Distributed Consensus and Implications of NVM on Database Management Systems

EXPERT-CURATED GUIDES TO THE BEST OF CS RESEARCH

Research for Practice combines the resources of the ACM Digital Library, the largest collection of computer science research in the world, with the expertise of the ACM membership. In every RfP column two experts share a short curated selection of papers on a concentrated, practically oriented topic.

I am thrilled to introduce our second installment of Research for Practice, which provides highlights from two critical areas in storage and large-scale services: distributed consensus and nonvolatile memory.

First, how do large-scale distributed systems mediate access to shared resources, coordinate updates to mutable state, and reliably make decisions in the presence of failures? Camille Fournier, a seasoned and experienced distributed-systems builder (and ZooKeeper PMC), has curated a fantastic selection on distributed consensus in practice. The history of consensus echoes many of the goals of RfP: for decades the study and use of consensus protocols were considered notoriously difficult to understand and remained primarily academic concerns. As a result, these protocols were largely ignored by industry. The rise of Internet-scale services and demands for automated solutions to cluster management, failover, and sharding in the 2000s finally led to the widespread practical adoption of these techniques. Adoption proved difficult, however, and the process in turn led to new (and ongoing) research on the subject. The papers in this selection highlight the challenges and the rewards of making the theory of consensus practical—both in theory and in practice.

Second, while consensus concerns *distributed* shared state, our second selection concerns the impact of hardware trends on *single-node* shared state. Joy Arulaj and Andy Pavlo provide a whirlwind tour of the implications of NVM (nonvolatile memory) technologies on modern storage systems. NVM promises to overhaul the traditional paradigm that stable storage (i.e., storage that persists despite failures) be block-addressable (i.e., requires reading and writing in large chunks). In addition, NVM's performance characteristics lead to entirely different design trade-offs than conventional storage media such as spinning disks.

As a result, there is an arms race to rethink software storage-systems architectures to accommodate these new characteristics. This selection highlights projected implications for recovery subsystems, data-structure design, and data layout. While the first NVM devices have yet to make it to market, these pragmatically oriented citations from the literature hint at the volatile effects of nonvolatile media on future storage systems.

I believe these two excellent contributions fulfill RfP's goal of allowing you, the reader, to become an expert in a weekend afternoon's worth of reading. To facilitate this process, as always, we have provided open access to the ACM Digital Library for the relevant citations from these selections so you can enjoy these research results in their full glory. Keep on the lookout for our next installment, and please enjoy! *—Peter Bailis*

"A distributed system is one in which the failure of a computer you didn't know existed can render your own computer unusable."
—Leslie Lamport

DISTRIBUTED CONSENSUS

BY CAMILLE FOURNIER

As Lamport predicted in this quote, the real challenges of distributed computing—not just communicating via a network, but communicating to unknown nodes in a network—have greatly intensified in the past 15 years. With the incredible scaling of modern systems, "we have found ourselves in a world where answering the question, what is running where?" is increasingly difficult. Yet, we continue to have requirements that certain data never be lost and that certain actions behave in a consistent and predictable fashion, even when some nodes of the system may fail. To that end, there has been a rapid adoption of systems that rely on consensus protocols to guarantee this consistency in a widely distributed world.

The three papers included in this selection address the real world of consensus systems: Why are they needed? Why are they difficult to understand? What happens when you try to implement them? Is there an easier way, something that more developers can understand and therefore implement?

The first two papers discuss the reality of implementing Paxos-based consensus systems at Google, focusing first on the challenges of correctly implementing Paxos itself, and second on the challenges of creating a system based on a consensus algorithm that provides useful functionality for developers. The final paper attempts to answer the question, is there an easier way? by introducing Raft, a consensus algorithm designed to be easier for developers to understand.

Theory Meets Reality

Chandra, T. D., Griesemer, R., Redstone, J. et al. 2007. Paxos made live—an engineering perspective. Proceedings of the 26th Annual ACM Symposium on Principles of Distributed Computing: 398-407.
http://queue.acm.org/rfp/vol14iss3.html

Paxos as originally stated is a page of pseudocode. The complete implementation of Paxos inside of Google's Chubby lock service is several thousand lines of C++. What happened? "Paxos Made Live" documents the evolution of the Paxos algorithm from theory into practice.

