Enterprise Database Applications and the Cloud: A Difficult Road Ahead

Michael Stonebraker
MIT CSAIL
stonebraker@csail.mit.edu

Andrew Pavlo
Carnegie Mellon University
pavlo@cs.cmu.edu

Rebecca Taft
MIT CSAIL
rytaft@mit.edu

Michael L. Brodie
MIT CSAIL
mlbrodie@mit.edu

***Abstract*—There is considerable interest in moving DBMS applications from inside enterprise data centers to the cloud, both to reduce cost and to increase flexibility and elasticity. Some of these applications are "green field" projects (i.e., new applications); others are existing legacy systems that must be migrated to the cloud. In another dimension, some are decision support applications while others are update-oriented. In this paper, we discuss the technical and political challenges that these various enterprise applications face when considering cloud deployment. In addition, a requirement for quality-of-service (QoS) guarantees will generate additional disruptive issues. In some circumstances, achieving good DBMS performance on current cloud architectures and future hardware technologies will be non-trivial. In summary, there is a difficult road ahead for enterprise database applications.**

I. Introduction

In this paper, we consider the migration of enterprise DBMS applications to the cloud. To discuss these applications, we must first classify them into a few constituent types. The DBMS market is much different today than it was in the early 1980s. Back then, there was only one target market for DBMSs, namely business data processing. Now we can characterize a much larger DBMS market into three approximately equal size segments:

Modern OLTP: These applications include traditional on-line transaction processing (OLTP), as well non-traditional use cases, such as maintaining the state of Internet games and real-time placement of advertising on web pages. Other notable applications include high-velocity use cases whereby an incoming message stream (either from humans, machines or the Internet of things) must be processed transactionally and some sort of state maintained. In modern OLTP applications, ACID transactions are omni-present and high availability is a requirement. Databases are typically in the Gbytes-to-Tbytes size range and transaction volumes are often high. These workloads are characterized by transactions that update a small number of records at a time [20], often interspersed with small-to-medium size queries.

Data Warehouses: Every enterprise keeps a historical database of customer facing data (e.g., sales, customers, shipments). Data warehouses usually ingest new data from OLTP systems or sensor networks after going through a data transformation/cleaning procedure. This is typically accomplished using an "Extract, Transform, and Load" tool that streams or bulk loads data from the front-end DBMSs into the warehouse. The sizes of these databases can be quite large (i.e., Tbytes to Pbytes), and the workload primarily consists of on-line analytical processing (OLAP) queries (i.e., business intelligence) that are more complex than queries in OLTP applications. Once the data is loaded, it is rarely-to-never updated.

Everything-Else: This class encompasses key-value stores (a portion of the NoSQL market [5] that are good at simple applications), document-oriented systems (another part of the NoSQL market), array stores (good at complex analytics), graph stores (good at storing social networks), and distributed computing platforms (good at "embarrassingly parallel" analytic applications).

There is little we can say about the everything-else market, since it is so diverse. As such, we focus on the other two kinds of enterprise applications in this paper.

In Section II, we begin with a description of what we mean by the cloud. Of course, for enterprise applications that we discuss here, the underlying DBMS must be deployed on cloud hardware, a topic we turn to in Section III. We also discuss the performance headaches that the virtualization of I/O in most public cloud systems will cause for most current DBMSs. In the process, we also consider other challenges that future DBMS deployments will encounter that will influence their performance in cloud environments. Then we turn in Section IV to the difficulties that enterprise DBMS applications will face when migrating to the cloud. This discussion centers on the migration cost of existing legacy systems as well as making quality of service guarantees. Finally, we explore how future storage technology will influence DBMS architecture and possibly cause enterprises to consider a different type of DBMS than currently deployed, which will cause additional migration issues.

II. The Meaning of the Cloud

In this section, we discuss what we mean by "the cloud" to set a context for the remainder of the paper.

Some of the authors remember a time (circa 1975) when we interacted with a computer using a dumb terminal and the OS multiplexed the server's resources among a collection of clients. Long ago, these were called time-sharing systems. We are now rapidly heading back to this architecture, under the sobriquet of "cloud computing".

