TRIPOD: An Efficient and Transparent Framework for Dynamic Program Analysis

Paper #10

Abstract

Driven by the increasing computational demands, cluster management systems (e.g., MESOS) are already pervasive and important. Unforunately, despite much effort, these systems are still difficult to support critical applications (e.g., trading, medical, and military services) because these applications naturally demand high-availability and short latency. Existing systems typically replicate their task controllers in order to tolerate single-point failures of this controller and to reschedule applications from their failures. However, the applications themselves are still often a signle point of failure, which inevitably hurt availability.

This paper proposes TRIPOD, a cluster management system design that automatically provides high-availability to general applications. TRIPOD's key to achieve fast fault-tolerance is a fast PAXOS replication protocol that leverages RDMA (Remote Direct Memory Access). TRIPOD runs replicas of the same application with a set of controllers, where controllers agree on computational requests efficiently with this protocol. Initial evaluation results show that TRIPOD's protocol has low latency overhead compared to the applications' unreplicated executions.

1 Introduction

Driven the the drastically increasing computational demands and the volumns of data, cluster managemement systems (e.g., Mesos [23]) are incorporating more and more applications (e.g., Spark [48] and Redis [41]) in order to harness the computational resources in clusters for these applications. Applications not only include typical batch frameworks (e.g., Hadoop [21]), but also critical applications such as trading platforms, fraud detection systems, health care systems, and military systems. We consider an application *critical* if its high requirements on availability and response time. For example, a high-frequency trading platform tends to be highly-availabile during the stock operation hours, and adding a few hundred μs to the platform's response time means huge money lost.

Unfortunately, despite recent advances in building and applying cluster managemement systems [9, 11, 23, 24, 26, 43, 44, 50], 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, to make the systems themselves highly-availabile, existing systems typically replicate their controller component which accepts tasks, allocate computational resources, and (re)schedule tasks on availabile 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 tasks, leaving an arbitral unavailable time window for these applications. A possible key reason of this problem is that these systems are initially designed for big-data engines which already considered fault recovery (but, not availability).

State machine replication (SMR) [31] may be 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 [30, 31, 34, 37, 42]) 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 [44] and MESOS [23] 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.

Remote Direct Access Memory (RDMA) is a promising technique to mitigate consensus latency because recently it becomes cheaper and increasingly pervasive in datacenters. 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 [36] shows that such a write round-trip takes only $\sim$3 μs

APSys '15, July 27-28, 2015, Tokyo, Japan.
2015 ACM. ISBN 978-1-4503-3554-6/15/07$15.00.
http://dx.doi.org/10.1145/2797022.2797033

in the Infiniband architecture [1]. Two recent RDMA-enabled PAXOS implementations [3, 40] reported ∼15 μs consensus latency.

A straightforward approach to achieve high-availability for critical applications could be integrating PAXOS within each application. However, doing so faces two issues. First, PAXOS is notoriously difficult to understand [38], implement [34], and verify [20]. Second, running a PAXOS-enabled application in a cluster managemement system could be problematic because the system is unaware of the. For instance, if the system schedules all the replicas of the same task on the same machine, then a single-point failure on this machine will turn down all replicas.

This paper proposes TRIPOD, a cluster management system design that automatically provides high-availability to general applications. To avoid the two aforementioned issues, TRIPOD's design chooses to integrate PAXOS in the cluster system, not the application. Doing so has two benefits. First, TRIPOD's PAXOS acts a single, general SMR service to general applications: we can just leverage existing reliability tools [20, 47] to make sure this PAXOS protocol is correct and then we can benefit many other applications. Second, now TRIPOD's own scheduler can handle the replication logic and schedule replicas of the same task on diverse physical machines.

TRIPOD achieves fast fault-tolerance with RDMA-enabled PAXOS protocols [3, 40]. TRIPOD supports both normal applications (i.e., non-critical) and critical ones. In TRIPOD, a replicas of controllers run a critical application's same task on diverse physical machines, and controllers use a fast PAXOS protocol to agree on computational requests.