The basic idea of Paxos is to use voting by replicas with consistent storage to ensure that, even in the presence of failures, there can be a unilateral consensus. This requires a coordinator be chosen, proposals sent and voted upon, and finally a commit recorded. Generally, systems record a series of these consensus values to a sequence log. This log-based variant is called multi-Paxos, which is less formally specified.

In creating a real system, durable logs are written to disks, which have finite capacity and are prone to corruption that must be detected and taken into account. The algorithm must be run on machines that can fail, and to make it operable at scale you need to be able to change group membership dynamically. While the system was expected to be fault-tolerant, it also needed to perform quickly enough to be useful; otherwise, developers would work around it and create incorrect abstractions. The team details their efforts to make sure the core algorithm is expressed correctly and is testable, but even with these conscious efforts, the need for performance optimizations, concurrency, and multiple developers working on the project still means that the final system is ultimately an extended version of Paxos, which is difficult to prove correct.

Hell is Other Programmers

Burrows, M. 2006. The Chubby lock service for loosely coupled distributed systems. Proceedings of the Seventh Symposium on Operating Systems Design and Implementation: 335-350.
http://queue.acm.org/rfp/vol14iss3.html

While "Paxos Made Live" discusses the implementation of the consensus algorithm in detail, this paper about the Chubby lock service examines the overall system built around this algorithm. As research papers go, this one is a true delight for the practitioner. In particular, it describes designing a system and then evolving that design after it comes into contact with real-world usage. This paper should be required reading for anyone interested in designing and developing core infrastructure software that is to be offered as a service.

Burrows begins with a discussion of the design principles chosen as the basis for Chubby. Why make it a centralized service instead of a library? Why is it a lock service, and what kind of locking is it used for? Chubby not only provides locks, but also serves small files to facilitate sharing of metadata about distributed system state for its clients. Given that it is serving files, how many clients should Chubby expect to support, and what will that mean for the caching and change notification needs?

After discussing the details of the design, system structure, and API, Burrows gets into the nitty-gritty of the implementation. Building a highly sensitive centralized service for critical operations such as distributed locking and name resolution turns out to be quite difficult. Scaling the system to tens of thousands of clients meant being smart about caching and deploying proxies to handle some of the load. The developers misused and abused the system by accident, using features in unpredictable ways, attempting to use the system for large data storage or messaging. The Chubby maintainers resorted to reviewing other teams' planned uses of Chubby and denying access until review was satisfied. Through all of this we can see that the challenge in building a consensus system goes far beyond implementing a correct algorithm. We are still building a system and must think as carefully about its design and the users we will be supporting.

Can We Make This Easier?

Ongaro, D., Ousterhout, J. 2014. In search of an understandable consensus algorithm. Proceedings of the Usenix Annual Technical Conference: 305-320.
https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

Finally we come to the question, have we built ourselves into unnecessary complexity by taking it on faith that Paxos and its close cousins are the only way to implement consensus? What if there were an algorithm that we could also show to be correct but was designed to be easier for people to comprehend and implement correctly?

Raft is a consensus algorithm written for managing a replicated log but designed with the goal of making the algorithm itself more understandable than Paxos. This is done both by decomposing the problem into pieces that can be implemented and understood independently and by reducing the number of states that are valid for the system to hold.

Consensus is decomposed into issues of leader election, log replication, and safety. Leader election uses randomized election timeouts to reduce the likelihood of two candidates for leader splitting the vote and requiring a new round of elections. It allows candidates for leader to be elected only if they have the most up-to-date logs. This prevents the need for transferring data from follower to leader upon election. If a follower's log does not match the expected state for a new entry, the leader will replay entries from earlier in its log until it reaches a point at which the logs match, thus correcting the follower. This also means that a history of changes is stored in the logs, providing a side value of letting clients read (some) historical entries, should they desire.

The authors then show that after teaching a set of students both Paxos and Raft, the students were quizzed on their understanding of each and scored meaningfully higher on the Raft quiz. Looking around the current state of consensus systems in industry, we can see this play out in another way: namely, several new consensus systems have been created since 2014 based on Raft, where previously there were very few reliable and successful open-source systems based on Paxos.

Bottlenecks, Single Points of Failure, and Consensus

Developers are often tempted to use a centralized consensus system to serve as the system of record for distributed coordination. Explicit coordination can make certain problems much easier to reason about and correct for; however, that puts the consensus system in the position of the bottleneck or critical point of failure for the other systems that rely on it to make progress. As we can see from these papers, making a centralized consensus system production-ready can come at the cost of adding optimizations and recovery mechanisms that were not dreamed of in the original Paxos literature.

