In-memory Databases – Challenges and Opportunities

From Software and Hardware Perspectives

Kian-Lee Tan, Qingchao Cai, Beng Chin Ooi*,
Weng-Fai Wong, Chang Yao, Hao Zhang
National University of Singapore
{tankl, caiqc, ooibc, wongwf, yaochang, zhangh}@comp.nus.edu.sg

ABSTRACT

The increase in the capacity of main memory coupled with the decrease in cost has fueled research in and development of in-memory databases. In recent years, the emergence of new hardware has further given rise to new challenges which have attracted a lot of attention from the research community. In particular, it is widely accepted that hardware solutions can provide promising alternatives for realizing the full potential of in-memory systems. Here, we argue that naive adoption of hardware solutions does not guarantee superior performance over software solutions, and identify problems in such hardware solutions that limit their performance. We also highlight the primary challenges faced by in-memory databases, and summarize their potential solutions, from both software and hardware perspectives.

1. INTRODUCTION

Despite the dominance of disk-based data processing systems for Big Data [17], in-memory computing is recently gaining traction rapidly. This is fueled by several contributing factors: the increased capacity of main memory, the low cost of DRAM, and more importantly, the orders of magnitude higher main memory bandwidth than the most advanced disk- or flash-based storage. While in-memory databases have been studied since the 80s, recent advances in hardware technology have re-generated interests in hosting the entirety of the database in memory in order to provide faster accesses and real-time analytics. A comprehensive survey on in-memory data management and processing can be found in [33].

However, in-memory databases not only embrace opportunities with the emergence of new technology, they also face challenges and problems that are non-trivial to be addressed. Simply replacing the storage layer of a traditional disk-based database with memory will not satisfy the real-time performance requirements because of the retention of the clumsy components from traditional database systems, such as the buffer manager, latching, locking and logging [8, 34]. Other sources of overhead from pointer-chasing, cache-un-friendly structures, transaction isolation and syscalls, further exacerbate the performance problems. In addition to the classical storage layer performance issues, in-memory data-bases are increasingly hitting the communication and concurrency bottlenecks [35, 29].

A significant amount of researches have been done to address these challenges, through the design of new algorithms/data structures on top of existing software stack, from the aspects of in-memory data placement [27], parallelism [7], efficient logging [21], concurrency control [29, 31], etc. Nevertheless, advances in hardware are fast changing the commodity processor scene. The availability of technologies such as NUMA architecture [20], SIMD instructions [30], RDMA networking, hardware transactional memory (HTM) [16], non-volatile memory (NVM), and on-chip GPUs [5], FPGAs and other hardware accelerators, can potentially provide better performance with low overhead [16, 11, 9, 19].

In this paper, we argue that these are by themselves not magic bullets. Each hardware solution has its own limitations and idiosyncrasies. Without new software optimization techniques, and extensive tuning, it will be hard to fully realize the potentials that the technology brings. We highlight some of these issues, and identify promising research directions that our community can contribute for both OLTP and OLAP systems.

The remainder of the paper is structured as follows. Section 2 discusses the challenges faced by in-memory databases, and Section 3 surveys existing approaches to address these challenges from both software and hardware perspectives. In Section 4, we point out the potential problems coming from hardware solutions. We propose some open research directions in Section 5, and conclude in Section 6.

*Corresponding author.

2. CHALLENGES FOR IN-MEMORY DATABASES

2.1 Parallelism

In general, there are three levels of parallelism, i.e., data-level parallelism (e.g., bit-parallel, SIMD), shared-memory scale-up parallelism (e.g., thread/process) and shared-nothing scale-out parallelism (e.g., distributed computation). Ideally, we would like to achieve linear scalability as the computation resources increase. This, however, is non-trivial and requires considerable tuning and well-designed algorithms. The fact that all these three levels of parallelism have been deployed in a wide variety of combinations further compounds the problem.

Research challenge 1: How can OLTP and OLAP systems benefit from the wide variety of parallelism paradigms present in today's systems?

2.2 Concurrency Control

An efficient concurrency control protocol is necessary and important in order to ensure the atomicity and isolation properties, and not to offset the benefit derived from parallelism. With the increasing number of machines that can be deployed in a cluster and the increasing number of CPU cores in each machine, it is not uncommon that more threads/processes will run in parallel, which dramatically increase the complexity for concurrency control. Surprisingly, current concurrency control algorithms fail to scale beyond 1024 cores [32].

