TRIPOD: An Efficient, Highly-available Cluster Management System

Paper #10

Abstract

Driven by the increasing computational demands, cluster management systems (e.g., MESOS) are already pervasive for deploying many applications. Unfortunately, despite much effort, existing systems are still difficult to meet the high requirements of critical applications (e.g., trading, medical, and military applications), because these applications naturally require high-availability and low performance overhead in deployments. Existing systems typically replicate their job controllers so that these controllers are highly-available and thus they can handle applications failures. However, applications themselves are still often a single point of failure, leaving arbitrary unavailabile time windows for themselves.

This paper proposes the design of TRIPOD, a cluster management system that automatically provides high-availability to general applications. TRIPOD's key to make applications achieve high-availability efficiently is a fast PAXOS replication protocol that leverages RDMA (Remote Direct Memory Access). TRIPOD runs replicas of the same job with a replicas of controllers, and controllers agree on job requests efficiently with this protocol. Evaluation shows that TRIPOD has low performance overhead in both throughput and resonse time compared to an application's unreplicated executions.

1 Introduction

Driven the the drastically increasing computational demands and the volumes of data, cluster management systems [10, 12, 26, 27, 29, 47, 48, 53] incorporate more and more applications. Applications not only include typical applications (e.g., Hadoop [24]), but also *critical* applications such as trading, fraud detection, health care, and military applications. These applications naturally require high-availability and low performance overhead in deployments. For example, a high-frequency trading application tends to be highly-available during its operation hours, and adding merely hundreds of μs to its response time means big money lost [25].

Unfortunately, despite much effort, these systems are still difficult to meet the high requirements of critical applications because these systems do not provide high-availability to the applications. Specifically, existing systems typically replicate their job controller component which accepts jobs, allocate computational resources, and (re)schedule jobs on available resources. However, the applications themselves are not replicated: if an application crashes or a computational resource goes down (e.g., hardware errors), these systems have to reschedule the jobs, leaving an arbitrary unavailable time window for these applications.

State machine replication (SMR) [34] is a promising approach to address the availability problem of critical applications. SMR runs the same program on a number of replicas and uses a distributed consensus protocol (e.g., PAXOS [33, 34, 37, 40, 46]) to enforce the same inputs among replicas. An input consensus can achieve as long as a majority of replicas agree, thus SMR can tolerate various faults such as minor replica failures. For instance, two existing systems BORG [48] and MESOS [26] use PAXOS to replicate their controllers.

However, PAXOS's consensus is notoriously difficult to be fast. To agree on an input, traditional consensus protocols invoke at least one message round-trip between two replicas. Given that a `ping` in Ethernet takes hundreds of μs, a program running in an SMR system with three replicas must wait at least this time before processing an input.

Recently, Remote Direct Access Memory (RDMA) becomes commonplace in datacenters because of the decreasing prices of RDMA hardware. RDMA allows a process to directly write to the user space memory of a remote process, completely bypassing the remote OS kernel or CPU (the so called "one-sided" operations). An evaluation [39] shows that such a write round-trip takes only $\sim$3 μs in the Infiniband architecture [1]. Two recent RDMA-enabled PAXOS implementations [3, 43] reported $\sim$15 μs in consensus latency.

A naive approach to achieve high-availability for critical applications could be integrating PAXOS within every application. However, doing so has two issues. First, PAXOS is notoriously difficult to understand [41], implement [37], and verify [22].

Submitted to APSys '16, August 4-5, 2016, Hong Kong, China.

Second, running a PAXOS-enabled application in a cluster management system could be problematic because the system is unaware of the replication logic. For instance, if the system happens to schedule all the replicas of the same job on the same machine, then a single-point failure of this machine will turn down all replicas.

