Nova: Diffused Database Processing using Clouds of Components [Vision Paper]*

Shahram Ghandeharizadeh, Haoyu Huang, Hieu Nguyen
Database Laboratory Technical Report 2019-01
Computer Science Department, USC
Los Angeles, California 90089-0781
{shahram,haoyuhua,hieun}@usc.edu

Abstract

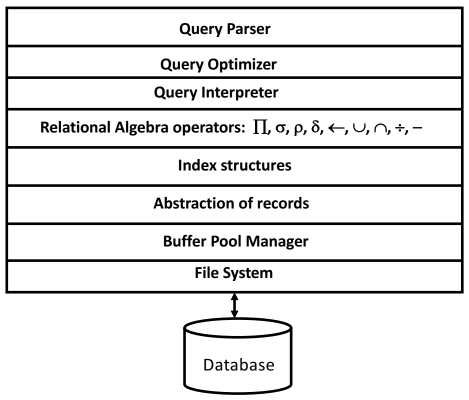
Nova proposes a departure from today's complex monolithic database management systems (DBMSs) as a service using the cloud. It advocates a server-less alternative consisting of a cloud of simple components that communicate using high speed networks. Nova will monitor the workload of an application continuously, configuring the DBMS to use the appropriate implementation of a component most suitable for processing the workload. In response to load fluctuations, it will adjust the knobs of a component to scale it to meet the performance requirements of the application. The vision of Nova is compelling because it adjusts resource usage, preventing either over-provisioning of resources that sit idle or over-utilized resources that yield a low performance, optimizing total cost of ownership. In addition to introducing Nova, this vision paper presents key research challenges that must be addressed to realize Nova. We explore two challenges in detail.

1 Introduction

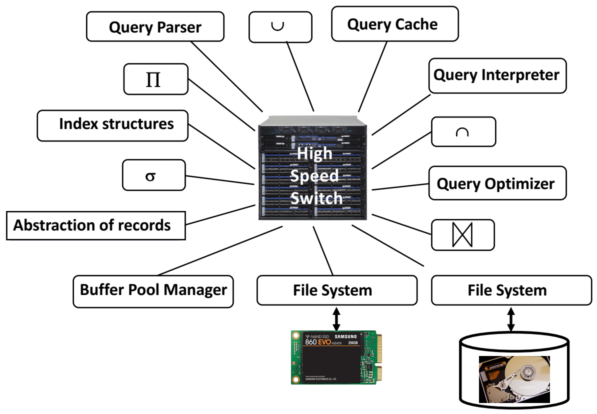
Database management systems (DBMSs) are a critical part of diverse applications in science, business, health-care, and entertainment. Today's DBMSs, both SQL and NoSQL [9], are complex and consist of numerous knobs to control their performance and scalability characteristics [31, 24]. Numerous cloud providers offer one or more of these DBMSs as a service. For example, at the time of this writing, Amazon RDS offers MariaDB, MySQL, PostgreSQL, Oracle, Microsoft SQL Server, and Amazon Aurora as a DBMS in the cloud.

Nova proposes a departure from today's complex DBMSs, see Fig. 1. It envisions simple components that communicate using high speed networks such as those with *remote direct memory access* (RDMA) capabilities to realize a DBMS. A component may be a file system, a buffer pool manager, abstraction of data items as records, documents, and key-value pairs,

*This paper appeared in the *15th IEEE International Conference on Beyond Database Architectures and Structures (BDAS)*, Ustron, Poland, May 28-31, 2019.



(a) A monolithic DBMS.



(b) Nova's simple components will communicate using high speed networks.

Figure 1: Architecture of today's monolithic DBMS and the envisioned Nova.

indexing structures, data encryption and decryption techniques, compression techniques, a library of data operators such as the relational algebra operator select/project/join/etc., a query result cache, a concurrency control protocol, a crash recovery protocol, a query processing engine, a query language, a query optimizer, etc. These components may be services in a cloud and inter-operate to realize a DBMS dynamically. The application logic itself may run in Nova's infrastructure, external to it on a device (e.g., a mobile client such as a smart-phone), or a hybrid where part of the application logic runs in the Nova cloud and other parts of it execute on devices.