Research challenge 2: We need truly scalable concurrency control.

2.3 Communication

Network communication is incurred for a variety of critical operations: data replication for fault tolerance, information exchange for coordination, data transmission for data sharing or load balancing, and so on. The limited size of main memory of a single server, in contrast to the big volume of data, exacerbates the network communication requirement. However, the data access latency gap between main memory and network is huge, making communication efficiency important to the overall performance.

Research challenge 3: How can we bridge the communication gap that is growing both in magnitude and diversity?

2.4 Storage

Even though in-memory databases store all the data in the main memory, the data should also be persisted to non-volatile storage for durability and fault tolerance. In traditional disk-based databases, this is achieved by logging each data update to a disk-resident write-ahead log. Logging to disk, however, is prohibitively expensive in the context of in-memory databases due to the extremely slow disk access, in contrast to the fast memory access.

Research challenge 4: How can we scale in-memory databases to exploit expanding NVM capacities effectively?

3. SOFTWARE AND HARDWARE SOLUTIONS

We shall summarize the potential solutions to these challenges, from both software and hardware perspectives in this section.

3.1 Parallelism

On the software level, we can impose different computation models or techniques to realize parallelism of different granularities, namely, fine-grained, coarse-grained and embarrassing parallelism. Multi-threaded programs are a common case that utilizes the multi-core architecture to scale up [29], while distributed computing takes advantage of the resources from hundreds or thousands of servers to scale out the computing/storage capability.

NUMA architecture is proposed to solve the data starvation problem in modern CPUs, by eliminating the coordination among different processors when accessing its local memory, and thus fully exerting the accumulated power from multiple processors. SIMD provides an easier alternative to achieve data-level parallelism, for its capability to operate on multiple data objects in one instruction [30]. In addition, bit-parallel algorithms are proposed to fully unleash the intra-cycle parallelism of modern CPUs, by packing multiple data values into one register, which can be processed in parallel [7]. The introduction of on-chip hardware accelerators such as GPUs, FPGAs, and other heterogeneous cores presents another challenge, as their usage does not just require new code but in fact new algorithms.

3.2 Concurrency Control

Following the canonical categorization in [32], concurrency control protocol can either be lock-based or timestamp-based, from the software perspective. A significant amount of research is trying to improve its efficiency, such as Very Lightweight Locking (VLL) [25] for lock-based approaches, and some MVCC protocols [13, 23] for timestamp-based approaches. High overhead from 2PC for distributed transactions are alleviated by some 2PC-avoidance partition schemes [10] and speculative concurrency strategies [24, 14].

The ideal case for concurrency control is that all transactions are executed in parallel without any concurrency control protocol overhead, which is usually very hard to achieve in practice. It has been shown that HTM-based timestamp-based concurrency control performs quite close to the ideal and hence simply cannot be ignored [16].

3.3 Communication

To improve the network performance, there are mainly two main approaches.

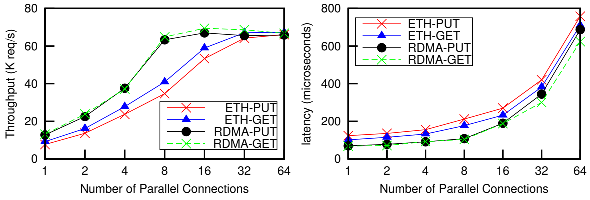
Minimizing communication. Data locality is key to minimizing communication overhead. With good data locality, there will be less frequent access to remote data. This can be achieved by a good partitioning algorithm [24], or an efficient query routing mechanism to push computation close to the data [14]. The data locality can only be considered from the software perspective, since the strategy is usually application-specific.

Improving the communication efficiency. From a software perspective, the network communication efficiency can be improved by object coalescing or batch communication [35, 2], kernel-bypass networking (e.g., netmap [26]), data-plane operating systems (e.g., IX [1]), and other techniques. On the other hand, RDMA boosts the networking performance from the hardware perspective, by enabling zero-copy and one-sided networking [22, 2]. In particular, RDMA enables user-level data transmission without involving the kernel or the CPU, thus achieving low latency and high throughput on the hardware level. It is free from the complexities and problems (e.g., lack of congestion control, retransmission) imposed by software solutions.

