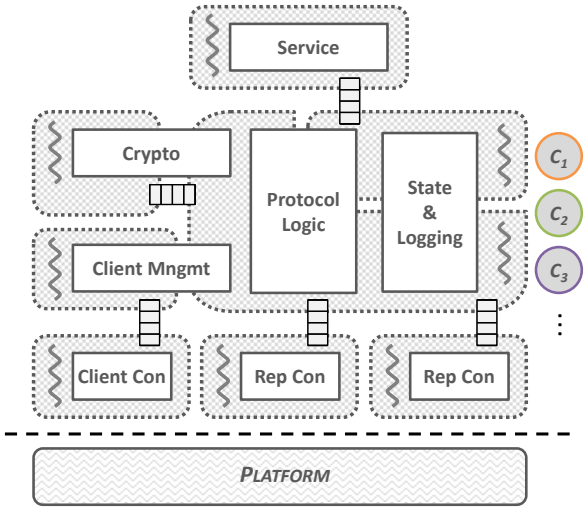

Figure 1: Threading model of contemporary BFT implementations: The replication protocol is realized through several functional modules. Threads are mainly organized around these modules and interact via message queues or shared data. Single consensus instances affect all modules, thus are processed within multiple threads.

Our first evaluations show that the consensus-oriented parallelization can achieve over two times higher throughput on a twelve-core machine than a state-of-the-art task-oriented parallelization. Additionally, it has the potential to use further processing capacities whereas the traditional approach is limited by the slowest component. Even in a single-core setup, the increase of efficiency is equally significant. In combination, both proposed concepts lay the foundation for systems that are able to preserve their availability and integrity even in the presence of, for instance, intrusions while providing the degree of performance required by critical central services.

2 State-of-the-Art BFT Implementations

In BFT systems, the purpose of agreement protocols is to guarantee a consistent behavior between the multiple instances of the replicated service, that is, to reach a consensus among all participating replicas about the processing of service requests. Regarding their capabilities to benefit from multi-core platforms, the architectures of contemporary implementations of such protocols [2, 3, 4, 5] usually look as the one shown in Figure 1: The system is divided into several functional modules, each carrying out one or more tasks that are required to realize the agreement protocol. For instance, interaction with clients is handled by a client management module. Another module implements the logic of the agreement protocol. The main task of the protocol is to ensure that all replicas execute the same requests in the same order. The ordering is achieved by running separate instances of the plain protocol, separate consensus instances (C_x), each comprising multiple message communication phases between replicas. During this procedure, the protocol logic module can rely on additional modules to fulfill tasks like authenticating and logging messages and managing replica state. Finally, requests are executed by the service implementation according to the determined order. In these systems, the execution is parallelized by organizing threads partly around the functional modules and hence the tasks they are responsible for. This leads to a *task-oriented parallelization* [17] with threads interacting via message queues or shared data.

Although this approach can make use of multi-core processors to some extent, it has several drawbacks: (1) The degree of parallelism is limited because task-oriented parallelization entangles concurrent activities with functional modules. If there are more cores available than tasks to be carried out, it is difficult to utilize them all. (2) On the other hand, when there are too many threads, this could entail unnecessary high scheduling overhead including inefficient usage of processor caches. All in all, aligning the number of threads to the number of cores is hardly possible. (3) Additionally, the maximum performance of the system is determined by the slowest task since single consensus instances are processed in cooperation of more or less all modules in a pipelined fashion. This is problematic as the extensive use of cryptographic operations in BFT systems for message authentication can quickly become a bottleneck [4]. (4) Messages and other data belonging to consensus instances must be exchanged between the multiple threads executing the different tasks, which results in a large number of contention points impairing the scalability of the system further. (5) Moreover, such an approach could facilitate situations where threads unnecessarily rely on shared data, for example when central modules are accessed. This makes the implementation more error-prone as well as harder to understand and to maintain due to required synchronization, while potentially introducing even more contention points.