This paper proposes the design of TRIPOD, a cluster management system that automatically provides high-availability to general applications. TRIPOD makes replicas of controllers agree on job requests efficiently with RDMA-enabled PAXOS protocols [3, 43]. To avoid the two aforementioned issues, TRIPOD chooses to integrate PAXOS in the cluster system, not in applications. Doing so has two benefits. First, TRIPOD's PAXOS acts a single, general SMR service to applications: we can just leverage existing verification tools [22, 51] to make sure this PAXOS protocol is robust and correct, and then we can benefit many applications. Second, now TRIPOD's own scheduler can handle the replication logic and do careful, replication-aware scheduling for jobs.

In an implementation level, TRIPOD integrates a RDMA-enabled PAXOS protocol [3] with MESOS [26], a widely used cluster management system. This integration forms a mutual beneficial eco-system[1]: by replicating tasks with PAXOS, TRIPOD ensures that minor failures in a job does not affect its replicated jobs to compute results efficiently. By leveraging MESOS's resource allocation and isolation feature, TRIPOD can run diverse applications on the same physical machine, saving much of the physical hardware cost in traditional replication techniques.

In addition to fault-tolerance, TRIPOD also provides a tail-tolerance feature [17] in cluster computing. The tail of latency distribution problem (or straggler) is pervasive for decades: since a job request consists of tasks on diverse machines, the response of this job is often bounded by the slowest task. Although TRIPOD is not the first work that presents a replication approach to address the straggler problem [11], TRIPOD provides this feature in a simple and general way on top of its PAXOS replication architecture.

As part of TRIPOD's development, we have implemented an RDMA-enabled PAXOS protocol [3] in Linux. This protocol intercepts a general program's inbound socket calls (e.g., `recv()`) and agrees on the received inputs in these calls across replicas of the same program. To emulate a typical Twitter deployment, we ran this protocol with Redis, a popular key-value store that Twitter uses, and we ran Redis's own benchmark workload. Evaluation showed that, compared to Redis's unreplicated execution, this protocol incurred merely 4.28% % overhead on response time and 4.16% % overhead on throughput. This protocol was 40.1X faster than `ZooKeeper`'s SMR protocol [45] which runs on TCP/IP.

The remaining of this paper is organized as follows. §2 introduces background on PAXOS and RDMA. §3 gives an overview of our TRIPOD design. §4 describes TRIPOD's design on tail-tolerance. §5 presents evaluation results, §6 discusses related work, and §7 concludes.

2 Background

2.1 PAXOS

An SMR system runs the same program and its data on a set of machines (replicas), and it uses a distributed consensus protocol (typically, PAXOS [14, 33–35, 37, 46]) to coordinates inputs across replicas. For efficiency, in normal case, PAXOS often lets one replica work as the leader which invokes consensus requests, and the other replicas work as backups to agree on or reject these requests. If the leader fails, PAXOS elects a new leader from the backups.

Two main reliability features must be enforced in a standard PAXOS protocol. The first one is durability. When a new input comes, the PAXOS leader writes this input in local stable storage. The leader then starts a new consensus round, which invokes a consensus request on "processing this input" to the other backups. A backup also writes the received consensus request in local storage if it agrees on this request. Durability ensures that even if the leader or backups fail and restart, they can still retrieve the requests from local stable storage and re-execute them.

The second feature is safety. As long as a majority of replicas agree on this input (i.e., this input is *committed*), PAXOS guarantees that all replicas consistently agree to process this input. If a replica sees that an input consensus has been reached, this consensus must have really been reached by at least a majority of replicas. Safety ensures that if a consensus has not really been reached, no replica will "think" that a consensus has been reached.

Durability and safety make replicas consistently agree on each input and tolerate various faults, including machine failures and network errors. As consensus rounds move on, PAXOS consistently enforces the same sequence of inputs across replicas. It also enforces same execution states across replicas without divergence if a program behaves as a deterministic state machine (i.e., always produces the same output on the same input).

Network latency of consensus messages is one key problem to make general server programs adopt SMR. For instance, in an efficient, practical PAXOS implementation [37], each input in normal case takes one consensus round-trip between every two replicas (one request from the leader and one reply from a backup).