The rest of this paper is organized as follows. Section 2 presents the vision of Nova and articulates its research challenges. Sections 3 and 4 focus on the first two research challenges (enumerated questions) of Section 2. Section 6 describes how Nova is related to several inspiring disciplines. Brief conclusions are provided in Section 7.

2 Overview and Research Challenges

When compared with today's service provider such as Amazon AWS, an application developer will use Nova without sizing a server. Instead, the developer will specify the desired performance (response time and throughput) and data availability (mean time to failure MTTF, mean time to data loss MTDL) characteristics of her application, leaving it to Nova to size its components to meet these specifications. Once the application is deployed, Nova will adjust the amount of resources allocated to each component to meet the application's performance goals in response to system load.

The Nova cloud may span hundreds of data centers operated by the same provider or different providers. It may host a large number of components for different DBMSs serving diverse applications. A Nova component may have variants suitable for different workloads. For example, a component that provides abstraction of data records may implement either a row store or a column store [37, 28]. The row store is suitable for an online transaction processing (OLTP) workload while a column store is suitable for an online analytical processing

(OLAP) workload.

Two or more components may inter-operate to provide a functionality. For example, a row store representation of records inter-operates with a storage component for storage and retrieval of data. Alternative variants of a storage components may be a block-store such as an Ext4 file system [29], an in-memory store such as RAMCloud [30], or a Log-Structured Merge-tree [27] (LSM-Tree) key-value store such as LevelDB [20] or RocksDB [13].

A component will have its own configuration knob [34, 36] to finetune its performance and data availability characteristics. There will be intra-component communication for it to scale horizontally. Nova is responsible for adjusting the knobs of a component based on the characteristics of a workload, e.g., read to write ratio, rate of conflicts among concurrent accesses to data, pattern of access to data, etc.

Nova will be a transformative system because its simple components are well defined. It will monitor an application's workload and will configure each component to maximize its performance. A component will scale both vertically and horizontally, providing high availability and elasticity characteristics. Either an expert system or a machine learning algorithm may perform these tasks. These may be similar in spirit to [39, 7, 40] for fine-tuning performance of today's complex DBMSs. Over time, as the application workload evolves, Nova will identify components that are more suitable than the current DBMS components. If a replacement requires an expensive data translation and migration phase, it will notify a database administrator to confirm several candidate off-peak time slots for the component replacement. If a replacement is trivial, e.g., an optimistic concurrency control protocol instead of locking, it will put it in effect transparently and report both the change and the observed performance improvement.

The vision of Nova is made possible by recent advances in CPU processing, network communication, and storage technology. High speed networks such as RDMA and 5G are essential for implementing Nova's high performance components that inter-operate with one another. Several studies report RDMA bandwidths comparable to memory bandwidth, see [4] for an example. High density memory in the form of volatile DRAM and non-volatile SSD enable Nova to maintain state information and replicate this data for high availability. Inexpensive fast CPUs implement components that were traditionally software stacks of a monolithic system. Finally, operating systems such as LegoOS [35] are in synergy with Nova's vision, enabling Nova to provision the right amount of resources from abstraction of a device managed by LegoOS.

Nova's hypotheses are as follows. First, alternative protocols and specifications can be implemented as simple components with well defined functionality. This will enable Nova to bring together those that best satisfy the application's workload in a DBMS. Second, simple components are easier to manage and scale. Third, the data availability requirement of an application can be used to provide smarter execution paradigms. If replication of data across two or more servers satisfies the mean time to data loss (MTDL) of an application, for a component that requires durability property of transactions (e.g., LSM-Tree file system [27]), we propose to replicate its data and eliminate log records altogether. The idea is to provide a more efficient execution paradigm by using memory and network to either (a) eliminate state data such as log records or (b) perform expensive writes to Hard Disk Drive (HDD) and Solid State Disk (SSD) asynchronously. Section 4 presents this idea in greater detail.

Nova is broad in scope and raises many interesting research topics.

1. What is the overall software architecture of Nova? Nova's components must interoperate to realize a DBMS. They may communicate using message passing or shared memory. The latter may be implemented using RDMA calls. Data encryption is a central component to secure data.

