Performance Characterization of Modern Databases on Out-of-Order CPUs

Reena Panda, Christopher Erb, Michael Lebeane, Jee Ho Ryoo and Lizy Kurian John
The University of Texas at Austin
{*reena.panda, cde593, mlebeane, jr45842*}*@utexas.edu, ljohn@ece.utexas.edu*

***Abstract*—Big data revolution has created an unprecedented demand for intelligent data management solutions on a large scale. While data management has traditionally been used as a synonym for relational data processing, in recent years a new group popularly known as NoSQL databases have emerged as a competitive alternative. There is a pressing need to gain greater understanding of the characteristics of modern databases to architect targeted computers. In this paper, we investigate four popular NoSQL/SQL-style databases and evaluate their hardware performance on modern computer systems. Based on data collected from real hardware, we evaluate how efficiently modern databases utilize the underlying systems and make several recommendations to improve their performance efficiency. We observe that performance of modern databases is severely limited by poor cache/memory performance. Nonetheless, we demonstrate that dynamic execution techniques are still effective in hiding a significant fraction of the stalls, thereby improving performance. We further show that NoSQL databases suffer from greater performance inefficiencies than their SQL counterparts. SQL databases outperform NoSQL databases for most operations and are beaten by NoSQL databases only in a few cases. NoSQL databases provide a promising competitive alternative to SQL-style databases, however, they are yet to be optimized to fully reach the performance of contemporary SQL systems. We also show that significant diversity exists among different database implementations and big-data benchmark designers can leverage our analysis to incorporate representative workloads to encapsulate the full spectrum of data-serving applications. In this paper, we also compare data-serving applications with other popular benchmarks such as SPEC CPU2006 and SPECjbb2005.**

I. Introduction

The big-data revolution has created an unprecedented demand for efficient data management solutions. Traditionally, data management systems were primarily driven by relational database management systems (RDBMS) based on the structured query language (SQL). However, in recent years, a new group of data management solutions, popularly known as NoSQL (Not-Only SQL) databases, have emerged as a competitive alternative. NoSQL technology is becoming increasingly popular because of its improved flexibility and scalability over RDBMS, and notable companies such as Google, Facebook etc with considerable amounts of online data are adopting NoSQL-style databases. Nevertheless, SQL systems have their own share of strengths: greater structure, powerful interface, ACID compliance and complex operation support. As SQL and NoSQL databases capitalize on different organization philosophies, neither database completely out-rules the other, and both have spaces in which one can be applied more successfully than the other. Recent years have thus, seen a surge in the number and diversity of SQL and NoSQL databases. Undoubtedly, SQL and NoSQL systems form a very important class of applications that are run on modern computer systems and it is imperative to understand their performance characteristics to architect targeted computers.

Several past studies [1], [2], [3] have evaluated the performance of online transaction processing (OLTP) workloads. However, significant advances have been made in computer designs during the last two decades. Are current architectures well positioned to meet the challenges imposed by modern databases working on large datasets? Are the emerging databases ill-suited to run on contemporary hardware systems? We undertake a detailed evaluation of four widely popular, modern NoSQL and SQL databases (Cassandra [4], MongoDB [5], MySQL [6] and VoltDB [7]) in order to discover their performance characteristics on modern systems. Based on data collected from real hardware, our evaluation aims to provide meaningful insights not only for computer architects to design targeted hardware with improved compatibility with database applications, but also for database designers to understand the performance of their software and tune it to better exploit underlying systems. To the best of our knowledge, the work presented in this paper is the first work that extensively compares the microarchitectural performance of different NoSQL and SQL offerings on modern computer systems.

Many recent research studies [8], [9], [10] have evaluated the overall performance spectrum of big-data applications, however their evaluation of the data-serving space has mostly been limited to a few database applications. Many big-data benchmarking suites [8] [9] have also been recently proposed, however these suites include at most a single (or two) database run(s) to represent the entire class of data-serving applications. In the light of the ever increasing diversity and number of NoSQL/SQL databases, we evaluate a wider range of databases and seek to find out how the different solutions compare against each other. Based on our evaluation, we show that there exists significant variability in terms of both performance and inherent characteristics between the different databases. While many recent benchmarking suites include one program to represent the class of data-serving applications, we find that it is not enough to be completely representative.