[1] We name our system after the ancient Chinese three-legged tripod, a reliable, multi-purpose container.

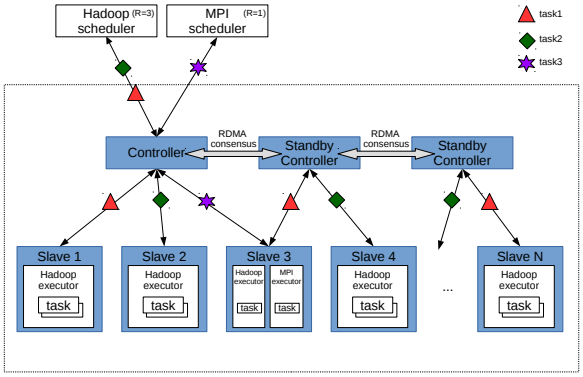

Figure 1: *The* TRIPOD *Architecture.* Key components are shaded (and in blue).

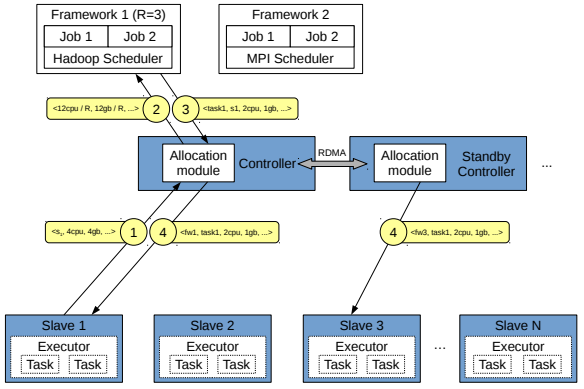

Figure 2: *The* TRIPOD *Workflow on Scheduling Jobs.*

2.2 RDMA

RDMA architecture such as Infiniband [1] or RoCE [5] recently becomes commonplace in datacenters due to its extreme low latency, high throughput, and its decreasing prices.

RDMA provides three types of communication primitives, from slowest to fastest: IPoIB (IP over Infiniband), message verbs, and one-sided read/write operations. A one-sided RDMA read/write operation can directly write from one replica's memory to a remote replica's memory, completely bypassing OS kernel and CPU of the remote replica. For brevity, the rest of this paper denotes a one-sided RDMA write operation as a "WRITE".

3 TRIPOD Overview

This section first introduces TRIPOD's architecture design (§3.1), its workflow on scheduling jobs with replication (§3.2), and its trade-off on reliability versus resource consumption (§3.3).

3.1 Architecture

TRIPOD's deployment model is similar to a typical cluster management system's (e.g., [26, 48]). TRIPOD has a replicas of three to five controllers, with each connects with the others using high-speed RDMA network. A controller is elected by PAXOS a the leading controller, which propose consensus requests to execute jobs. The other controllers are standby controllers which agree on or reject consensus requests. A number of slave machines act as computational resources that hold applications' running jobs. Controllers and slaves connect with either RDMA or TCP/IP. Application schedulers submit jobs to the leader controller.

Figure 1 depicts TRIPOD's architecture, and its key components are shaded (and in blue). To illustrate how TRIPOD works in an application perspective, this figure shows two applications, Hadoop and MPI. Each application has a *replica strength* (`R`) to denote the level of fault-tolerance it demands. This value is either 1 or equals the number of replicas of controllers in TRIPOD. By default, each application has `R=1`, which means that this application does not need replication. For such a default setting, TRIPOD runs the job as is without replication, like a typical cluster management system (e.g., Mesos).

In this figure, Hadoop's `R` is 3, which means that it wants to replica each of its job in three replicas for high-availability. Suppose Hadoop submits two jobs to the leader controller, each has different shapes (triangle or hexagon). The leader controller thens invokes a consensus on each job across controllers. Once a consensus is reached, each controller assigns the same job on different slave machines.