2. How does Nova implement the concept of a transaction and its atomicity, consistency, isolation, and durability (ACID) semantics? This includes an investigation of weaker forms of consistency.

3. How does Nova decide the number of each component for an application and across applications? System size impacts response time and throughput observed by an application. Load imposed by the workload also impacts these performance metrics and may be diurnal in nature. We envision Nova to use service level agreements required by an application to continuously monitor performance metrics and adjust the number of components dynamically in response to system load. The adjustment may be either detective or preventive.

4. How does Nova change one or more components of a deployed DBMS? Some of these may be driven by external factors. For example, an application developer may desire to switch from a relational model to a document model. Others may be driven internally. For example, a machine learning algorithm may identify a row store to be more appropriate than a column store, a lock-based protocol instead of an optimistic concurrency protocol, a log-structured file system instead of a regular Ext4 file system, and others. Some of these changes may require live data migration, translation, and re-formatting of data. Ideally, Nova should perform these operations with no disruption in processing application requests. They may impact system load. However, they can be processed during off-peak hours identified by Nova and confirmed by a system administrator.

5. How secure is Nova? This challenging research question examines the interplay between components selected for an application and where encryption is used to store and process data. Alternative Nova configurations may enhance security of data for an application. They may also require additional resources or slow down the application. It is important to provide an intuitive presentation of this tradeoff to an application developer to empower them to make an informed decision.

6. What would be an intelligent configuration advisor for Nova? Nova provides a rich spectrum of components with diverse tradeoffs. Ideally, a system architect should specify their workload and Nova's configuration advisor should suggest candidate components suitable for that workload. Similar to a query optimizer, the advisor must provide an explain feature for an architect to interrogate Nova's choice of components and how efficient they are in meeting the workload requirements.

The next two sections focus on the first two research questions.

3 Nova's software architecture

Nova's envisioned software architecture will be more than a classification of components with well defined interfaces. Components must inter-operate using high speed networks and RDMA. To illustrate the challenge, consider a traditional block-based buffer pool manager and a block-based file system. Nova will require these to inter-operate using RDMA. A key question is how will one scale the buffer pool manager to increase the amount of memory assigned to the buffer pool manager? One answer is to scale-up [10] as long as the server hosting the buffer pool manager has sufficient physical memory. However, once this memory is exhausted, the buffer pool manager must either migrate [26, 38, 22] to a server with a larger memory size or scale to run across multiple servers. Each has its own interesting research questions. With the first, how will Nova migrate the buffer pool manager without disrupting service? What happens to the RDMA connection of components that use the buffer pool manager? How will the buffer pool manager re-establish its RDMA connection with the file manager?

With the second approach, different instances of the buffer pool manager component assigned to different servers will be coordinated to act as one component. File system pages may be hash partitioned across these instances. The instances will be aware of one another and collaborate to route a request to the right instance. Research questions include: How will the instances communicate with one another, is it via RDMA or a regular message passing network? What is the trade off associated with separating the data fabric from the network fabric? Once scaled out, how will these instances communicate with the file system component using RDMA? Similarly, how will the software layer that uses the buffer pool manager communicate with the different buffer pool instances?

In addition to the above, Nova's architecture raises the following research questions:

1. Is it possible to scale each component independently? For example, for a workload with a low read-write ratio, is it possible to have tens of instances of the file system, 1 buffer pool manager with 1 lock manager and 1 log manager and a few instances of the indexing technique?

2. Is it possible to add components dynamically? For example, once the workload of an application evolves to exhibit a high read-write ratio (say 500 to 1 [5]), is it possible to insert a query result cache in its DBMS dynamically?

3. How to replace one or more components of a DBMS with another component to make it more suitable for a workload? For example, a LSM-Tree filesystems [27] such as LevelDB [20] and RocksDB [13] is suitable for workloads with a low read-write ratio. How would one use such a file system instead of an Ext4 file system? Given a workload with a low read-write ratio, how will Nova quantify the performance improvement observed with this change?

4. How to tolerate server failures without disruption of service? For example, if the server hosting the lock manager fails, is it important to recover its state at the time of failure? If so, how will Nova maintain the state of the lock manager and restore it to continue operation?