3 Actor-Based Decomposition

Following these observations, we first developed a prototype that implements a standard BFT replication protocol, namely PBFT [4], using an *actor-based decomposition* of replicas. Here, the functional modules, called actors [1], interact exclusively by means of asynchronous message passing; hence, they do not share any modifiable state. Due to actors being executed by at most one thread at each point in time, only the queues used for message passing have to be synchronized. Moreover, while processing messages, actors are not allowed to use blocking operations. This way, they define concurrent tasks that can share threads of execution.

Relieving the protocol implementation of synchronization concerns by means of this actor-based decomposition eases the development essentially. The resulting code is easier to understand and to maintain due to a clear concept and the absence of disturbing synchronization mechanisms. Moreover, the strong decoupling of functional modules in form of actors supports an even further separation of protocol tasks by deploying them on different machines as proposed in the literature [5, 21]. The most important advantage with regard to multi-core processors, however, is the strict distinction between concurrency and (potential) parallelism, the distinction between concurrent activities of the protocol realization and the threads eventually executing them, which accompanies the pure actor-based approach. This enables flexible thread and dispatching configurations that are aligned to the processor topology of the executing platforms.

4 Consensus-Oriented Parallelization

Despite the benefits of our actor-based BFT implementation compared to contemporary approaches, some limitations remain: Single instances of consensus are still processed by multiple actors, each fulfilling its particular part, such as connection handling, message authentication, protocol execution, etc. If two dependent actors are executed by different threads on different cores, not only the synchronization overhead for the internal actor-to-actor message passing becomes noticeable. Also the usage of the processor caches gets inefficient, because in that case, the message passing has to be carried out by means of slower caches. If the involved cores are located at different processors of the system, this already expensive procedure becomes yet more expensive [7]. In the worst case, an actor-based BFT implementation suffers from similar problems as contemporary systems in this regard. The same holds for the maximum achievable degree of parallelism, since the parallelization scheme of an actor-based decomposition as presented so far still remains task-oriented.

The Concept To solve these problems, we propose a *concurrency and parallelization scheme oriented on complete instances of consensus* instead of certain tasks. The idea is, broadly speaking, to lower the data dependencies between different consensus instances and to bind all messages and tasks related to a specific instance to a particular processor core. This way, the throughput of the agreement protocol of a replicated service is able to scale with the demand and the number of available cores by adapting the number of consensus instances carried out independently on dedicated cores in parallel.

Figure 2 illustrates this concept: Actors realizing the replication protocol are organized in *pillars* where each pillar is responsible for certain instances of consensus and executed by a dedicated thread. Overall, the scheme

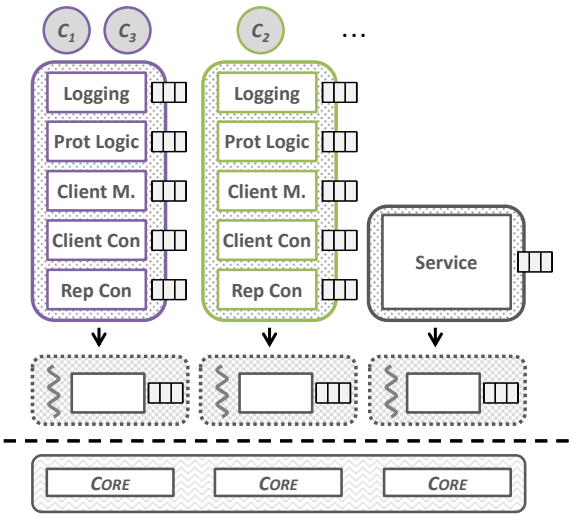

Figure 2: Consensus-oriented parallelization. Instances of functional modules are organized in separate pillars executing all tasks related to associated consensus instances. Pillars are assigned to dedicated threads and their number is aligned to the number of available cores.

works as follows: If a client sends a request, it is received by the responsible connection actor and forwarded to the client management actor located at the same pillar. Then, the actor, or module respectively, that executes the protocol logic for that pillar initiates a new consensus instance. The entire message exchange with other replicas during the agreement phase is carried out via the pillar's network connections. All cryptographic operations can be executed where they arise and do not need to be transferred to other actors in order to enable concurrent processing. If a consensus has been reached, the corresponding requests are forwarded to the central service actor, which executes the requests in the appropriate order.