The leader controller directly returns its computation result to the Hadoop scheduler unless a tail-tolerance mechanism is triggered (§3.2). Standby controllers ignore the results unless the same mechanism is triggered.

3.2 Workflow on Scheduling Jobs

Figure 2 shows TRIPOD's workflow on scheduling jobs with four steps. This workflow is similar to that in Mesos except the second and fourth steps. These two steps TRIPOD abstract away the replication logic in its resource offers and allocations from the application. An application runs as if TRIPOD does not replicate any of its jobs, and TRIPOD handles all the replication logic.

In the first step, slave machines periodically report their available computing resources (e.g., CPU cores and memory) to the leader controller. In the second step, instead of reporting available resources, TRIPOD divides the amount of resources by each application's `R` value and then reports to the application. This is to reserve enough resources for TRIPOD to replicate the same job with `R` copies.

In the third step, an application scheduler submits jobs to the leader controller. The leader controller then invokes a consensus on this job by carrying the resource offer made to the application.

Once a majority of controllers agrees on executing this job, each controller does the fourth step. It schedules this

job on an available slave machine according to the resource offer. To avoid controllers putting the same job on the same slave machine, each controller maintains a disjoint bit-mask of slave machines; it only schedules this job on machines belongs to its bit-mask if this machine has available resource to hold this job.

3.3 Discussion

The main goal of TRIPOD is to provide high-availability to critical applications running in clusters. On one hand, clusters are included more and more computing resources (e.g., machines), where failures of these resources and applications are already commonplace [4]. On the other hand, high requirements on response times and availability become more and more important to critical applications.

For instance, Both NYSE and Nasdaq have experienced outage of their whole site [7] or specific IPO events [2] due to minor machine errors. Even social-networking applications like Facebook has strong fault-tolerance requirements, because minor machine failures have turned down the whole Facebook site for several times in the last few years [4], costing huge money lost.

Therefore, TRIPOD's design favors more on availability and performance (low consensus latency). It's design tends to use `R` times of resources compared to traditional applications. We argue that this extra resource utilizations are acceptable for critical applications, because if they have high demand on availability and response times, they can often tolerate costs on computing resources (e.g., trading and medical platforms).

4 Tail-tolerance Design

TRIPOD's tail tolerance design is based on its PAXOS consensus protocol. This design consists of three steps. First, for each job request, both the leader controller and standby controllers schedule the same job and receive output of this job. When each standby controller receives a job output, it uses RDMA WRITEs to write back their job outputs to the leader master's local memory.

Second, the leader master also waits for its own job output and frequently polls from the other replicas' outputs for this job. Third, if the leader finds that standby masters have already written back an output for a job, it sends back the output to the application scheduler and ignores its own job output. This guarantees that application schedulers receive job outputs without being affected by stragglers in minor replicated jobs.

Although TRIPOD is not the first work that presents a replication approach to address the straggler problem (e.g., Dolly [11]), TRIPOD's tail-tolerance design has two practical benefits. First, TRIPOD's design is simple and general to applications, because it is built on top of its PAXOS replication architecture without extra replication logic. Second, the standby masters' output write-back are fast through RDMA writes.

5 Evaluation

As part of TRIPOD's development, we have implemented a fast RDMA-enabled PAXOS protocol [3] for general applications. We evaluated this protocol with Redis [44], a popular key-value store. We chose Redis because it complies with a social-networking platform setting, where high-availability is important (§3.3).

For instance, Twitter uses MESOS as its scheduler and runs Redis as its core key-value store. Key-value stores are also widely used in critical, financial applications [25, 42]. In our evaluation, TRIPOD transparently supported Redis without the need of modifying Redis's code.

Our evaluation used three Dell R430 servers as SMR replicas. Each server has Linux 3.16.0, 2.6 GHz Intel Xeon CPU with 24 hyper-threading cores, 64GB memory, and 1TB SSD. Each machine has a Mellanox ConnectX-3 Pro Dual Port 40 Gbps NIC. These NICs are connected using the Infiniband RDMA architecture.