5. How will Nova monitor its components and control their placement across servers to resolve bottlenecks?

4 Nova Transactions

Nova's support for transactions with ACID semantics simplify design and implementation of applications [23, 25]. They empower an application developer to reason about application behavior to debug and test it effectively. Transactions may be a feature of a component. For example, durability of writes as transactions may be provided by an implementation of LSM-Tree such as LevelDB or RocksDB. Several studies describe use of RDMA and data locality to scale a transaction processing workload such as TPC-C to a large number of nodes [6, 42].

An accepted practice to provide atomicity and durability is to generate log records and flush them to persistent store prior to either a write (with LevelDB or RocksDB) or a transaction commit (with MySQL, Oracle, or MongoDB). Nova will offer alternatives to enhance performance. These alternatives are based on the key insight that today's racks of servers offer high mean time to failures, MTTF. Based on this, one may quantify mean time to data loss (MTDL) of replicating a data item two or more times. If this MTDL is lower than an application's tolerable MTDL then Nova may replicate data instead of generating log records to satisfy atomicity and durability properties of transactions. Should log records be required for auditing of transactions then these log records can be replicated across multiple servers. These log records are subsequently flushed to a HDD/SSD file system and deleted from memory asynchronously. This speeds up performance compared with today's state of the art that flushes log records to HDD/SSD synchronously.

To illustrate the superiority of the proposed approach, Fig. 2 shows an evaluation of MySQL using the TPC-C benchmark [33]. We show the transactions per minute of MySQL as a function of the number of concurrent threads generating the TPC-C workload. All results are gathered from a cluster of Emulab [41, 15] nodes. Each node has two 2.4 GHz 64-bit 8-Core E5-2630 Haswell processors, 8.0 GT/s bus speed, 20 MB cache, 64 GB of RAM, and connects to the network using 10 Gigabits networking card. Each node runs Ubuntu OS version 16.04 (kernel 4.10.0). We adjust knobs of MySQL to maximize its performance, e.g., we increase its default memory size from 128 Megabytes to 48 Gigabytes.

We show two MySQL configurations. One using HDD and the other using SSD, labeled MySQL HDD and MySQL SSD, respectively. SSD is faster than HDD. Hence, tpmC with SSD is higher and levels off with 5 concurrent threads due to transactions waiting on one another in MySQL's lock manager.

When MySQL is extended with a write-back client-side cache [17] (top line), its performance is enhanced more than two folds. This is because all writes to the data store are performed by generating buffered writes across the network. Background threads apply the buffered writes to the data store asynchronously.

Write-back's tpmC also levels off beyond 5 threads due to contention for leases in the caching layer. Note that with the write-back cache, choice of SSD or HDD for MySQL does not impact[1] tpmC because almost all reads and writes are processed using the cache.

[1]The impact is on the amount of required memory because HDD slows down the rate at which buffered

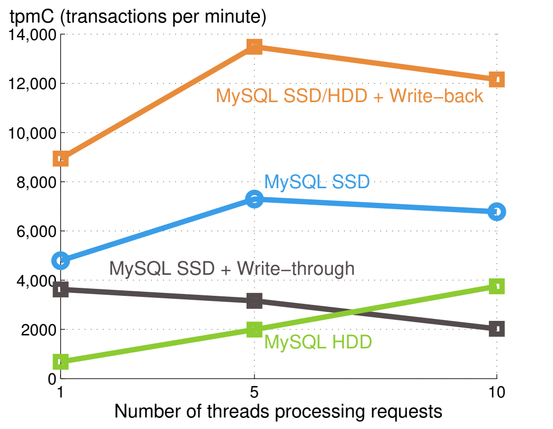


Figure 2: TPC-C's tpmC of MySQL by itself and with IQ-Twemcached [18] configured with write-back and write-through.