In an implementation level, TRIPOD proposes to integrate a RDMA-enabled PAXOS protocol with MESOS [23], a widely used cluster managemement system. This integration forms a mutual beneficial ecosystem[1]: by replicating tasks with PAXOS, TRIPOD ensures that minor failures in a task does not affect the other tasks to compute reply efficiently. By leveraging MESOS's resource allocation and isolation feature, TRIPOD can run diverse applications on the same physical machine, saving much of the phyiscal hardware cost in typical PAXOS.

In addition to fault-tolerance, TRIPOD's design can also mitigate the tail-tolerance problem (or straggler) [15] in cluster computing. The tail of latency distribution problem 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 [10], TRIPOD's tail-tolerance design is built on top of its PAXOS replication achitecture for simplicy and generality.

As part of TRIPOD's development, we 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 this program. To simulate 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 X.X% overhead on response time and Y.Y% overhead on throughput. This initial evaluation result suggests that TRIPOD could achieve reasonable performance overhead on critical applications.

The remaining of this paper is organized as follows. §2 introduces background on PAXOS and RDMA. §3 gives an overview of our TRIPOD system. §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 [13, 30–32, 34, 42]) 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 quorum (typically, 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 this consensus

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

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 [34], 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).

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 tasks with replication (§3.2), and its trade-off on reliability vesus resource consumption (§3.3).

3.1 Architecture

TRIPOD's deployment model is similar to a typical cluster management system's (e.g., [23, 44]). 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 tasks. 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 tasks. Controllers and slaves connect with either RDMA or TCP/IP. Application schedulers submit tasks 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 strengh* (R) to denote the level of fault-tolerance it demands. This value is either 1 or equals the

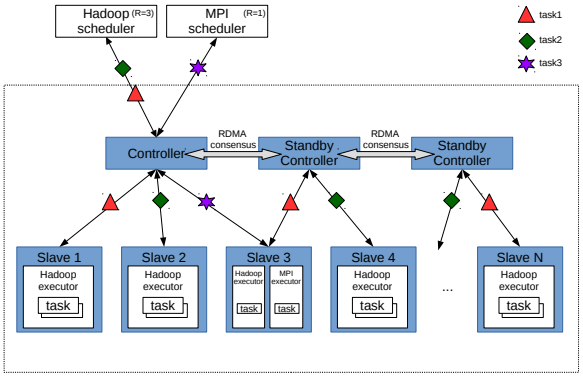

Figure 1: *The* TRIPOD *Architecture.* Key components are shaded (and in blue).

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 task 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 task in three replicas for high-availability. Suppose Hadoop submits two tasks to the leader controller, each has different shapde (triangle or hexagon). The leader controller thens invokes a consensus on each task across controllers. Once a consensus is reached, each controller assigns the same task on different slave machines.

The leader controller returns its task computation result to the Hadoop scheduler unless a tail-tolerance mechanism is triggered (§3.2). standby controllers ignore the results of their tasks unless this mechanism is triggered.

3.2 Workflow on Scheduling Tasks

Figure 2 shows TRIPOD's workflow on scheduling tasks 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 tasks, and TRIPOD handles all the replication logica.

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 task with `R` copies.

In the third step, an application scheduler submits

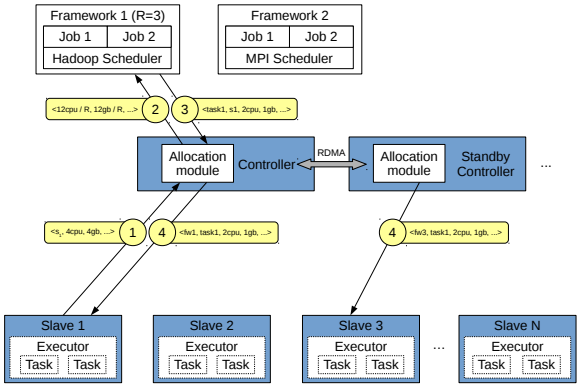

Figure 2: *The* TRIPOD *Workflow on Scheduling Tasks.*

tasks to the leader controller. The leader controller then invokes a consensus on this task by carring the resource offer maded to the application.