We also compare the database applications with three widely popular benchmarking suites representing other application classes, desktop-computing (SPEC CPU2006 [11] (SPEC)), Java applications (SPECjbb2005 [12] (SPECjbb)) and high-performance computing (HP Linpack [13] (Linpack)). Our evaluation identifies important correlations among the tested workloads in terms of specific performance characteristics (e.g., cache behavior) and can aid performance analysts to select most representative substitute benchmarks to evaluate performance of data-serving applications if they cannot run real-world applications, which is an important concern especially in early stages of design analysis.

Based on our evaluation of the performance of four NoSQL/SQL databases, we seek to answer the following :-

- How effectively do modern SQL and NoSQL

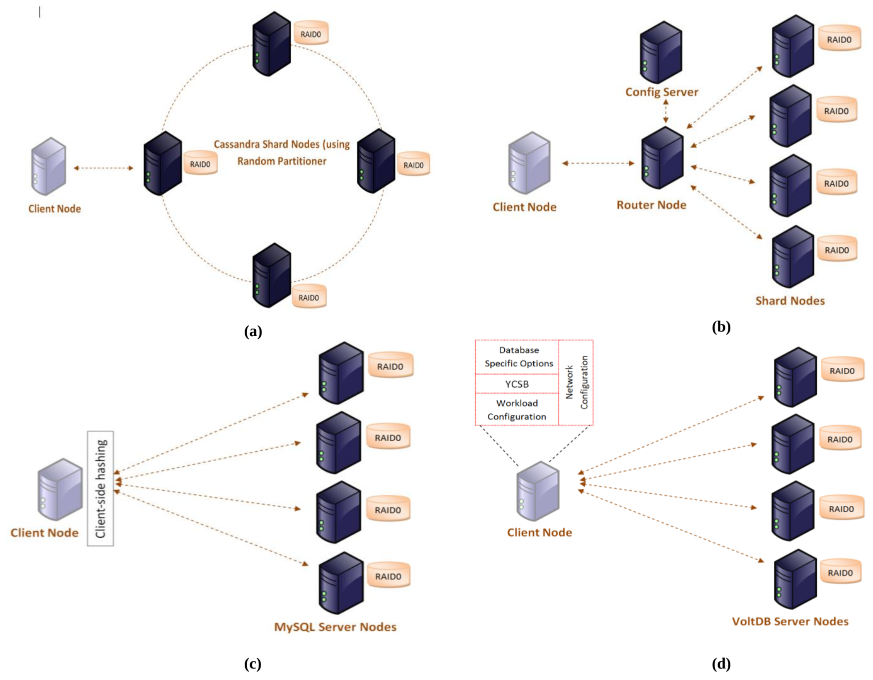

Fig. 1: Experimental Multi-node Setup for (a) Cassandra, (b) MongoDB, (c) MySQL, (d) VoltDB

databases utilize the underlying computer systems? For instance, are out-of-order (OOO) processors necessary for modern databases?

- What are the sources of any performance inefficiencies?
- Is the performance of NoSQL databases different from SQL databases?
- Is the performance of data-serving applications noticeably different from other application classes? If so, what are the most unique features defining the behavior of the database applications.
- Do any benchmarks from standard suites exhibit resemblance to database applications in terms of selected performance metrics (e.g., cache behavior)?

The rest of this paper is organized as follows: In Section 2, we present an overview of the databases and our methodology. In section 3, we describe our test infrastructure. Section 4 presents our results and analysis. Finally, we discuss prior work and summarize our key findings in sections 5 and 6 respectively.

II. Methodology

In this section, we present a brief background of the tested databases, benchmarks and our experimental methodology.

A. Databases

*1) **Cassandra**:* Apache Cassandra [4] is a popular Java implementation of a column-family style NoSQL database. Cassandra is incrementally scalable, eventually consistent, and has no single point of failure. Every node in the Cassandra cluster knows of and has the key for at least one other node and any node can service a request. The node structure can be best visualized as a ring of interconnected nodes. It was primarily designed for write-intensive applications and it tries to achieve linear throughput scaling with the number of nodes. Reads have added time complexity over writes because Cassandra has a temporary data structure for recent writes and consistency across nodes is enforced at read time through majority voting. Cassandra is semi-structured in that its data may share some of the same fields/columns, but not all. Our experimental setup of Cassandra is shown in Figure 1a, which includes four data-partitioned server nodes, connected in a ring network.