This transformation is driven by economies of scale. In a typical enterprise setting, the traditional methodology is to approve a budget for a new application that includes the cost of dedicated hardware to host it. Such hardware is invariably

sized for the peak load that is expected over the life of the application, which is usually 10× the average load. Under this model, service level agreements (SLAs) with the people that use or manage the application are always met. But the dedicated hardware is running at ∼10% capacity since it can only be used by this one application.

Once the application is deployed in production, one of the authors (who spent a large portion of their career in a large enterprise) estimates that the typical downstream operational budget is:

- 10% – Hardware Maintenance and Upgrades
- 10% – Software Maintenance and Upgrades
- 80% – Systems and Database Administration (labor)

This budget is repeated for every application in the enterprise, which often number in the 1000s or even 10000s. The end result is a large number of "computing silos" running on dedicated and often heterogeneous hardware.

The obvious solution to reducing costs is to combine silos together on common (larger) hardware. The biggest benefit is the reduction in administration cost, since database and systems administrators can be shared over multiple applications. Every enterprise that we know of is moving toward this shared (cloud-like) model to cut costs.

Such time-sharing systems can be deployed inside the firewall as a private cloud or in a public server farm (e.g., Amazon, Rackspace). Amazon claims that it can deploy a server at 10% of the cost of a typical enterprise customer [7]. As one would expect, some of these savings come from hardware purchasing leverage. More significant cost savings, however, come from the automation and streamlining of system administration. In addition, Amazon constructs a new data center several times a year, so they are more efficient and technically adept at this task than companies that may build a new data center only once a decade.

Although the cloud (public or private) can mean the rental of physical machines, it usually entails time-sharing of virtual resources. That is, an application is running on a virtual CPU connected to a virtual disk and using virtual networking. In other words, the cloud usually means "virtual everything." Thus, in the next section we discuss the implications of this virtualization on DBMSs.

III. Issues with DBMSs in the Cloud

In order to discuss DBMS issues with the cloud, we first need to discuss the relationship of the DBMS with modern distributed file systems that are often used in cloud-based systems. We discuss how DBMSs' use of file systems have improved in recent years, and the impact of newer I/O virtualization and data replication technologies.

A. I/O Through the File System

One of the authors wrote a paper in 1981 [18] discussing the shortcomings of then file system technology from the DBMS perspective. At that time, no serious DBMS used the file system for data storage, preferring instead to allocate a "raw device" and build its own storage system. At this point, all DBMSs that we are aware of that were written in the last decade use the file system as their exclusive storage option. Moreover, most legacy DBMSs have also been converted to use a file system as their preferred storage mechanism. This change has been facilitated by the extension of file systems with the following features:

Direct Read to User Space: The old mechanism of reading disk blocks into kernel buffers and then copying the blocks to user space was too slow. Not only does it require double copying, but also it generates two buffer pools (i.e., one in the DBMS address space and one in kernel space), an obviously inefficient architecture. Now DBMSs can retrieve data directly from the file system with little overhead from the OS.

Asynchronous I/O: All notable DBMSs have moved from a process-per-user architecture to a multi-threaded server architecture. A process-per-user system consumes too much memory and its performance is hindered by slow process switches. A server architecture with several active threads does not block when it issues an I/O request on behalf of one of its threads, but instead switches to work on another thread.

Large-space Allocations: It is now straightforward to request a large block of contiguous space from the file system. The old way of requesting small blocks one-by-one is not required any more. Anecdotal evidence suggests that a DBMS-specific storage system improves performance by ∼10% over using an OS-provided file system. For this level of performance gain, it is not worthwhile to build a raw-device storage system.

Just as these issues were obstacles to DBMSs three decades ago, there are now several new technologies found in cloud-oriented file systems that are problematic to DBMSs. We now discuss these new problems.

B. File System Distribution

File system-level distribution, known as I/O virtualization, allows a file system to span multiple nodes and distribute data across a cluster of machines. Such distribution is provided by services like GFS [6] and HDFS [17]. Running on top of such a file system is not likely to find acceptance among performance conscious DBMSs. Essentially all multi-node DBMSs "send the query to the data" rather than "bring the data to the query." This is because there is significantly less network traffic if the query runs at the location of the data. Achieving this is not trivial or even possible in these modern distributed file systems.