To mitigate network latency of public network, benchmarks were ran in a Dell R320 server (the application scheduler machine), with Linux 3.16.0, 2.2GHz Intel Xeon 12 hyper-threading cores, 32GB memory, and 160GB SSD. This server connects with the server machines with 1Gbps bandwidth LAN. The average `ping` latency between the client machine and a server machine is 301 μs. A larger network latency (e.g., sending job requests from WAN) will further mask TRIPOD's overhead.

To perform a stress testing on TRIPOD's input consensus protocol, we chose Redis's own benchmark to spawn a workload with 50% SET and 50% GET operations. We chose such a significant portions of writes because SET operation contains more bytes than a GET.

We spawned up to 32 concurrent connections, and then we measured both response time and throughput. We also measured TRIPOD's bare consensus latency. All evaluation results were done with a replica group size of three. Each performance data point in the evaluation is taken from the mean value of 10 repeated executions.

The rest of this section focuses on these questions:

§5.1: What is TRIPOD's performance compared to the unreplicated executions? What is the consensus latency of TRIPOD's PAXOS protocol?

§5.2: What is TRIPOD's performance compared to existing SMR systems?

5.1 Performance Overhead

Figure 3 shows TRIPOD's throughput and Figure 4 response time. We varied the number of concurrent client connections for each server program by from one to 32

threads. Overall, compared to Redis's unreplicated executions, TRIPOD merely incurred a mean throughput overhead of 4.16%.

As the number of threads increases, Redis's unreplicated executions got a performance improvement. TRIPOD scaled almost as well as the unreplicated executions.

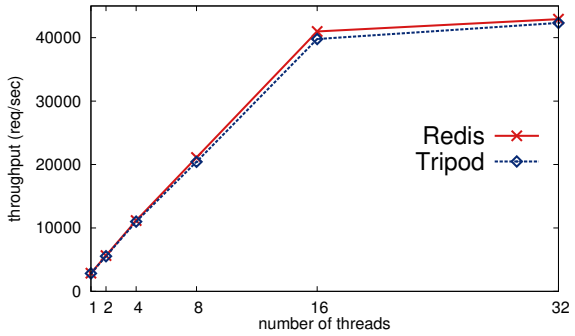

Figure 3: TRIPOD *throughput compared to the unreplicated execution.*

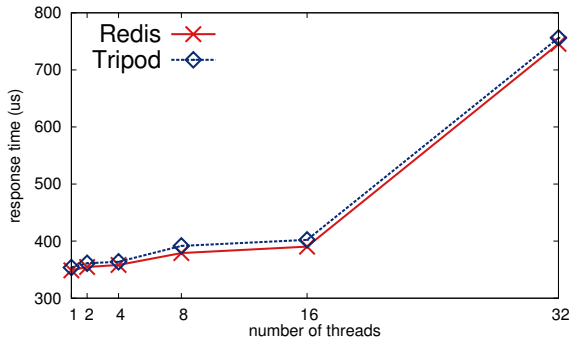

Figure 4: TRIPOD *response time compared to the unreplicated execution.*

To deeply understand TRIPOD's performance overhead, we collected the number of socket call events and consensus durations on the leader controller side. Given 10K requests and 8 concurrent connections, Redis spawned 10,016 socket calls, each of which invoked a PAXOS consensus. The average received data bytes in these socket calls were 42.0 (e.g., inputs from `recv()`). The average consensus latency for these calls were 9.9 μs because consensus were done by RDMA WRITEs across the leader and standby masters.

These results suggest that TRIPOD has the potential to provide a fast PAXOS replication service to general applications with low overhead.