What is the way forward? Arguably, writing systems that do not rely on centralized consensus brokers to operate safely would be the best option, but we are still in the early days of coordination-avoidance research and development. While we wait for more evolution on that front, Raft provides an interesting alternative, an algorithm designed for readability and general understanding. The impact of having an easier algorithm to implement is already being felt, as far more developers are embedding Raft within distributed systems and building specifically tailored Raft-based coordination brokers. Consensus remains a tricky problem—but one that is finally seeing a diversity of approaches to reaching a solution.

IMPLICATIONS OF NVM ON DATABASE MANAGEMENT SYSTEMS

BY JOY ARULRAJ AND ANDREW PAVLO

The advent of NVM (nonvolatile memory) will fundamentally change the dichotomy between memory and durable storage in a DBMS (database management system). NVM is a broad class of technologies—including phase-change memory, memristors, and STT-MRAM (spin-transfer torque-magnetoresistive random-access memory)—that provide low-latency reads and writes on the same order of magnitude as DRAM (dynamic random-access memory), but with persistent writes and large storage capacity like an SSD (solid-state drive). Unlike DRAM, writes to NVM are expected to be more expensive than reads. These devices also have limited write endurance, which necessitates fewer writes and wear-leveling to increase their lifetimes.

The first NVM devices released will have the same form factor and block-oriented access as today's SSDs. Thus, today's DBMSes will use this type of NVM as a faster drop-in replacement for their current storage hardware.

By the end of this decade, however, NVM devices will support byte-addressable access akin to DRAM. This will require additional CPU architecture and operating-system support for persistent memory. This also means that existing DBMSes are unable to take full advantage of NVM because their internal architectures are predicated on the assumption that memory is volatile. With NVM, many of the components of legacy DBMSes are unnecessary and will degrade the performance of data-intensive applications.

We have selected three papers that focus on how the emergence of byte-addressable NVM technologies will impact the design of DBMS architectures. The first two present new abstractions for performing durable atomic updates on an NVM-resident database and recovery protocols for an NVM DBMS. The third paper addresses the write-endurance limitations of NVM by introducing a collection of write-limited query-processing algorithms. Thus, this selection contains novel ideas that can help leverage the unique set of attributes of NVM devices for delivering the features required by modern data-management applications. The common theme for these papers is that you cannot just run an existing DBMS on NVM and expect it to leverage its unique set of properties. The only way to achieve that is to come up with novel architectures, protocols, and algorithms that are tailor-made for NVM.

ARIES Redesigned for NVM

Coburn, J., et al. 2013. From ARIES to MARS: transaction support for next-generation, solid-state drives. Proceedings of the 24th ACM Symposium on Operating Systems Principles: 197-212.
http://queue.acm.org/rfp/vol14iss3.html

ARIES is considered the standard for recovery protocols in a transactional DBMS. It has two key goals: first, it provides an interface for supporting scalable ACID (atomicity, consistency, isolation, durability) transactions; second, it maximizes performance on disk-based storage systems. In this paper, the authors focus on how ARIES should be adapted for NVM-based storage.

Since random writes to the disk whenever a transaction updates the database obviously decrease performance, ARIES requires that the DBMS first record a log entry in the write-ahead log (a sequential write) before updating the database itself (a random write). It adopts a *no-force* policy wherein the updates are written to the database lazily after the transaction commits. Such a policy assumes that sequential writes to nonvolatile storage are significantly faster than random writes. The authors, however, demonstrate that this is no longer the case with NVM.

The MARS protocol proposes a new hardware-assisted logging primitive that combines multiple writes to arbitrary storage locations into a single atomic operation. By leveraging this primitive, MARS eliminates the need for an ARIES-style undo log and relies on the NVM device to apply the redo log at commit time. We are particularly fond of this paper because it helps in better appreciating the intricacies involved in designing the recovery protocol in a DBMS for guarding against data loss.