Splitting the Protocol Logic To realize this, the first step is to split the actor implementing the protocol logic. Using PBFT [4] as basis, the data mainly shared between consensus instances is the consecutive sequence number that has to be increased for each new instance. This sequence number identifies the messages belonging to an instance and finally determines the order in which the corresponding requests have to be executed by the service. At this point, PBFT and similar protocols make use of a distinguished replica in the group, the leader, which is responsible for making a proposal about which requests are to be ordered in the next consensus instance.[1]

To enable multiple concurrently working protocol instances, the space of sequence numbers can be parti-

[1] Is the current leader suspected by the other replicas to be faulty, a so-called view-change protocol is carried out to eventually elect a new leader. This and other sub-protocols of PBFT are not affected by the proposed parallelization scheme and can be used without modifications since the entity of a replica is not altered by the basic concept.

tioned. If N_P protocol actors shall be used, each is responsible for the consensus instances $c(i, j) = i + jN_P$ with i being the identifier of the actor and j a locally increased counter. When a request is received by the client management actor of the current leader, it forwards it to a protocol actor, for instance with a round-robin strategy. Subsequently, if a protocol actor i has received unordered requests, it initiates the next consensus instance $c(i, j)$ and increments its local counter j. During the execution of the agreement protocol for $c(i, j)$, all messages belonging to it are directly forwarded from the actors handling replica connections to the responsible protocol actor. The correct protocol actor i can be easily determined by the sequence number $n = c(i, j)$ contained in all protocol messages and the formula $i = n \bmod N_P$. After an instance has been finished, that is, the sequence number and corresponding requests have been approved, the requests can be processed by the actor that integrates the service. Here, this actor has to ensure that requests are executed in the order defined by the sequence numbers.

Splitting the Replica Connections So far, actors carrying out the protocol logic for assigned consensus instances are able to make progress independently from each other, but when a message is received on a replica connection, this message has to find its way through the processor caches before it can be handled by the associated protocol actor. To mitigate this effect, not only the protocol implementation but also the actors responsible for the replica connections can be split. This naturally leads to the usage of multiple network sockets or connections for the communication between every pair of replicas.[2] Consequently, each protocol actor gets its own set of replica connections and connection actors assigned, leading to increased cache efficiency.

Splitting the Client Management Also the client management in conjunction with the client connections can be distributed among all protocol actors or consensus pillars, respectively. This reduces potential contention points, increases the achievable parallelism, and enhances cache efficiency further. However, it requires additional considerations: If clients are exclusively assigned to single protocol actors, there could occur situations where the actor responsible for instance $n + 1$ received requests that can be ordered but the actor responsible for instance n is idle. Since requests have to be executed in consecutive order of sequence numbers, it is not allowed to execute the requests of instance $n + 1$ until n was carried out. As a result, the whole system could stall if clients connected to the protocol actor for n do not send any requests and thus prevent the execution of n.

[2]This also facilitates flow steering techniques as realized by operating systems (`http://lwn.net/Articles/382428/`) or even network adapters and which try to optimize the cache usage from the reception of a packet at the adapter up to the delivery to the application.

There are several strategies to solve this and to enable a divided client handling: If a protocol actor has no requests to order, it can signal this to the others. Then, the instance could be exceptionally executed by another protocol actor that does have unordered requests. Moreover, unordered requests as well as entire client connections could be migrated to the idle actor installing a form of load balancing. For the possible case that there are no more requests to order but instances with higher sequence numbers have already been agreed on, empty consensus instances must be carried out effectively skipping the instances and closing the gap to the instances containing requests [13]. Another approach would be to replace the described a priori partitioning of sequence numbers with a dynamic assignment. Here, an arbiter actor would allocate sequence numbers or blocks of numbers for protocol actors on demand, additionally providing opportunities for constant load balancing.

The Results Other actors (e.g., the actor handling state and message logging) can be split similarly to the examples described. Further, different configurations are possible: Clients could be managed by central actors shared by all or some pillars. It is also not mandatory to employ multiple connections between replicas. The actor-based decomposition still allows a flexible configuration that optimally fits the needs of specific deployments.