3.4 Storage

To alleviate durability overhead for in-memory databases, recent studies [21] proposed to use *command logging*, which logs only operations instead of the updated data, and combines with *group commit* to further reduce the number of loggings. However, there still exists a fundamental design trade-off between the high durability and logging overhead. If only a small number of transactions are committed each time, then the logging overhead may still substantially affect the transaction throughput due to its orders of magnitude higher latency than the execution time of transactions. On the other hand, accumulating a large number of transactions for *group commit* can lead to more data lost upon failure.

The emergence of NVM, including flash, PCM or STT-RAM, offers a new promising alternative for data storage. These non-volatile technologies have been touted as DRAM replacements [6]. However, if we look deeper, things are not as rosy. The denser form of flash memory, i.e., NAND flash, works in units of pages. PCM and STT-RAM are bit addressable but PCM suffers from inherent endurance and write disturbance issues [15] while STT-RAM suffers from inherently non-zero write failure probabililty [18]. Furthermore, all three classes of NVM have asymmetric read and write latencies [18].



(a) Throughput

(b) Latency

Figure 1: Redis Performance with Ethernet and RDMA

4. POTENTIAL PROBLEMS WITH HARDWARE SOLUTIONS

The challenges faced by in-memory databases can be addressed from both software and hardware perspectives. Software solutions are constrained by the underlying software stack, and increasingly hitting the performance bottleneck [3], while hardware solutions can boost the performance from the lowest underlying layers (i.e., transistors, circuits), which usually does not introduce much overhead.

However, naïve adoption of hardware solutions does not guarantee superior performance over software solutions. For example, simply using RDMA does not necessarily improve the performance greatly. In Figure 1, we show the performance difference for Redis [28], with its default Ethernet network (1 Gbps), and simple replacement of RDMA network (40 Gbps), as an illustration for the unexpected behavior by only relying on hardware. Specifically, we can see that RDMA does not improve its performance significantly in terms of both throughput and latency, even though RDMA throughput is 40× faster than Ethernet. With enough connections (i.e., more than 32), the performance is almost the same, losing the potential high performance advantage of RDMA networking. We summarize some issues that arise from hardware solutions below.

Mismatch with traditional software/OS stack. Newly-emerged hardware sometimes does not match with the traditional software/OS stack, which will cause unexpected behaviors or performance degradation. For example, although the byte addressability and durable writes of NVM render it perfect for building an in-memory database, simply replacing DRAM or the disk with NVM to build an in-memory database fails to take advantage of these features. This is because of the mismatch between block/page granularity of OS paging and the byte granularity of NVM, and the fact that the system is oblivious to the limited lifetime of NVM. The disparity in the write patterns between two partitions as described in Section 3.4, and the workload with high write skewness, can result in some NVM cells being written much more times and consequently worn out much earlier than others. This poses a stringent requirement for the wear-levelling algorithm. In addition, traditional file system requires *read* or *write* syscalls to access the file, whose overhead can play a significant role in the total latency when accessing NVM.

Scalability. The scale of some new hardware cannot catch up with advances of other parts of the system, and some new hardware cannot easily scale up/out without significant performance degradation. For example, it is not uncommon that a multicore machine has more than 100 cores and hundreds or thousands gigabytes of memory, but HTM can only scale to a limited number of CPU cores [16], and Xeon Phi coprocessor can only support up to 16 GB memory. Doubts have also been raised on the scalability of some of the RDMA solutions [4]. RDMA memory consumption also poses a big challenge on large cluster due to the flexibility of communication model and user-dependent message passing implementation, losing the advantages that can be derived from mature network stack mechanism and memory management by OS.

Another scalability issue is that the current hardware support for virtual memory does not scale to terabytes (and beyond) of physical memory. The use of small page sizes and fine protection boundaries will require significant space overhead in the corresponding page tables. The naïve solution of enlarging page sizes is only a stop-gap measure as it will eventually lead to large page protection boundaries, fragmentation, and inefficient use of memory.

Generality/Compatibility. Hardware solutions are usually architecture specific, and not general enough to satisfy the different requirements from a variety of applications. For example, RDMA cannot easily communicate with the traditional Ethernet network directly. In addition, not all database transaction semantics can be expressed using HTM, since there are some other factors restricting its usage, e.g., limited HTM transaction size that is restricted by L1 cache, unexpected transaction abort due to cache pollution problems. Moreover, data alignment requirement for SIMD makes SIMD-based implementation architecture specific.

Extra overhead. In order to utilize some hardware, there are extra preparation work to do, which may offset the performance gain from the hardware. For example, in order to use SIMD instructions, we need to gather data into SIMD registers and scatter it to the correct destination locations, which is tricky and can be inefficient. RDMA only allows a pre-registered memory region to be the transmission source/destination, which also cannot be released/ modified until the transmission is completed.