5.2 Comparison with Traditional SMR Systems

We compared TRIPOD with the CALVIN database system [45] because CALVIN's input consensus uses `ZooKeeper` [9], one of the most widely used coordination service built on TCP/IP. To conduct a fair comparison, we ran CALVIN's own transactional database server in TRIPOD as the server program, and we compared throughputs and the consensus latency with CALVIN's consensus protocol `ZooKeeper`.

Performance metric	`ZooKeeper`	TRIPOD
Throughput (requests/s)	19,925	17,614
Consensus latency (μs)	511.9	12.5

Table 1: *Comparison with* CALVIN*'s* `ZooKeeper` *replication.*

As shown in Table 1, CALVIN's `ZooKeeper` replication achieved 19.9K transactions/s with a 511.9 μs consensus latency. TRIPOD achieved 17.6K transactions/s with a 12.5 μs consensus latency. The throughput in CALVIN was 13.1% higher than that in TRIPOD because CALVIN puts transactions in a batch with a 10 ms timeout, it then invokes `ZooKeeper` for consensus on this batch. The average number of bytes in CALVIN's batches is 18.8KB, and the average number of input bytes in each TRIPOD consensus (one for each `recv()` call) is 93 bytes. Batching helps CALVIN achieve good throughput. TRIPOD currently has not incorporated a batching technique because its latency is already reasonable (§5.1).

Notably, TRIPOD's consensus latency was 40.1X faster than `ZooKeeper`'s mainly due to TRIPOD's RDMA-accelerated consensus protocol, although we ran CALVIN's `ZooKeeper` consensus on IPoIB. A prior SMR evaluation [43] also reports a similar 320 μs consensus latency in `ZooKeeper`. Two other recent SMR systems Crane [16] and Rex [23] may incur similar consensus latency as `ZooKeeper`'s because all their consensus protocols are based on TCP/IP.

This TRIPOD-CALVIN comparison suggests TRIPOD's PAXOS protocol may be leveraged to speedup the SMR services in existing cluster management systems (e.g., MESOS currently uses `ZooKeeper`).

6 Related Work

Cluster Management System. Cluster management systems [10, 12, 26, 27, 29, 47, 48, 53] are widespread because they can transparently support many diverse applications (e.g., Hadoop [24], Dryad [28], and key-value stores [44]). These existing systems mainly focus on high availability for themselves by replicating important components (e.g., controllers) within these systems, or focus on fault-recovery of applications [53]. To the best of our knowledge, no existing system provides efficient and general high-availability service to applications. Although TRIPOD's current design leverages an existing system Mesos [26], its general PAXOS protocol design can also be integrated in other systems.

State machine replication (SMR). SMR is a powerful, but complex fault-tolerance technique. The literature has developed a rich set of PAXOS algorithms [33, 34, 37, 40, 46] and implementations [13, 14, 37]. PAXOS

is notoriously difficult to be fast and scalable [38]. To improve speed and scalability, various advanced replication models have been developed [21, 32, 36, 40]. Since consensus protocols play a core role in datacenters [8, 26, 52] and distributed systems [15, 36], a variety of study have been conducted to improve different aspects of consensus protocols, including performance [35, 40, 43], understandability [34, 41], and verifiable reliability rules [22, 51]. Although TRIPOD tightly integrates RDMA features in PAXOS, its implementation mostly complies with a popular, practical approach [37] for reliability. Other PAXOS approaches can also be leveraged in TRIPOD.

RDMA techniques. RDMA techniques have been implemented in various architectures, including Infiniband [1], RoCE [5], and iWRAP [6]. RDMA have been leveraged in many systems to improve application-specific latency and throughput, including high performance computing [20], key-value stores [18, 30, 31, 39], transactional processing systems [19, 49], and file systems [50]. These systems are largely complementary to TRIPOD.

7 Conclusion

We have presented the design of TRIPOD, the first cluster management system design that automatically provides high-availability to general applications. TRIPOD leverages an RDMA-enabled PAXOS implementation to efficiently agree on task requests for critical applications across its controllers, and it replicates the same tasks to make these tasks highly available. TRIPOD also presents a design on achieving tail-tolerance with its fault-tolerance architecture. Initial evaluation shows that TRIPOD's protocol has reasonable overhead, which may be suitable for critical application with short response time requirements.