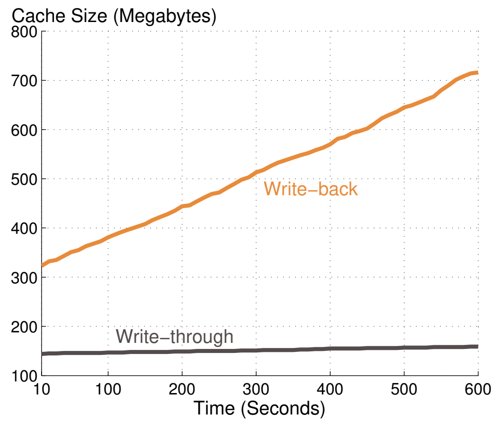


Figure 3: Memory size of IQ-Twemcached with write-back and write-through, 5 TPC-C threads, 1 warehouse.

In Fig. 2, the tpmC with the same cache configured using the write-through policy (black line) is lower than MySQL by itself. This is because 92% of TPC-C workload is writes. The overhead of updating the cache entries outweighs the benefits observed by reads that constitute 8% of the workload.

Figure 3 shows the size of memory required by write-back and write-through as TPC-C executes its workload. TPC-C generates new orders that increase both the database size and the number of entries in the caching layer. The difference between the write-back and write-through approximates the size of buffered writes generated by the write-back policy.

With write-back, we observe a 19% decrease in the reported tpmC when buffered writes are replicated thrice. We replicate buffered writes to satisfy the MTDL requirements of the target application. The buffered writes and their replicas are deleted asynchronously once they are applied to the data store.

Nova will improve performance of a component by replicating its transactional data in order to (a) make writes durable, (b) meet the MTDL requirements of an application, and (c) perform the transactional write to a mass storage device (HDD/SSD) asynchronously. We hypothesize the latter to enhance performance and scalability of the component. These replicas are deleted once the data is written to HDD/SSD. As demonstrated by the results of Fig. 2 and Fig. 3, these execution paradigms require memory. Nova will assume availability of servers with a large amount of memory to enable these execution paradigms.

There is a tradeoff between space, performance observed by the foreground load, and MTDL. Specifically, one may trade space for time and MTDL by maintaining a priority queue of read and write requests. Nova may give a low priority to processing log records to minimize impact on the foreground requests. This will cause replicas of the log records to stay across Nova's server for a longer time. However, it will enhance the performance observed by the foreground system load.

writes are applied and deleted.

5 In-memory Data Containers, MemDCs

To make the proposed ideas feasible, we present the concept of physical in-memory data containers (MemDCs). MemDC is envisioned as a component managed by Nova and created based on a policy. The policy may specify a MTDL requirement or the number of replicas for a data item. A MemDC may span servers across different racks of the same data center and potentially across data centers. Once data is assigned to a MemDC, the MemDC replicates it across its servers.

Design and implementation of MemDCs raise many interesting research questions. First, should a component that requires durable writes delegate persistent write of its data to MemDC? What is the tradeoff between the component performing the write asynchronously versus MemDC performing it? Does the former simplify component design? If the answer is affirmative then how does MemDC maintain the order in which data items should be written to the persistent store? The answer to the latter question is important when there are dependencies between log records and/or written data items.

Nova may consist of a large number (trillions) of MemDCs with different policies. How does Nova place these MemDCs across racks of servers in a data center and across data centers? How does it modify the initial placement in response to an evolving system load? A concept similar to MemDC is presented as a μshard by Facebook [3]. MemDC will be a super-set of a μshard in the form of memory for both application data and transient data such as log records.

In passing, we note that MTDL as a metric depends on mean time to repair (MTTR). A MemDC may detect loss of an in-memory replica due to a server failure and repair it by constructing a replica on a different server. This form of repair will enhance MTDL observed by an application.

Second, how does Nova implement non-idempotent writes? A non-idempotent write must be applied only once. In the presence of arbitrary failures, Nova may either (a) convert a non-idempotent write to an idempotent one or (b) maintain sufficient information to guarantee a non-idempotent write is applied only once. It is important to quantify the tradeoffs associated with these alternatives to develop a methodology for the components that utilize MemDCs.

Finally, based on our experiences with implementing a write-back policy using query result caches, there are a variety of race conditions that must be identified and addressed [19]. We anticipate Nova to use leases to address these race conditions. Leases [21, 19] are different than locks in that they have a finite lifetime. This concept is useful when the server hosting the component that owns the lease fails. Once a lease expires, its data becomes available again.