Bottleneck shift. Even though the use of new hardware may tackle one bottleneck, the contribution to the system's overall performance may be restricted. In some cases, it only results in a bottleneck shift. For instance, the adoption of RDMA usually moves the bottleneck from network I/O to CPU, as with a fast networking, CPU is usually busy with data preparation and notification checking [11, 2]. And even though HTM can reduce the overhead caused by concurrency control to some extent, some components such as durability and communication still restrict the overall performance.

5. INTEGRATED SOFTWARE AND HARDWARE DESIGN

In this section, we discuss some open research directions in taking advantage of both software and hardware. We believe that hardware solutions, when combined with software solutions, would be able to fully exploit the potentials of in-memory databases.

Atomic primitives can be used for single object synchronization, and virtual snapshot by *fork*-ing facilitates a hardware-assisted isolation among processes [12]. HTM combined with other concurrency control mechanisms (e.g., timestamp-based) can be an alternative to the lock-based mechanism, but its special features (e.g., limited transaction size, unexpected aborts under certain conditions) should be taken into consideration. A mix of these protection mechanisms should enable a more efficient concurrency control model. Since the bottleneck for in-memory databases shifts from disk to memory. a good concurrency control protocol also needs to consider the underlying memory hierarchy, such as NUMA architecture and caches, whose performance highly depends on the data locality. The coordination overhead caused by 2PC protocol can be further alleviated, by eliminating distributed transactions via a dynamic data partition strategy, or designing a protocol based on distributed atomic operations provided by RDMA, for example. A clientoriented transaction execution strategy, where the processing is performed at the client side simultaneously rather than in a centralized server, is also promising, which is made viable by the one-sided networking feature provided by RDMA.

In order to speed up the performance, various levels of parallelism should be exploited. Specifically, the data-level parallelism (e.g., bit-parallel, SIMD) can make extensive use of the "circuits" for parallelization. GPU, with a massively parallel architecture consisting of thousands of smaller cores, fits perfectly for embarrassingly parallel algorithms (e.g., filter, deep learning). Thus a software-coordinated CPU-GPU framework, which combines CPU's generality and GPU's specificity, can be utilized to distribute the tasks with different parallelism properties to different units in the warehouse or OLAP systems. The emergence of MIC co-processors (e.g., Intel Xeon Phi), provides a promising alternative for parallelizing computation, with many lower-frequency in-order cores and wider SIMD. Nevertheless, robust data structures that are parallelism-conscious, memory-economical, and access-efficient form the foundation for further parallelism exploration. For example, the skip-list, which allows fast point- and range-queries of an ordered sequence of elements with $O(logn)$ complexity, can potentially be an alternative to B-trees for in-memory databases, as it can be implemented latch-free easily and can be structured to be more parallelism-conscious (e.g., SIMD-friendly, NUMA-aware).

Distributed computing requires fast networking in order to achieve high scalability, where RDMA can play a big role. However, simply relying on RDMA networking is not guaranteed to improve the system performance, due to the restrictions of RDMA, bottleneck shift issues, etc. Combined with a good partition strategy (i.e., to achieve data locality), and efficient communication model (e.g., batch or coalescing transmission), the communication performance can be significantly enhanced. Besides, special features provided by RDMA should be taken into consideration, such as inline data, unsignalled, unreliable transport type, to fully exert its performance potential. And a heterogeneous software-coordinated communication model is also worth investigating, which can exploit the advantages from both the Ethernet and RDMA networks. Moreover, with the increasing throughput of RDMA network, the throughput of intra- and inter-server (i.e., memory bus among NUMA nodes and network in a cluster) is becoming similar. Hence, it is possible to develop a unified system framework that can be used in both a single server with multiple NUMA nodes and a cluster connected via high-speed networks.

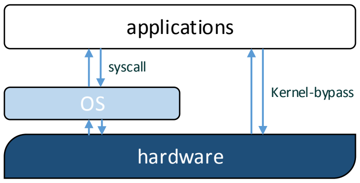


Figure 2: Interfaces with Hardware

For NVM-based in-memory databases, we believe a unified space management is required to effectively exploit its features (e.g., byte addressability and durable write). Although NVM is proposed to be placed side-by-side with DRAM on the memory bus, its distinct characteristics, such as limited endurance, write/read asymmetry, uncertainty of ordering and atomicity make it difficult to work effectively and efficiently. One way is to manage the NVM space as the main memory in a log-structured manner, such that the unnecessary reads/writes and the expensive syscalls, if used as a block device, will be eliminated, and the sequential write can be fully exploited. Due to the append-only feature of the log, the writes to NVM will be distributed uniformly among all cells, which in turn prolongs the lifetime of NVM. Besides, NVM enables more efficient fault tolerance strategies, if equipped with carefully designed algorithms to guarantee write atomicity and deterministic orderings.