Virtualizing the CPU is just as onerous because it hides the database's physical location from the DBMS, thereby virtualizing the I/O. Put differently, virtualizing the CPU has been shown to work well on compute-oriented tasks, where I/O performance is not a factor. But when applied to DBMSs, it causes I/O virtualization, which results in the same performance degradation as a distributed file system. In Section III-D we report some simple experiments that show the magnitude of this performance degradation.

C. Replication

Finally, we turn to file system-level replication, as implemented in services like HDFS [17]. There are several reasons why this service is not likely to find DBMS acceptance. Foremost is that some DBMSs, such as MemSQL [1], SQLFire [3],

HANA [10], and VoltDB, are main memory DBMSs, and thus do not use the file system as the primary storage location for databases. These systems have to provide replication in some other way.

There are two reasonable methods that OLTP DBMSs use to replicate data. With active-passive replication, the DBMS writes a log at a primary node containing the before image and after image of each update. Then, the log is moved over the network to roll the data forward at a secondary node. In effect the primary node does all the heavy lifting, while the secondary node simply keeps up with primary node updates. This approach is popular in commercial DBMS products, but would require some adaptation to work with a file system replication scheme. If the file system also supports transactions, then this facility can be turned off in the DBMS and the file system-supplied one used instead. Since few file systems support both replication and transactions, this active-passive approach will be problematic, and DBMSs are unlikely to use it.

More troubling is the other popular active-active DBMS replication mechanism, where the DBMS executes a single transaction multiple times, once at each replica. This is the approach used by both VoltDB and SolidDB [12]. A recent paper [13] suggests there are many use cases for which active-active is faster than active-passive. Obviously, an active-active scheme is incompatible with file system replication.

In the data warehouse market, there are additional concerns over file system-level replication. First, OLAP-style queries are I/O intensive, and thus "sending the query to the data" is likely to substantially outperform bringing the data to the query. In addition, some data warehouse DBMSs, such as Vertica [8], store replicas in different sort orders so that the query executor can use the replica that offers the best performance for each individual query. For example, one replica might store an "employee" table sorted on the person's name while a second replica is sorted on the person's department. The DBMS will automatically route queries that filter on the employee department to the second replica because it will run faster there. Clearly, such logical replicas cannot be maintained by the file system.

D. Experimental Results

To better understand the value of local storage and logical replicas, we now present a series of simple experiments. These experiments evaluate the performance of a DBMS running on different environments that are representative of the options available for those considering cloud-based deployments.

We first compared local block-oriented I/O to remote I/O. Across a LAN, I/O is about a factor of 50× slower than local I/O. However, most DBMSs are CPU intensive, and will show a smaller performance difference.

To identify the difference between local and remote I/O in a DBMS context, we ran a simple workload using the Vertica DBMS. We executed two queries on a pair of tables, where one is 10 Gbytes and the other is 50 Mbytes. One query filtered the larger table and computed an aggregate. The second filtered the larger table, joined it to the smaller table, and then computed an aggregate on the joined result. These queries are typical of the workloads found in decision support systems. With a cold cache, the queries' execution times were a factor of 12× slower when using a remote (i.e., virtualized) disk deployment, versus a local disk. Of course, if the cache is warm and the tables are smaller, then the difference decreases, but in our experiments it was never smaller than a factor of 2×. The performance will obviously vary depending on local circumstances and database sizes, but the difference is likely to be an order of magnitude.

Lastly, we also ran the above queries with a pair of bit-wise identical replicas and compared that result to using the Vertica logical replica facility. This facility allows the DBMS to maintain replicas that are in different sort orders so the query optimizer can choose the replica that offers the best performance. Vertica ran our queries 1.8× faster with two replicas in different sort orders relative to two replicas in the same sort order. If the file system provides both distribution and replication, then the performance degradation will be at least an order of magnitude. Thus, such a file system facility is not likely to gain acceptance at this cost.