6 Related Work

Vision of Nova is at the intersection of several inspiring disciplines, including extensible DBMSs, parallel DBMSs, web services (WS) and grid computing, federated DBMSs and polystores, and distributed systems. We describe these in turn.

Extensible DBMSs [12] advocate use of components that plug-n-play to provide customized data services to an application. Nova is different in two ways. First, its components will communicate using the network (instead of being part of a monolithic system). Second, we envision Nova as a framework that adjusts the components of a DBMS continuously to enhance efficiency without violating an application's service level agreements. It will do so by continuously monitoring the key metrics of an application's workload (e.g., its read to write ratio) and matching it with the capabilities of candidate components.

Nova is similar to parallel DBMSs [11] in that its components communicate with one another and the data may flow across multiple components, implementing both inter and intra component parallelism. It is different than parallel DBMSs in that the components are (a) shared among different applications in a cloud and (b) the components are not decided in advance. We envision Nova to adjust the number of components per required service level agreement (SLA) of an application. We also envision Nova to implement security policies across components shared among different applications.

Web services [2, 1] and grid computing systems [16] consist of services that are assembled together. They may use security and encryption when exchanging data between services. These services communicate using IP network. Vision of Nova is different in that its micro-services are at a finer granularity, e.g., concurrency control protocols, buffer pools, etc. It envisions use of RDMA with servers accessing and sharing memory to implement these services efficiently, enabling them to scale horizontally.

Federated DBMSs such as Garlic [8] provide a single interface for different DBMSs. Their successor is a multi-store [32] or polystore [14] that supports multiple data models and databases, combining concepts from federated and parallel DBMSs. A polystore includes a middleware that provides an abstraction of data and implement a programming API for query planning, optimization, and execution using its databases. It may support multiple query languages for different applications. Similar to a parallel DBMS, a polystore may partition data to scale and use data flow execution paradigms for query processing. Nova is similar to polystores in that it envisions a DBMS of components that may support multiple data models and queries that execute across them. At the same time, it envisions switching from one data model to another if the workload justifies the switch. Nova is different than polystores in that it does not assume a DBMS as one of its components. It envisions a DBMS as a collection of components to be tailored together based on the application workload. A component of Nova may have a simple interface and functionality, e.g., get, put, append, etc. Moreover, Nova's components may share memory using fast networks such as RDMA. One may realize a polystore using multiple Nova DBMSs.

Finally, Nova is inspired by distributed systems such as LegoOS that implement services of an operating system as components. Nova is in synergy with such an operating system by implementing the corresponding DBMS service using the appropriate service of the operating system. Nova is different because it implements a query language such as SQL and ACID transactions.

7 Conclusions

This vision paper presents the vision of Nova as a DBMS realized using simple components. A cloud provider may deploy these components across one or more data centers.

These components may run on devices and services of a modular operating system such as LegoOS [35], leveraging the scalability of its underlying hardware. The key advantage of simple components will be their simple knobs for scale up, scale out, high availability, and elasticity. Nova will select components of a DBMS with the objective to meet the performance requirements of an application workload. It will adjust the knobs of these components in response to the application workload and may change one or more components of a DBMS as the workload of the application evolves.

8 Acknowledgments

We gratefully acknowledge use of Utah Emulab network testbed [41] ("Emulab") for all experimental results presented in this paper. We thank anonymous BDAS 2019 reviewers for their valuable comments.

References

[1] E. Alwagait and S. Ghandeharizadeh. A Comparison of Alternative Web Service Allocation and Scheduling Policies. In *2004 IEEE International Conference on Services Computing, September, Shanghai, China*, pages 319–326, 2004.

[2] E. Alwagait and S. Ghandeharizadeh. DeW: A Dependable Web Services Framework. In *(RIDE-WS-ECEG 2004), March, Boston, MA*, pages 111–118, 2004.

[3] M. Annamalai, K. Ravichandran, H. Srinivas, I. Zinkovsky, L. Pan, T. Savor, D. Nagle, and M. Stumm. Sharding the Shards: Managing Datastore Locality at Scale with Akkio. In *OSDI*, pages 445–460, Carlsbad, CA, 2018. USENIX Association.