As shown in Figure 2, the manipulation of hardware can be achieved either through syscalls or kernel-bypass methods. Some new hardware already provides direct kernel-bypass interfaces. But with kernel-bypass, the mature functionalities of the OS, such as memory management, concurrency control, buffer management are no longer available. It will also mean breaking the traditional boundaries of protection and separation of responsibilities. It is essential to retain key features of traditional OSes, even if direct access to the hardware is enabled. This can be achieved by moving the data path from the kernel space to the user space, resulting in a data-plane OS. Alternatively, new ABI boundaries will have to be drawn so that infrequent yet secure operations are handled over to the OS, while others are executed directly in the user-space.

6. CONCLUSIONS

In this paper, we summarized the challenges of in-memory databases, and their solutions from both the software and hardware perspectives. While hardware solutions are known for its efficiency with less overhead, as shown in this paper, they do not always outperform software solutions. In order to fully exploit the potentials of in-memory systems, we believe that a combined hardware-software solution is needed. There are a lot more that our community can bring to the table!

7. ACKNOWLEDGMENTS

This work was supported by the National Research Foundation, Prime Ministers Office, Singapore under Grant No. NRF-CRP8-2011-08.

8. REFERENCES

[1] A. Belay, G. Prekas, A. Klimovic, S. Grossman, C. Kozyrakis, and E. Bugnion. Ix: A protected dataplane operating system for high throughput and low latency. In *OSDI '14*, pages 49–65, 2014.

[2] Q. Cai, H. Zhang, G. Chen, B. C. Ooi, and K.-L. Tan. Memepic: Towards a database system architecture without system calls. Technical report, NUS, 2015.

[3] C. Cascaval, C. Blundell, M. Michael, H. W. Cain, P. Wu, S. Chiras, and S. Chatterjee. Software transactional memory: Why is it only a research toy? *Queue*, 6(5):40:46–40:58, Sept. 2008.

[4] Chelsio. Roce at a crossroads. Technical report, Chelsio Communications Inc., 2014.

[5] S. Damaraju, V. George, S. Jahagirdar, T. Khondker, R. Milstrey, S. Sarkar, S. Siers, I. Stolero, and A. Subbiah. A 22nm ia multi-cpu and gpu system-on-chip. In *ISSCC '12*, pages 56–57, 2012.

[6] J. DeBrabant, A. Joy, A. Pavlo, M. Stonebraker, S. Zdonik, and S. R. Dulloor. A prolegomenon on oltp database systems for non-volatile memory. In *ADMS '14*, pages 57–63, 2014.

[7] Z. Feng, E. Lo, B. Kao, and W. Xu. Byteslice: Pushing the envelop of main memory data processing with a new storage layout. In *SIGMOD '15*, 2015.

[8] S. Harizopoulos, D. J. Abadi, S. Madden, and M. Stonebraker. Oltp through the looking glass, and what we found there. In *SIGMOD '08*, pages 981–992, 2008.

[9] S. Jha, B. He, M. Lu, X. Cheng, and H. P. Huynh. Improving main memory hash joins on intel xeon phi processors: An experimental approach. In *PVLDB '15*, pages 642–653, 2015.

[10] E. P. C. Jones, D. J. Abadi, and S. Madden. Low overhead concurrency control for partitioned main memory databases. In *SIGMOD '10*, pages 603–614, 2010.

[11] A. Kalia, M. Kaminsky, and D. G. Andersen. Using rdma efficiently for key-value services. In *SIGCOMM '14*, pages 295–306, 2014.

[12] A. Kemper and T. Neumann. Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *ICDE '11*, pages 195–206, 2011.

[13] P.-A. Larson, S. Blanas, C. Diaconu, C. Freedman, J. M. Patel, and M. Zwilling. High-performance concurrency control mechanisms for main-memory databases. In *PVLDB '11*, pages 298–309, 2011.