E. Summary

DBMSs have been re-engineered in the last 20 years to use local file systems more efficiently. When considered for cloud deployment, this means either renting dedicated hardware (which we expect will decline in popularity) or using the public cloud with file systems like HDFS or Amazon's EBS. As noted above, these provide I/O virtualization that have an order of magnitude or more performance degradation. They also provide file system replication, which causes additional performance issues. In general, one should expect a DBMS running on the public cloud to have an order of magnitude worse performance, compared to a non-virtualized I/O system. There are several ways to look at this issue.

First, there is a trend toward running main memory DBMSs. At least for OLTP applications, the size of the database is increasing slower than the cost of main memory is declining. In addition, [14] makes the case that even a large retailer like Amazon is a candidate for main memory database deployment. Main memory DBMSs are being marketed by SAP [10], Oracle [21], Microsoft [9] and a variety of startups, including VoltDB [4] and MemSQL [1]. If an enterprise DBMS application is deployed on a main memory DBMS or is contemplating moving to a main memory DBMS, then none of the above discussion applies as such DBMSs do not use the file system for data storage.

Second, if you are running an enterprise DBMS application on a SAN storage device, such as sold by EMC or Hitachi, then you are already running on virtualized I/O and paying the performance penalty mentioned above. In this case, you can move from one virtualized environment to another, and it will be a lateral move, probably with no performance penalty.

Third, if you are running Hadoop, then you are already using HDFS for storage. It is virtualized, so there will be no additional penalty. One should only note that Hadoop applications typically run an order of magnitude slower than the same application run on a parallel DBMS [16].

Lastly, if you are running a parallel disk-oriented DBMS, then it will assuredly expect storage that is not virtualized. If

you move such a DBMS onto virtualized storage, then you will see dramatic performance degradation. As such, whether or not your enterprise application will see this degradation from cloud deployment will depend on which of the above categories your DBMS fits into.

IV. Issues with Enterprise Cloud Applications

Besides a cost reduction mentioned earlier, one of the potential benefits regularly touted for cloud computing is elasticity. In other words, if the DBMS (or any other application) needs more resources, then it can automatically allocate additional machines and run in parallel over them. This is likely to remain elusive for the majority of applications. To understand why this is the case, we consider three kinds of applications in this section and discuss their possible movement to the cloud. The first group is "one-shot" applications, which are distinguished from "production enterprise applications" that have been running for longer periods of time or "green field" applications that are not yet written.

A. "One-Shot" Applications

This category encompasses much of the workload currently running on the public cloud. These include proof-of-concept prototypes, ad hoc analytics, and other non-mission critical operations. In order for them to be elastic, they must be coded as parallel programs, either for running on a parallel DBMS or on a distributed computing framework.

Most of the traditional DBMSs used today are unable to run on multiple nodes out-of-the-box. For the ones that do, few are able to easily scale resources up or down without significant downtime. Several NoSQL vendors claim this ability but they achieve this by forgoing all transactional guarantees of traditional DBMSs [5].

Writing an application as a standalone parallel program is just as difficult, as few of us are skilled parallel algorithm developers. If one uses a distributed computing framework, such as Hadoop or Spark [22], there are still other challenges. Techniques for parallelizing computations may entail considerable effort. For example, efficiently parallelizing matrix multiply requires a block cyclic layout of the matrices and clever parallel code.

Hence, there are significant challenges to achieving this elasticity goal. The one use case that can easily be made elastic is "embarrassingly parallel" applications. For example, suppose one wants to search a document collection for all the instances that have "MIT" followed within five words by "CMU." In this case, we can divide the documents among multiple servers, run the same code on each one, and then perform a single roll-up at the end. Such applications are trivial to make elastic, but unfortunately few applications are actually like this. In our analysis of MIT's Lincoln Labs data-intensive applications in [19], we found that only 5% of their workload fits this categorization. We suspect that these findings are similar to other organizations.

B. Production Enterprise Applications

Service-level agreements (SLAs) are critical to enterprise computing. A business will only agree to participate in a shared cloud if its SLAs can be guaranteed. But there are several problems with guaranteeing SLAs in these environments. First, if all resources are virtualized, then the variance in response times will assuredly increase, as well as perhaps the average response time. Obviously, I/O virtualization will compromise performance in the DBMS. In addition, the cloud operator hopes that load spikes in different applications occur at different times so that a cloud can be sized at $\sim$10% of the sum of the individual silos.