Near-Instantaneous Recovery Protocols

Arulraj, J., Pavlo, A., Dulloor, S. R. 2015. Let's talk about storage and recovery methods for nonvolatile memory database systems. Proceedings of the ACM SIGMOD International Conference on Management of Data: 707-722.
http://queue.acm.org/rfp/vol14iss3.html

This paper takes a different approach to performing durable atomic updates on an NVM-resident database than the previous paper. In ARIES, during recovery the DBMS first loads the most recent snapshot. It then replays the redo log to ensure that all the updates made by committed transactions are recovered. Finally, it uses the undo log to ensure that the changes made by incomplete transactions are not present in the database. This recovery process can take a lot of time, depending on the load on the system and the frequency with which snapshots are taken. Thus, this paper explores whether it is possible to leverage NVM's properties to speed up recovery from system failures.

The authors present a software-based primitive called a *nonvolatile pointer*. When a pointer points to data residing on NVM, and is itself stored on NVM, then it will remain valid even after the system recovers from a power failure. Using this primitive, the authors design a library of nonvolatile data structures that support durable atomic updates. They propose a recovery protocol that, in contrast to MARS, obviates the need for an ARIES-style redo log. This enables the system to skip replaying the redo log, and thereby allows the NVM DBMS to recover the database almost instantaneously.

Both papers propose recovery protocols that target an NVM-only storage hierarchy. The generalization of these protocols to a multitier storage hierarchy with both DRAM and NVM is a hot topic in research today.

Trading Expensive Writes for Cheaper Reads

Viglas, S. D. 2014. Write-limited sorts and joins for persistent memory. Proceedings of the VLDB Endowment 7(5): 413-424.
http://www.vldb.org/pvldb/vol7/p413-viglas.pdf

The third paper focuses on the higher write costs and limited write-endurance problems of NVM. For several decades algorithms have been designed for the random-access machine model where reads and writes have the same cost. The emergence of NVM devices, where writes are more expensive than reads, opens up the design space for new write-limiting algorithms. It will be fascinating to see researchers derive new bounds on the number of writes that different kinds of query-processing algorithms must perform.

Viglas presents a collection of novel query-processing algorithms that minimize I/O by trading off expensive NVM writes for cheaper reads. One such algorithm is the *segment sort*. The basic idea is to use a combination of two sorting algorithms—external merge sort and selection sort—that splits the input into two segments that are then processed using a different algorithm. The selection-sort algorithm uses extra reads, and writes out each element in the input only once at its final location. By using a combination of these two algorithms, the DBMS can optimize both the performance and the number of NVM writes.

Game Changer for DBMS Archtectures

NVM is a definite game changer for future DBMS architectures. It will require system designers to rethink many of the core algorithms and techniques developed over the past 40 years. Using these new storage devices in the manner prescribed by these papers will allow DBMSes to achieve better performance than what is possible with today's hardware for write-heavy database applications. This is because these techniques are designed to exploit the low-latency read/writes of NVM to enable a DBMS to store less redundant data and incur fewer writes. Furthermore, we contend that existing in-memory DBMSes are better positioned to use NVM when it is finally available. This is because these systems are already designed for byte-addressable access methods, whereas legacy disk-oriented DBMSes will require laborious and costly overhauls in order to use NVM correctly, as described in these papers.

Peter Bailis *is an assistant professor of computer science at Stanford University. His research in the Future Data Systems group (http://futuredata.stanford.edu/) focuses on the design and implementation of next-generation data-intensive systems. He received a Ph.D. from UC Berkeley in 2015 and an A.B. from Harvard in 2011, both in computer science.*

Camille Fournier *is a writer, speaker, and entrepreneur. Formerly the CTO of Rent the Runway, she serves on the technical oversight committee for the Cloud Native Computing Foundation, as a Project Management Committee member of the Apache ZooKeeper project, and a project overseer of the Dropwizard web framework. She has an M.S. from the University of Wisconsin-Madison and a B.S. from*

Carnegie Mellon University. You can find more of her writing at elidedbranches.com.

Joy Arulraj *is a Ph.D. candidate at Carnegie Mellon University. He is interested in the design and implementation of next-generation database management systems, particularly in the areas of realtime data analytics, self-driving modules, and adoption of nonvolatile memory technologies. He earned an M.S. in computer sciences from the University of Wisconsin, Madison, and received a B.E. in computer science and engineering from the College of Engineering, Guindy. He is a recipient of the 2016 Samsung Ph.D. Fellowship.*

Andy Pavlo *is an assistant professor of databaseology in the computer science department at Carnegie Mellon University.*

CONTENTS