Once a majority of controllers agrees on executing this task, each controller does the fourth task. It schedules this task on an available slave machine according to the resource offer. To avoid controllers putting the same task on the same slave machine, each controller maintains a disjoint bitmask of slave machines; it only schedules this task on machines belongs to its bitmask if this machine has available resource to hold this task.

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 [10]), 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 achitecture 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 [41], a popular key-value store. We chose Redis because it complies with a social-networking platform setting. 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 [22, 39].

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.

In our evaluation, we did not need to modify Redis's code and it ran with TRIPOD's replication protocol. 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%.

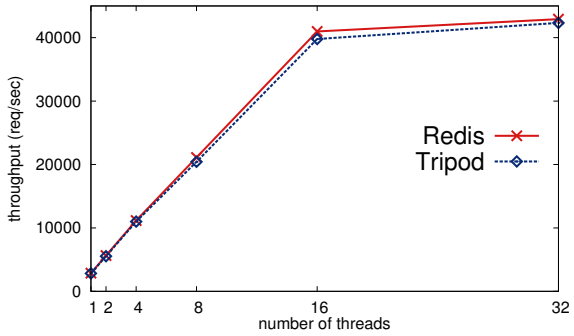

Figure 3: TRIPOD *throughput compared to the unreplicated execution.*

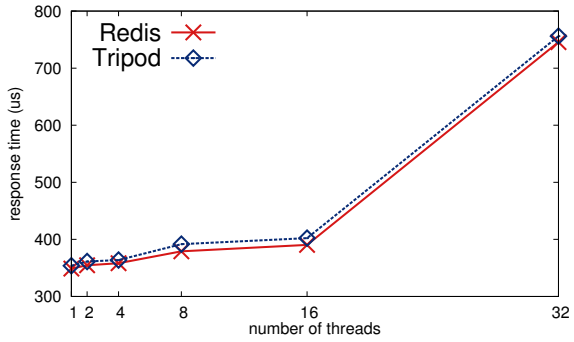

Figure 4: TRIPOD *response time compared to the unreplicated execution.*

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

6 Related Work

Cluster Management System. Cluster management systems [9, 11, 23, 24, 26, 43, 44, 50] are widespread because they can transparently support many diverse applications (e.g., Hadoop [21], Dryad [25], and key-value stores [41]). 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 [50]. 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 [23], 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 developeed a rich set of PAXOS algorithms [30, 31, 34, 37, 42] and implementations [12, 13, 34]. PAXOS is notoriously difficult to be fast and scalable [35]. To improve speed and scalability, various advanced replication models have been developed [19, 29, 33, 37]. Since consensus protocols play a core role in datacenters [8, 23, 49] and distributed systems [14, 33], a variety of study have been conducted to improve different aspects of consensus protocols, including performance [32, 37, 40], understandability [31, 38], and verifiable reliability rules [20, 47]. Although TRIPOD tightly integrates RDMA features in PAXOS, its implementation mostly complies with a popular, practical approach [34] 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 [18], key-value stores [16, 27, 28, 36], transactional processing systems [17, 45], and file systems [46]. These systems are largely complementary to TRIPOD.

7 Conclusion

We have presentd 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 availabile. TRIPOD also presents a design on achieving tail-tolerance with its fault-tolerance architecture. Initial evaluation shows that TRIPOD's procol has reasonable overhead, which may be suitable for critical application with short response time requirements.

*Bibliography

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

[2] Facebook ipo delay. http://www.reuters.com/article/us-nasdaq-omx-facebook-litigation-idUSKBN0HL1ED20140423.

[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. http://radar.oreilly.com/2014/12/why-the-data-center-needs-an-operating-system.html.

[9] Tupperware. http://www.slideshare.net/Docker/aravindnarayanan-facebook140613153626phpapp02-37588997, 2014.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

[21] Hadoop. http://hadoop.apache.org/core/.

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

[23] 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.

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

[25] 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.

[26] 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.

[27] 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.

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

[29] 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.

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

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

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

[33] 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.

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

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

[36] 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.

[37] 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.

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

[39] 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.

[40] 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.

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

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

[43] 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.

[44] 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.

[45] 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.

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

[47] 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.

[48] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[49] 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.

[50] 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.