In most enterprises, however, application usages are highly correlated and cyclic. For example, humans usually call customer support during business hours. Since they also invoke web services with essentially the same skewed frequencies, it is imperative that the cloud platform co-locates applications in such a way to minimize the impact of this correlation. Furthermore, the administrator will also want to co-locate applications that have similar maintenance windows. Otherwise, it will not be possible to perform upgrades because of uptime requirements.

Another issue in these enterprise applications is their long-term support and maintenance. Any application that requires legacy DBMS software will not be able to move to the cloud. For example, IBM's IMS DBMS only runs on IBM "big iron" machines. Similarly, any application that requires specialized hardware, such as Netezza or Teradata, will also be unmovable. But even systems written for commodity hardware could have problems. For example, an application may be coded to run on exactly two large DBMS nodes with the logic to manually allocate resources between them embedded in the application's code. A cloud provider with less powerful nodes will not be able to easily run this application. Legacy applications may have any number of these unexpected issues that will require significant recoding. Any requirement for major changes in the application will reduce or eliminate the benefit of cloud deployment. As such, we expect moving legacy applications to the cloud to be a drawn out affair.

Even if an application is able to move to a cloud platform, an enterprise is more likely to move them to an internal (private) cloud long before they will use a public one. This is because of regulatory and security restrictions on enterprises that prevent deployments on outside resources. For example, most telcos are effectively precluded from using public clouds. Second, many companies have policies in place that dictate that their intellectual property (i.e., their data) must stay inside their firewall. This is partly because there is a widespread belief (almost certainly unfounded) among these companies that the cloud is less secure than their own data centers.

Beyond legal barriers, we believe that elastic scale-out is unlikely to be achieved by most (or possibly none) of the legacy applications. First, the DBMS must be elastic, as discussed above. Unless the schema is designed carefully, it is likely that sharding a production single node database onto multiple nodes will result in worse performance than observed previously [15]. Such behavior will occur if most of the transactions move from being single-sited to multi-sited. Rewriting the application to be "shard-friendly" will likely be a daunting task.

C. Green Field Applications

To take full advantage of cloud computing resources, new applications will have to be built according to certain design principles. We now discuss two of them that we believe are relevant to enterprise applications and DBMSs:

Self-Service Provisioning: This means that the system can adjust its resource needs on the fly, as it experiences shifts in load, whether due to transient local spikes, time-cyclic spikes, or longer term changes. Although it is possible to code an application to perform automated provisioning, this will require a substantial amount of new logic. This means that enterprise applications and DBMSs will have to perform monitoring of their resource usage and then employ a predictive model to anticipate changes in load. Such modeling is the domain of "data scientists" and is currently not well understood by many developers.

Elasticity: Self-service provisioning will only work if the application is coded natively to support elastic scale out. This was discussed above, and entails careful coding and choice of subsystems by application and DBMS developers.

D. Cloud SLAs on Hardware Resources

An application can only provide SLAs to its customers if it can get guarantees on allocations from the underlying cloud platform. But current guarantees on CPU and disk resources are just approximations. Furthermore, the state of the art in guaranteeing network bandwidth is in its infancy and we do not expect emerging software-defined network technologies to improve this situation any time soon.

Even main memory virtualization will be a big challenge. Migrating a main memory DBMS installation from one server to another requires moving the entire main memory state of the system, which can be exceedingly large. In addition, moving resources may turn single-node transactions into multi-node ones, which are much less efficient [15]. Optimizing such a workload over a collection of nodes at the cloud infrastructure level seems impossible. At the DBMS level, however, it is a challenge but perhaps doable. But this requires that the DBMS understands the physical resources, which is not possible in a virtualized environment.

In order to optimize data access, to permit sending the "query to the data", and to manage copies for data virtualization, a cloud DBMS must know the location of the data whether in-memory, in directly-attached storage, or distributed to minimize network traffic for database access. Just how cloud DBMSs, OSs, hypervisors, and file systems will cooperate to optimize both computation and data management at scale in the cloud remains to be seen.