[4] C. Binnig, A. Crotty, A. Galakatos, T. Kraska, and E. Zamanian. The End of Slow Networks: It's Time for a Redesign. *Proc. VLDB Endow.*, 9(7):528–539, 2016.

[5] N. Bronson, T. Lento, and J. L. Wiener. Open Data Challenges at Facebook. In *31st IEEE International Conference on Data Engineering, ICDE, Seoul, South Korea, April 13-17, 2015*, pages 1516–1519, 2015.

[6] Q. Cai, W. Guo, H. Zhang, D. Agrawal, G. Chen, B. C. Ooi, K.-L. Tan, Y. M. Teo, and S. Wang. Efficient Distributed Memory Management with RDMA and Caching. *Proc. VLDB Endow.*, 11(11):1604–1617, July 2018.

[7] Z. Cao, V. Tarasov, S. Tiwari, and E. Zadok. Towards Better Understanding of Black-box Auto-tuning: A Comparative Analysis for Storage Systems. In *Usenix Annual Technical Conference*, pages 893–907, Berkeley, CA, USA, 2018.

[8] M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, J. Thomas, J. H, and E. L. Wimmers. Towards Heterogeneous Multimedia Information Systems: The Garlic Approach. In *In RIDE-DOM*, pages 124–131, 1995.

[9] R. Cattell. Scalable SQL and NoSQL Data Stores. *SIGMOD Rec.*, 39:12–27, May 2011.

[10] S. Das, F. Li, V. R. Narasayya, and A. C. König. Automated Demand-driven Resource Scaling in Relational Database-as-a-Service. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 1923–1934, New York, NY, USA, 2016. ACM.

[11] D. DeWitt, S. Ghandeharizadeh, D. Schneider, A. Bricker, H.-I. Hsiao, and R. Rasmussen. The Gamma Database Machine Project. *Knowledge and Data Engineering*, 2(1), 1990.

[12] K. R. Dittrich and A. Geppert, editors. *Component Database Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[13] S. Dong, M. Callaghan, L. Galanis, D. Borthakur, T. Savor, and M. Strum. Optimizing Space Amplification in RocksDB. In *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*, 2017.

[14] A. Elmore, J. Duggan, M. Stonebraker, M. Balazinska, U. Cetintemel, V. Gadepally, J. Heer, B. Howe, J. Kepner, T. Kraska, S. Madden, D. Maier, T. Mattson, S. Papadopoulos, J. Parkhurst, N. Tatbul, M. Vartak, and S. Zdonik. A Demonstration of the BigDAWG Polystore System. *Proc. VLDB Endow.*, 8(12):1908–1911, Aug. 2015.

[15] Emulab. Emulab. `https://www.emulab.net/portal/frontpage.php`. Accessed: 2018-11-15.

[16] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *The International Journal of High Performance Computing Applications*, 15(3):400–222, 2001.

[17] S. Ghandeharizadeh and H. Nguyen. Design, Implementation, and Evaluation of Write-Back Policy with Cache Augmented Data Stores. Technical Report 2018-07, USC Database Lab, 2018. Submitted to VLDB 2019: In revision.

[18] S. Ghandeharizadeh, J. Yap, and H. Nguyen. IQ-Twemcached, http://dblab.usc.edu/users/iq/.

[19] S. Ghandeharizadeh, J. Yap, and H. Nguyen. Strong Consistency in Cache Augmented SQL Systems. *Middleware*, December 2014.

[20] S. Ghemawat and J. Dean. LevelDB. `https://github.com/google/leveldb`. Accessed: 2018-11-15.

[21] C. Gray and D. Cheriton. Leases: An Efficient Fault-tolerant Mechanism for Distributed File Cache Consistency. *SIGOPS Oper. Syst. Rev.*, 23(5):202–210, Nov. 1989.

[22] C. Kulkarni, A. Kesavan, T. Zhang, R. Ricci, and R. Stutsman. Rocksteady: Fast Migration for Low-latency In-memory Storage. In *SOSP*, pages 390–405, New York, NY, USA, 2017. ACM.

[23] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Don't Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS. In *SOSP*, Oct. 2011.