*2) **MongoDB**:* MongoDB [5] is a C++ implementation of a document-style NoSQL database. It is designed for speed and scalability and can store large documents such as binaries, audio/video files etc. It has a flexible schema and it allows objects to not have fixed schema or type. Documents are stored as Binary JSON objects and may be organized into collections. Within a collection, each document has a primary key, and an index can be created for each query-able field. Queries are normally performed over a collection, but more complex queries may be performed using a variation of map-reduce. MongoDB's data can be of any type, and it is searched using keys and meta-data information. Our multi-node setup of MongoDB is shown in Figure 1b. The dataset is sharded over four server nodes and we use separate nodes for MongoDB's router and config-server. Client node connects with the router node, which then distributes work over the server nodes.

*3) **MySQL**:* MySQL [6] is one of the world's most popular open-source RDBMS. It enables the cost-effective delivery of reliable, high-performance and scalable web-based and embedded database applications. MySQL is designed to work on data whose fields are structured and finite in number. Given this knowledge of data, MySQL is able to organize and search through it in multiple dimensions. This is both its strength and its limitation, as it cannot use the same strategy on less structured data. MySQL does not support automatic data sharding. Our multi-node MySQL server setup (Figure 1c) employs client-side hashing to determine the home node for data records. This approach to scale-out MySQL database has also been used by other research studies [14]. Since the type of operations performed against the databases are simple (read, write etc), the operations do not involve any significant communication between the server nodes and so, can be compared with other automatically sharded databases without loss of accuracy.

*4) **VoltDB**:* VoltDB [7] is a scale-out in-memory database, belonging to the class of modern RDBMSs, that seeks to offer the speed and scalability of NoSQL databases but with ACID guarantees, relational data models, and transactional capability of traditional RDBMSs. It uses a shared nothing architecture to achieve database parallelism. Data and its corresponding processing is distributed among all the CPU cores within the servers composing a single VoltDB cluster. By extending its shared-nothing foundation to the per-core level, VoltDB scales with the increasing core-per-CPU counts on modern commodity servers. VoltDB relies on horizontal partitioning down to the individual hardware thread to scale, synchronous replication to provide high availability and a combination of continuous snapshots and command logging for durability. Our multi-node setup of VoltDB is shown in Figure 1d.

B. *Benchmark Description*

We use the Yahoo! Cloud Serving Benchmark (YCSB)[14] as the client for integrating against our tested databases. YCSB is a standard benchmarking framework that was developed to assist in the evaluation of different cloud systems. The YCSB framework consists of a workload generating client and a package of standard 'core' workloads. Details of YCSB's core workloads are provided in Table I. The YCSB workloads cover a majority of the most important operations, which are performed against a typical data-serving database. Our test database is generated using the YCSB framework and consists of over 10 million records, for a total size of over 12 GB. The YCSB workloads executed between 5-20 billion total instructions on average for all the databases, with different operations needing different number of instructions to complete.

We also compare the performance of NoSQL and SQL databases with several widely used benchmarking suites representing other application classes. We evaluate SPEC CPU2006 benchmarks as a representative of the general-purpose, desktop applications. We run 20 out of 26 benchmarks from the SPEC CPU2006 suite, six Fortran benchmarks could not be run on our test infrastructure due to compile/run-time issues. SPECjbb is a SPEC benchmark for evaluating performance and scalability of real-world server-side Java business applications.

Workload	Operations	Record Selection	Application Example
A - Update heavy	Read: 50%,Update: 50%	Zipfian	Session store recording recent actions in a user session
B - Read heavy	Read: 95%,Update: 5%	Zipfian	Photo tagging; add a tag is an update, but most operations are to read tags
C - Read only	Read: 100%	Zipfian	User profile cache, where profiles are constructed elsewhere (e.g, Hadoop)
D - Read latest	Read: 95%,Insert: 5%	Latest	User status updates; people want to read the latest status
E - Short Ranges	Scan: 95%,Insert: 5%	Zipfian/ Uniform	Threaded conversations, where each scan is for the posts in a given thread