E. Summary

The challenges of moving legacy applications to a cloud deployment may lower the return on investment for enterprises. In addition to this, we expect enterprise cloud deployments to proceed much more rapidly on private clouds than on public ones. And until cloud services get much better, SLAs will be difficult for an application to meet and may hinder adoption. As such, we foresee a long and difficult road ahead.

V. Future Hardware Trends

There are two coming hardware advancements that are likely to cause substantial rewrites of current DBMSs and other system software: (1) many-core CPU architectures and (2) non-volatile memory storage. As we will now discuss, these may help or hinder moving enterprise applications to the cloud.

A. Many-Core CPUs

All DBMSs written more than a decade ago implement a multi-threaded server architecture. This approach works well with a single-core CPU. But now DBMSs are deployed on multi-core systems, with promises of "many-core" systems ($\geq$1000 cores) within the next decade.

Such many-core architectures will be a significant challenge to DBMSs because of shared data structures, such as:

1. Indexes,
2. Lock tables (if dynamic locking is implemented),
3. Temporary space to be malloc-ed or freed,
4. Temporary buffers for sorting,
5. Resource information data structures.

The most notable of these are indexes, in particular the pervasive B-tree, because they are found in every DBMS. Using standard locking techniques for B-trees on a multi-core system greatly impairs parallelism, and thus DBMSs use special case concurrency control measures for these indexes. One popular technique is to latch a B-tree block until the thread is sure that it will not be changing the block, at which point it releases the latch. As the thread descends to a block at the next level of the tree, it repeats the process, and this strategy is known as "latch crabbing." It will tend to cause serialization of DBMS threads. There are, of course, other options, e.g. [11], but each has its own issues with parallelism.

We have a strong suspicion that latch contention will kill the performance of traditional DBMSs on many-core systems. As a result, we predict that all traditional DBMSs will have to be rewritten to remove such impediments. Alternatively, the DBMS could use lock-free data structures, as in MemSQL or Hekaton [9], or use a single-threaded execution engine, as in VoltDB and Redis [2]. It is also likely that this rewrite will be a daunting challenge to OSs, file systems, application servers, and other pieces of system software. Many-core may well impact enterprise applications that use shared data structures in much the same way.

As such, many-core CPUs are likely to cause substantial rewrites of applications, DBMSs, file systems, OSs, and other pieces of system software. Such rewrites may well distract software developers from tackling cloud issues, since it is well known that throwing too many balls up in the air at once is usually a disaster. Alternatively, the re-factoring of systems for many-core CPUs may also improve their ability to operate in elastic cloud environments.

B. Storage Advancements

There are several storage trends that will heavily impact future DBMSs. The first is that the plummeting cost of main memory will allow a greater number of DBMS applications to

be main memory-resident. We expect most (if not all) modern OLTP applications to fit entirely in main memory within a few years. Over time, a substantial fraction of the DBMS market will move to main memory-only storage. This trend will render the file system irrelevant to DBMSs, except as a storage place for checkpoints and long-term archival.

Flash is currently too slow to be considered as a main memory replacement. Hence, administrators are deploying flash as a disk replacement for those DBMS applications that are too large to fit in main memory but have performance requirements. The most notable of these is Facebook. In effect, this is replacing one block-structured device by a second block-structured device that is faster but more expensive. As such, we do not see flash fundamentally altering DBMS architectures.

There are several possible technologies that may prove viable later this decade that have the possibility of replacing DRAM as the primary storage for DBMS data. These include memristors, phase-change memory, and spintronic devices. The expectation is that these technologies will be able to read and write data at close to DRAM speeds, but at a fraction of the cost and with much higher storage capacities. This technology will extend the reach of main memory DBMSs into the 1–10 Pbytes range, further enlarging the main memory DBMS market. If successful, we expect block-structured DBMSs to disappear during this decade as these devices lower the cost of byte-addressable persistent storage. In other words, all modern OLTP applications and all but the largest data warehouses will use this technology.

An enterprise application on a main memory DBMS will find an easier time with cloud migration, and this may alter the economic equation in favor of cloud deployment.