[14] J. Lee, Y. S. Kwon, F. Farber, M. Muehle, C. Lee, C. Bensberg, J. Y. Lee, A. H. Lee, and W. Lehner. Sap hana distributed in-memory database system: Transaction, session, and metadata management. In *ICDE '13*, pages 1165–1173, 2013.

[15] S. Lee, M. Kim, G. Do, S. Kim, H. Lee, J. Sim, N. Park, S. Hong, Y. Jeon, K. Choi, et al. Programming disturbance and cell scaling in phase change memory: For up to 16nm based $4F^2$ cell. In *VLSIT '10*, pages 199–200, 2010.

[16] V. Leis, A. Kemper, and T. Neumann. Exploiting hardware transactional memory in main-memory databases. In *ICDE '14*, pages 580–591, 2014.

[17] F. Li, B. C. Ooi, M. T. Özsu, and S. Wu. Distributed data management using mapreduce. *ACM Computing Surveys*, 46(3):31:1–31:42, Jan. 2014.

[18] H. Li, X. Wang, Z.-L. Ong, W.-F. Wong, Y. Zhang, P. Wang, and Y. Chen. Performance, power, and reliability tradeoffs of STT-RAM cell subject to architecture-level requirement. *IEEE Transactions on Magnetics*, 47(10):2356–2359, Oct. 2011.

[19] D. Loghin, B. M. Tudor, H. Zhang, B. C. Ooi, and Y. M. Teo. A performance study of big data on small nodes. In *PVLDB '15*, 2015.

[20] L. M. Maas, T. Kissinger, D. Habich, and W. Lehner. Buzzard: A numa-aware in-memory indexing system. In *SIGMOD '13*, pages 1285–1286, 2013.

[21] N. Malviya, A. Weisberg, S. Madden, and M. Stonebraker. Rethinking main memory oltp recovery. In *ICDE '14*, pages 604–615, 2014.

[22] C. Mitchell, Y. Geng, and J. Li. Using one-sided rdma reads to build a fast, cpu-efficient key-value store. In *USENIX ATC '13*, pages 103–114, 2013.

[23] T. Neumann, T. Mühlbauer, and A. Kemper. Fast serializable multi-version concurrency control for main-memory database systems. In *SIGMOD '15*, 2015.

[24] A. Pavlo, C. Curino, and S. Zdonik. Skew-aware automatic database partitioning in shared-nothing, parallel oltp systems. In *SIGMOD '12*, pages 61–72, 2012.

[25] K. Ren, A. Thomson, and D. J. Abadi. Lightweight locking for main memory database systems. In *PVLDB '13*, pages 145–156, 2013.

[26] L. Rizzo. Netmap: A novel framework for fast packet i/o. In *USENIX ATC '12*, pages 101–112, 2012.

[27] S. M. Rumble, A. Kejriwal, and J. Ousterhout. Log-structured memory for dram-based storage. In *FAST '14*, pages 1–16, 2014.

[28] S. Sanfilippo and P. Noordhuis. Redis. http://redis.io, 2009.

[29] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden. Speedy transactions in multicore in-memory databases. In *SOSP '13*, pages 18–32, 2013.

[30] T. Willhalm, N. Popovici, Y. Boshmaf, H. Plattner, A. Zeier, and J. Schaffner. Simd-scan: Ultra fast in-memory table scan using on-chip vector processing units. In *PVLDB '09*, pages 385–394, 2009.

[31] C. Yao, D. Agrawal, P. Chang, G. Chen, B. C. Ooi, W.-F. Wong, and M. Zhang. Dgcc: A new dependency graph based concurrency control protocol for multicore database systems. *ArXiv e-prints*, Mar. 2015.

[32] X. Yu, G. Bezerra, A. Pavlo, S. Devadas, and M. Stonebraker. Staring into the abyss: An evaluation of concurrency control with one thousand cores. In *PVLDB '15*, pages 209–220, 2014.

[33] H. Zhang, G. Chen, B. C. Ooi, K.-L. Tan, and M. Zhang. In-memory big data management and processing: A survey. *TKDE*, 27(7):1920–1947, July 2015.

[34] H. Zhang, G. Chen, W.-F. Wong, B. C. Ooi, S. Wu, and Y. Xia. Anti-caching-based elastic data management for big data. In *ICDE '15*, pages 592–603, 2014.

[35] H. Zhang, B. M. Tudor, G. Chen, and B. C. Ooi. Efficient in-memory data management: An analysis. In *PVLDB '14*, pages 833–836, 2014.