References

[1] An Introduction to the InfiniBand Architecture. http://buyya.com/superstorage/chap42.pdf.

[2] Facebook ipo: What went wrong? http://money.cnn.com/2012/05/23/technology/facebook-ipo-what-went-wrong/.

[3] A fast, general rdma-enabled paxos protocol implementation. https://github.com/apsys16-p10/rdma-paxos.

[4] Is facebook down? a history of outages. https://www.theguardian.com/technology/2015/jan/27/is-facebook-down-outages.

[5] Mellanox Products: RDMA over Converged Ethernet (RoCE). http://www.mellanox.com/page/products_dyn?product_family=79.

[6] RDMA iWARP. http://www.chelsio.com/nic/rdma-iwarp/.

[7] This is why the nyse shut down today. http://fortune.com/2015/07/08/nyse-halt/.

[8] Why the data center needs an operating system. cloud.berkeley.edu/data/dcos.pptx.

[9] ZooKeeper. https://zookeeper.apache.org/.

[10] Tupperware. https://www.youtube.com/watch?v=C_WuUgTqgOc, 2014.

[11] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Effective straggler mitigation: Attack of the clones. In *NSDI'13*, 2013.

[12] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 285–300, Berkeley, CA, USA, 2014. USENIX Association.

[13] M. Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the Seventh Symposium on Operating Systems Design and Implementation (OSDI '06)*, pages 335–350, 2006.

[14] T. D. Chandra, R. Griesemer, and J. Redstone. Paxos made live: An engineering perspective. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing (PODC '07)*, Aug. 2007.

[15] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's globally-distributed database. Oct. 2012.

[16] H. Cui, R. Gu, C. Liu, and J. Yang. Paxos made transparent. In *Proceedings of the 25th ACM Symposium on Operating Systems Principles (SOSP '15)*, Oct. 2015.

[17] J. Dean and L. A. Barroso. The tail at scale. *Commun. ACM*, 56(2):74–80, Feb. 2013.

[18] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro. Farm: Fast remote memory. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, 2014.

[19] A. Dragojević, D. Narayanan, E. B. Nightingale, M. Renzelmann, A. Shamis, A. Badam, and M. Castro. No compromises: Distributed transactions with consistency, availability, and performance. In *Proceedings of the 25th ACM Symposium on Operating Systems Principles (SOSP '15)*, Oct. 2015.

[20] M. P. I. Forum. Open mpi: Open source high performance computing, Sept. 2009.

[21] L. Glendenning, I. Beschastnikh, A. Krishnamurthy, and T. Anderson. Scalable consistency in scatter. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP '11)*, Oct. 2011.

[22] H. Guo, M. Wu, L. Zhou, G. Hu, J. Yang, and L. Zhang. Practical software model checking via dynamic interface reduction. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP '11)*, pages 265–278, Oct. 2011.

[23] Z. Guo, C. Hong, M. Yang, D. Zhou, L. Zhou, and L. Zhuang. Rex: Replication at the speed of multicore. In *Proceedings of the 2014 ACM European Conference on Computer Systems (EUROSYS '14)*, page 11. ACM, 2014.

[24] Hadoop. `http://hadoop.apache.org/core/`.

[25] R. Hecht and S. Jablonski. Nosql evaluation: A use case oriented survey. *2012 International Conference on Cloud and Service Computing*, 0:336–341, 2011.

[26] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX conference on Networked Systems Design and Implementation*, NSDI'11, Berkeley, CA, USA, 2011. USENIX Association.

[27] M. Isard. Autopilot: Automatic data center management. *SIGOPS Oper. Syst. Rev.*, 41(2):60–67, Apr. 2007.

[28] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys '07: Proceedings of the Second ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 59–72, 2007.