All in all, the advantages of a consensus-oriented parallelization scheme are manifold: (1) If consensus pillars are assigned to single threads that in turn are aligned to the number of available processor cores, ideally client requests, protocol messages, and everything else belonging to a consensus instance are accessed by the same core up to the point where the instance has been finished and the results are delivered to the service; hence, the processor caches come to their full potential. (2) Since there are only a few, if any, interactions between actors of different pillars, the need for synchronization and the number of contention points are reduced significantly, further improving the system's efficiency. (3) It gets easier to balance load across homogeneous processor cores since all pillars have to carry out roughly the same work, assumed that differences related to requests are balanced out. Compared to a pure actor-based decomposition, this simplifies the thread configuration considerably. (4) Development is simplified as well. The plain replication protocol can be implemented in a similar way as it is specified. There is no or at least less need for transferring tasks to concurrent modules and thereby obfuscating the control flow solely for improving the run-time performance. (5) Last but not least, the performance of systems can scale with the demand and available computational and networking capacities. The more cores a platform comprises, the more pillars can be used and thus the more consensus instances can be executed in parallel.

5 Evaluation

In a first evaluation of our proposed concepts, we compare a BFT system that follows a state-of-the-art approach with task-oriented parallelization and no distinction between concurrency and parallelism, an implementation which separates concurrent tasks from threads, and a configuration using a consensus-oriented parallelization regarding their maximum throughput in a scenario with an increasing number of available processor cores.

As starting point for all three compared approaches, we use our actor-based BFT prototype implementation written in Java. To measure the behavior of the state-of-the-art approach, where threads are bound to concurrent tasks and cannot be aligned to the available processor cores, we configure it with a fixed number of threads for all tested core configurations. It can be assumed that this setup is very favorable for the traditional approach since the basis is still an optimized actor-based implementation with a clear concurrency architecture and defined synchronization points at the message queues. Comparing other existing BFT prototypes is left for future work. To test the consensus-oriented parallelization, we split single instance actors as described in the last section.

The measurements are conducted on five machines equipped with two 2.5 GHz six-core processors. Each core offers two hardware threads via simultaneous multithreading. Different numbers of available cores are achieved by confining the process of the Java Virtual Machine to certain cores. The machines are connected using three switched gigabit Ethernet adapters. Four machines host a replicated counter service, the remaining machine simulates 400 clients continuously issuing synchronous requests. To measure the scalability regarding the maximum achievable number of consensus instances per second, each instance orders solely one request.[3] All measurements are carried out five times; results are averaged.

The results are presented in Figure 3. Using a single core with two hardware threads, the actor-based approach is able to process 170% more requests than the state-of-the-art approach since the latter suffers from high scheduling overhead. The consensus-oriented approach achieves even 20% more throughput than the task-oriented actor-based configuration. It utilizes both hardware threads more uniformly. With an increasing number of cores, both task-oriented approaches quickly reach their limits primarily due to the authentication of messages; at some point, the cryptographic operations on messages cannot be further parallelized beneficially. Providing all available twelve cores to the replicated system, the consensus-oriented approach is able to leverage its fully parallelized execution of consensus instances

[3]Thus, the option to order batches of requests with single instances is not used. It would increase the throughput of all approaches equally.

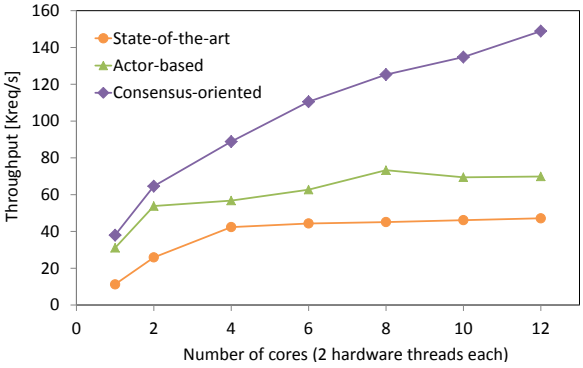

Figure 3: Comparison of parallelization schemes regarding their scalability in terms of achievable throughput with increasing number of cores.

and its better cache efficiency. It achieves a throughput twice as high as the task-oriented actor-based variant and a throughput three times as high as the state-of-the-art approach while still having the potential for even higher rates with an increasing number of cores.