C. Summary

Between many-core CPUs and non-volatile RAM, we expect most current system software will have to be rewritten, causing possible extensive marketplace disruption as traditional DBMSs make way for new technology. This may accelerate or slow down cloud migration. In any case, the road ahead will again be difficult.

VI. Conclusion

Although there have been significant improvements, we still see considerable turmoil between system software and DBMSs. The cloud, many-core CPUs, and future storage technology are likely to make the legacy DBMS implementations in the marketplace today obsolete. These issues will require a nearly complete rewrite of current DBMSs, and other architectural ideas may well replace the underpinnings of current systems. In addition, we see the cloud as a major disruptive force in enterprise computing. Specifically, we expect that all current enterprise applications will have to be largely rewritten to take full advantage of the cloud. In addition, how to get the various pieces of cloud infrastructure, including DBMSs, to work harmoniously together in guaranteeing SLAs is a difficult problem. Even when the technology issues are resolved, the anticipated DBMS disruption will have to be funded by demand from new applications. Industry faces the age-old issue of legacy application and database migration.

References

[1] MemSQL. http://www.memsql.com.

[2] Redis. http://redis.io.

[3] VMware vFabric SQLFire. http://www.vmware.com/go/sqlfire.

[4] VoltDB. http://www.voltdb.com.

[5] R. Cattell. Scalable SQL and NoSQL data stores. *SIGMOD Rec.*, 39:12–27, 2011.

[6] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. SOSP, pages 29–43, 2003.

[7] J. Hamilton. Internet-scale datacenter economics: Where the costs & opportunities lie. Presented at HPTS, 2011. http://mvdirona.com/jrh/talksandpapers/JamesHamilton_HPTS2011.pdf.

[8] A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandiver, L. Doshi, and C. Bear. The Vertica analytic database: C-Store 7 years later. *Proc. VLDB Endow.*, 5(12):1790–1801, Aug. 2012.

[9] P.-A. Larson, S. Blanas, C. Diaconu, C. Freedman, J. M. Patel, and M. Zwilling. High-performance concurrency control mechanisms for main-memory databases. *VLDB*, 5(4):298–309, Dec. 2011.

[10] J. Lee, M. Muehle, N. May, F. Faerber, V. S. H. Plattner, J. Krueger, and M. Grund. High-performance transaction processing in SAP HANA. *IEEE Data Eng. Bull.*, 36(2):28–33, 2013.

[11] P. L. Lehman and S. B. Yao. Efficient locking for concurrent operations on b-trees. *ACM Trans. Database Syst.*, 6(4):650–670, Dec. 1981.

[12] J. Lindstrom, V. Raatikka, J. Ruuth, P. Soini, and K. Vakkila. IBM solidDB: In-memory database optimized for extreme speed and availability. *IEEE Data Eng. Bull.*, 36(2):14–20, 2013.

[13] N. Malviya, A. Weisberg, S. Madden, and M. Stonebraker. Recovery algorithms for in-memory OLTP databases. *In Submission*, 2013.

[14] T. Mühlbauer, W. Rödiger, A. Reiser, A. Kemper, and T. Neumann. ScyPer: elastic olap throughput on transactional data. DanaC, pages 11–15, 2013.

[15] A. Pavlo, C. Curino, and S. Zdonik. Skew-aware automatic database partitioning in shared-nothing, parallel OLTP systems. In *SIGMOD*, pages 61–72, 2012.

[16] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *SIGMOD*, pages 165–178, 2009.

[17] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. MSST, pages 1–10, 2010.

[18] M. Stonebraker. Operating system support for database management. *ACM*, 24(7):412–418, 1981.

[19] M. Stonebraker and J. Kepner. Possible Hadoop Trajectories – blog@CACM (2012). http://cacm.acm.org/blogs/blog-cacm/149074.

[20] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The end of an architectural era: (it's time for a complete rewrite). In *VLDB*, pages 1150–1160, 2007.

[21] T. Team. In-memory data management for consumer transactions the timesten approach. SIGMOD, pages 528–529, 1999.

[22] 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. NSDI, 2012.