[29] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 261–276, New York, NY, USA, 2009. ACM.

[30] J. Jose, H. Subramoni, K. Kandalla, M. Wasi-ur Rahman, H. Wang, S. Narravula, and D. K. Panda. Scalable memcached design for infiniband clusters using hybrid transports. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12, 2012.

[31] A. Kalia, M. Kaminsky, and D. G. Andersen. Using rdma efficiently for key-value services. Aug. 2014.

[32] M. Kapritsos and F. P. Junqueira. Scalable agreement: Toward ordering as a service. In *Proceedings of the Sixth International Conference on Hot Topics in System Dependability*, HotDep'10, 2010.

[33] L. Lamport. Paxos made simple. `http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf`.

[34] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[35] L. Lamport. Fast paxos. Fast Paxos, Aug. 2006.

[36] Y. Mao, F. P. Junqueira, and K. Marzullo. Mencius: building efficient replicated state machines for wans. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, volume 8, pages 369–384, 2008.

[37] D. Mazieres. Paxos made practical. Technical report, Technical report, 2007. http://www. scs. stanford. edu/dm/home/papers, 2007.

[38] E. Michael. *Scaling Leader-Based Protocols for State Machine Replication*. PhD thesis, University of Texas at Austin, 2015.

[39] C. Mitchell, Y. Geng, and J. Li. Using one-sided rdma reads to build a fast, cpu-efficient key-value store. In *Proceedings of the USENIX Annual Technical Conference (USENIX '14)*, June 2013.

[40] I. Moraru, D. G. Andersen, and M. Kaminsky. There is more consensus in egalitarian parliaments. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles (SOSP '91)*, Nov. 2013.

[41] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the USENIX Annual Technical Conference (USENIX '14)*, June 2014.

[42] B. K. Park, W.-W. Jung, and J. Jang. Integrated financial trading system based on distributed in-memory database. In *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems*, RACS '14, pages 86–87, New York, NY, USA, 2014. ACM.

[43] M. Poke and T. Hoefler. Dare: High-performance state machine replication on rdma networks. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, 2015.

[44] `http://redis.io/`.

[45] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi. Fast distributed transactions and strongly consistent replication for oltp database systems. May 2014.

[46] R. Van Renesse and D. Altinbuken. Paxos made moderately complex. *ACM Computing Surveys (CSUR)*, 47(3):42:1–42:36, 2015.

[47] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, pages 5:1–5:16, New York, NY, USA, 2013. ACM.

[48] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, pages 18:1–18:17, New York, NY, USA, 2015. ACM.

[49] X. Wei, J. Shi, Y. Chen, R. Chen, and H. Chen. Fast in-memory transaction processing using rdma and htm. In *Proceedings of the 25th ACM Symposium on Operating Systems Principles (SOSP '15)*, SOSP '15, Oct. 2015.

[50] G. G. Wittawat Tantisiriroj. Network file system (nfs) in high performance networks. Technical Report CMU-PDLSVD08-02, Carnegie Mellon University, Jan. 2008.

[51] J. Yang, T. Chen, M. Wu, Z. Xu, X. Liu, H. Lin, M. Yang, F. Long, L. Zhang, and L. Zhou. MODIST: Transparent model checking of unmodified distributed systems. In *Proceedings of the Sixth Symposium on Networked Systems Design and Implementation (NSDI '09)*, pages 213–228, Apr. 2009.

[52] M. Zaharia, B. Hindman, A. Konwinski, A. Ghodsi, A. D. Joesph, R. Katz, S. Shenker, and I. Stoica. The datacenter needs an operating system. In *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing*, 2011.

[53] Z. Zhang, C. Li, Y. Tao, R. Yang, H. Tang, and J. Xu. Fuxi: A fault-tolerant resource management and job scheduling system at internet scale. *Proc. VLDB Endow.*, 7(13):1393–1404, Aug. 2014.