6 Related Work

Many efforts have been made to increase the efficiency of BFT agreement protocols by reducing the number of exchanged messages. Proposals range from the use of hybrid fault models [11, 19], over speculative approaches [15], to different modes of operation [8, 11].

Bessani et al. [3], by contrast, investigate the practical realization of single replicas of standard BFT protocols. Their software architecture borrows from staged event-driven architectures (SEDA) [20]. This leads to a task-oriented parallelization scheme that does not distinguish between concurrency and parallelism. The deficiencies of such a design were already discussed in this paper.

Consensus-oriented parallelization can be regarded as a form of vertical process architecture as described by Schmidt et al. [17] for transport layer protocols. Contrary to these protocols, however, BFT agreement protocols require several phases of message exchange in a group communication setting and have dependent instances.

Looking at crash fault-tolerant systems, Santos and Schiper [16] present an architecture that is explicitly devised for multi-core platforms. They state that their design is based on SEDA as well as the actor model. However, they also state that they deliberately break the actor model at several points. All in all, their proposed architecture is very similar to the architecture from Bessani et al., hence, it shares the same problems.

Kapritsos and Junqueira [13] propose another crash fault-tolerant replication system. They distribute multiple distinct clusters of agreement replicas over several physical machines, which all handle a particular part of the client requests. In contrast, our approach aims at

the better utilization of available processor cores to improve the efficiency and performance of agreement replicas without the need for additional hardware.

The consensus-oriented parallelization scheme as presented in this paper addresses the parallelization of agreement protocols within each replica. In its current form, the service implementation is still executed sequentially. However, proposals for how to parallelize the service execution already exist [9, 12, 14]. These systems are complementary to our concept and we will investigate combined approaches as future work.

7 Conclusion

In this paper, we advocated an *actor-based decomposition* of BFT replica implementations and presented a *consensus-oriented parallelization scheme* for BFT agreement protocols. While both concepts have the potential to simplify the notoriously complex realization of BFT systems, their main advantage is that they enable BFT installations to scale with the number of available processor cores. As our evaluation showed, they significantly increase the efficiency compared to traditional approaches when the demand is low and only few cores shall be used. If a high throughput is required, a consensus-oriented parallelization allows BFT deployments to exploit the capabilities of modern multi-core platforms where state-of-the-art systems quickly reach their limits imposed by not further parallelizable tasks. Additionally, we believe that the concepts proposed in this paper can also be beneficially applied to crash fault-tolerant systems, even though they typically do not require expensive message authentication and are generally less complex than BFT systems that tolerate arbitrary faults.

References

[1] AGHA, G. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, 1986.

[2] AMIR, Y., COAN, B., KIRSCH, J., AND LANE, J. Byzantine replication under attack. In *Proc. of the 2008 Int'l Conf. on Dependable Systems and Networks (DSN '08)* (2008), pp. 197–206.

[3] BESSANI, A., SOUSA, J., AND ALCHIERI, E. State machine replication for the masses with BFT-SMaRt. In *Proc. of the 2014 Int'l Conf. on Dependable Systems and Networks (DSN '14)* (2014).

[4] CASTRO, M., AND LISKOV, B. Practical Byzantine fault tolerance. In *Proc. of the 3rd Symp. on Operating Systems Design and Implementation (OSDI '99)* (1999), pp. 173–186.

[5] CLEMENT, A., KAPRITSOS, M., LEE, S., WANG, Y., ALVISI, L., DAHLIN, M., AND RICHE, T. UpRight cluster services. In *Proc. of the 22nd Symp. on Operating Systems Principles (SOSP '09)* (2009), pp. 277–290.

[6] CORBETT, J. C., DEAN, J., EPSTEIN, M., FIKES, A., FROST, C., FURMAN, J. J., GHEMAWAT, S., GUBAREV, A., HEISER, C., HOCHSCHILD, P., HSIEH, W., KANTHAK, S., KOGAN, E., LI, H., LLOYD, A., MELNIK, S., MWAURA, D., NAGLE, D., QUINLAN, S., RAO, R., ROLIG, L., SAITO, Y., SZYMANIAK, M., TAYLOR, C., WANG, R., AND WOODFORD, D. Spanner: Google's globally-distributed database. In *Proc. of the 10th Symp. on Operating Systems Design and Implementation (OSDI '12)* (2012), pp. 251–264.