[24] L. Ma, D. V. Aken, A. Hefny, G. Mezerhane, A. Pavlo, and G. J. Gordon. Query-based Workload Forecasting for Self-Driving Database Management Systems. In *SIGMOD, Houston, TX, USA, June 10-15, 2018*, pages 631–645, 2018.

[25] S. A. Mehdi, C. Littley, N. Crooks, L. Alvisi, N. Bronson, and W. Lloyd. I Can't Believe It's Not Causal! Scalable Causal Consistency with No Slowdown Cascades. In *NSDI, Boston, MA, USA, March 27-29, 2017*, pages 453–468, 2017.

[26] B. N. Memon, X. C. Lin, A. Mufti, A. S. Wesley, T. Brecht, K. Salem, B. Wong, and B. Cassell. RaMP: A Lightweight RDMA Abstraction for Loosely Coupled Applications. In *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*, Boston, MA, 2018. USENIX Association.

[27] P. E. O'Neil, E. Cheng, D. Gawlick, and E. J. O'Neil. The Log-Structured Merge-Tree (LSM-Tree). *Acta Inf.*, 33(4):351–385, 1996.

[28] P. E. O'Neil and D. Quass. Improved Query Performance with Variant Indexes. In *SIGMOD, May 13-15, 1997, Tucson, Arizona, USA.*, pages 38–49, 1997.

[29] opensource.com. Understanding Linux Filesystems: Ext4 and Beyond.

[30] J. Ousterhout, A. Gopalan, A. Gupta, A. Kejriwal, C. Lee, B. Montazeri, D. Ongaro, S. J. Park, H. Qin, M. Rosenblum, S. Rumble, R. Stutsman, and S. Yang. The RAMCloud Storage System. *ACM Trans. Comput. Syst.*, 33(3):7:1–7:55, Aug. 2015.

[31] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, P. Menon, T. C. Mowry, M. Perron, I. Quah, S. Santurkar, A. Tomasic, S. Toor, D. V. Aken, Z. Wang, Y. Wu, R. Xian, and T. Zhang. Self-Driving Database Management Systems. In *CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*, 2017.

[32] E. Pluciennik and K. Zgorzalek. The Multi-model Databases - A Review. In *Beyond Databases, Architectures and Structures, BDAS*, pages 141–152, 2017.

[33] F. Raab, W. Kohler, and A. Shah. Overview of the TPC-C Benchmark. `http://www.tpc.org/tpcc//detail.asp`. Accessed: 2018-11-15.

[34] M. Seltzer. Beyond Relational Databases. *Commun. ACM*, 51(7):52–58, July 2008.

[35] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang. LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation. In *OSDI*, pages 69–87, Carlsbad, CA, 2018. USENIX Association.

[36] M. Smolinski. Impact of Storage Space Configuration on Transaction Processing Performance for Relational Database in PostgreSQL. In *Beyond Databases, Architectures and Structures, BDAS*, pages 157–167, 2018.

[37] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik. C-Store: A Column-oriented DBMS. In *VLDB, Trondheim, Norway, August 30 - September 2, 2005*, pages 553–564, 2005.

[38] K. Tsakalozos, V. Verroios, M. Roussopoulos, and A. Delis. Live VM Migration Under Time-Constraints in Share-Nothing IaaS-Clouds. *IEEE Trans. Parallel Distrib. Syst.*, 28(8):2285–2298, 2017.

[39] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang. Automatic Database Management System Tuning Through Large-scale Machine Learning. In *SIGMOD*, pages 1009–1024, New York, NY, USA, 2017. ACM.

[40] S. Wang, C. Li, H. Hoffmann, S. Lu, W. Sentosa, and A. I. Kistijantoro. Understanding and Auto-Adjusting Performance-Sensitive Configurations. In *ASPLOS*, pages 154–168, New York, NY, USA, 2018. ACM.

[41] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *OSDI*, Dec. 2002.

[42] E. Zamanian, C. Binnig, T. Harris, and T. Kraska. The End of a Myth: Distributed Transactions Can Scale. *Proc. VLDB Endow.*, 10(6):685–696, Feb. 2017.