[7] DAVID, T., GUERRAOUI, R., AND TRIGONAKIS, V. Everything you always wanted to know about synchronization but were afraid to ask. In *Proc. of the 24th Symp. on Operating Systems Principles (SOSP '13)* (2013), pp. 33–48.

[8] GUERRAOUI, R., KNEŽEVIĆ, N., QUÉMA, V., AND VUKOLIĆ, M. The next 700 BFT protocols. In *Proc. of the 5th European Conf. on Computer Systems (EuroSys '10)* (2010), pp. 363–376.

[9] GUO, Z., HONG, C., YANG, M., ZHOU, D., ZHOU, L., AND ZHUANG, L. Rex: Replication at the speed of multi-core. In *Proc. of the 9th European Conf. on Computer Systems (EuroSys '14)* (2014).

[10] HUNT, P., KONAR, M., JUNQUEIRA, F. P., AND REED, B. ZooKeeper: Wait-free coordination for Internet-scale systems. In *Proc. of the 2010 USENIX Annual Technical Conf. (ATC '10)* (2010), pp. 145–158.

[11] KAPITZA, R., BEHL, J., CACHIN, C., DISTLER, T., KUHNLE, S., MOHAMMADI, S. V., SCHRÖDER-PREIKSCHAT, W., AND STENGEL, K. CheapBFT: Resource-efficient Byzantine fault tolerance. In *Proc. of the 7th European Conf. on Computer Systems (EuroSys '12)* (2012), pp. 295–308.

[12] KAPITZA, R., SCHUNTER, M., CACHIN, C., STENGEL, K., AND DISTLER, T. Storyboard: Optimistic deterministic multithreading. In *Proc. of the 6th Workshop on Hot Topics in System Dependability (HotDep '10)* (2010), pp. 1–6.

[13] KAPRITSOS, M., AND JUNQUEIRA, F. P. Scalable agreement: Toward ordering as a service. In *Proc. of the 6th Workshop on Hot Topics in System Dependability (HotDep '10)* (2010).

[14] KAPRITSOS, M., WANG, Y., QUÉMA, V., CLEMENT, A., ALVISI, L., AND DAHLIN, M. All about Eve: Execute-verify replication for multi-core servers. In *Proc. of the 10th Symp. on Operating Systems Design and Implementation (OSDI '12)* (2012), pp. 237–250.

[15] KOTLA, R., ALVISI, L., DAHLIN, M., CLEMENT, A., AND WONG, E. Zyzzyva: Speculative Byzantine fault tolerance. In *Proc. of the 21st Symp. on Operating Systems Principles (SOSP '07)* (2007), pp. 45–58.

[16] SANTOS, N., AND SCHIPER, A. Achieving high-throughput state machine replication in multi-core systems. In *Proc. of the 33rd Int'l Conf. on Distributed Computing Systems (ICDCS '13)* (2013), pp. 266–275.

[17] SCHMIDT, D. C., AND SUDA, T. Transport system architecture services for high-performance communications systems. *IEEE Journal on Selected Areas in Communications 11*, 4 (1993), 489–506.

[18] SCHNEIDER, F. B. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys 22* (1990), 299–319.

[19] VERONESE, G. S., CORREIA, M., BESSANI, A. N., LUNG, L. C., AND VERÍSSIMO, P. Efficient Byzantine fault-tolerance. *IEEE Transactions on Computers 62*, 1 (2013), 16–30.

[20] WELSH, M., CULLER, D., AND BREWER, E. SEDA: An architecture for well-conditioned, scalable Internet services. In *Proc. of the 18th Symp. on Operating Systems Principles (SOSP '01)* (2001), pp. 230–243.

[21] YIN, J., MARTIN, J.-P., VENKATARAMANI, A., ALVISI, L., AND DAHLIN, M. Separating agreement from execution for Byzantine fault tolerant services. In *Proc. of the 19th Symp. on Operating Systems Principles (SOSP '03)* (2003), pp. 253–267.