TABLE I: YCSB Core Workloads

Performance Metric	Description
Instructions per Cycle (IPC)	Total retired instructions (insts) / Total cycles
UOPs / Instruction	Total retired UOPs / Total retired insts
IMIX-Control	Retired control insts / Total retired insts
IMIX-Loads	Retired load insts / Total retired insts
IMIX-Stores	Retired store insts / Total retired insts
IMIX-INT	Retired integer insts / Total retired insts
IMIX-FP	Retired floating-point(FP) insts / Total retired insts
IMIX-SIMD	Retired SIMD insts / Total retired insts
FLOPs / Instruction	Retired FP operations / Total retired insts
Cond. Branch Frequency	Retired conditional branches / Retired branches
Branch Taken Rate	Total taken branches / Total branches
Branch Speculation Factor	Total branches decoded / Total branches retired
Call/Return Frequency	Total calls + returns / Total branches retired
Branch Misprediction Rate	Total branch mispredictions / Total branches retired
L1 DCache MPKI (L1D)	L1 data cache (DCache) misses / Retired kilo-insts
L1 DTLB MPKI (L1 DTLB)	L1 data translation look-aside buffer (DTLB) misses / Total kilo-insts retired
L1 ICache MPKI (L1I)	L1 inst cache (ICache) misses/ Retired kilo-insts
L1 ITLB MPKI (L1 ITLB)	L1 ITLB misses / Total kilo-insts retired
LLC MPKI	LLC misses / Total kilo-insts retired
Page-walks per kilo-inst	Page-walk count / Total kilo-insts retired
Memory transactions per Inst	Memory bus transactions / Total kilo-insts retired
ROB Full Stalls	Reorder buffer (ROB) full resource (res) stalls / res stalls
RS Full Stalls	Reservation Station (RS) full res stalls / res stalls
LD/ST Buffer Full Stalls	Load-Store (LDST) buffer full res stalls / res stalls
FPCW Stalls	FPCW res stalls / res stalls
BrMis Prediction Stalls	Branch Misprediction res stalls / Total res stalls
RAT Stalls	Register Alias Table (RAT) stalls / Total res stalls
Front-end Stalls (Fr-St)	Front-end stalls / Total res stalls
Resource Stalls/Total Cycles	Total res stalls / Total cycles

TABLE II: Measured Hardware Performance Counter Metrics

It resembles an online transaction processing based application and is implemented in an object-oriented manner to emulate a three-tier client/server system. High-Performance(HP) Linpack benchmark is a high-performance computing benchmark, which solves a dense system of linear equations. It is a very popular benchmark and is used to measure performance of supercomputers (Top 500). We parametrize Linpack to run on a 10 GB dataset size.

C. *Performance Metrics*

To characterize hardware performance, we use performance counters available on contemporary microprocessors. Performance counters provide the means to track detailed events that occur on an actual chip. Details of all the performance metrics we measure are provided in Table II. We also capture and compare the fraction of time each application spends executing operating system (OS) and user/database code and its performance implications.

D. *Similarity Analysis*

We perform a systematic (dis)similarity study between the databases and SPEC, SPECjbb and Linpack benchmarks. Since the number of performance metrics is large, we use statistical analysis techniques to compare all the programs based on their measured characteristics. To remove any correlations between the measured data, we first pre-process the data by normalizing the raw data to a unit normal distribution and performing principal component analysis (PCA) [15] on the same. PCA is an effective statistical data analysis technique to reduce the dimensionality of a dataset, while maintaining most of its original information. Finally, we use hierarchical clustering in the PCA space to find clusters of programs possessing similar properties. Our benchmark similarity analysis approach is similar to what has been adopted in prior research [16], [17] for traditional single-node workloads.

III. Experimental Framework

For the experiments, we use a four-node configuration that is more representative of scale-out server deployments. In the multi-node setup, individual nodes are connected through a 1Gbps Ethernet network. Each node has two 64-bit Intel(R) Xeon(R) E5405 processors running at 2GHz, each having four out-of-order cores. Each core has private L1 caches (64 KB instruction and 64 KB data cache) and the cores share a single 12 MB L2 cache. Each node has 16GB of main memory.

Individual database server nodes are setup as shown in Figure 1. We use MongoDB version 2.6.5, running one mongod instance per server node. We use different nodes for MongoDB's config server and router node. We verified that the router node and config server were lightly loaded and were not bottlenecks in our tests. We use Cassandra version 0.1.7 with Java Oracle JDK version 1.7 and a JVM heap size of 8GB. The Cassandra multi-node cluster was setup using four nodes, connected in a peer-to-peer ring-based network. We use Cassandra's RandomPartitioner (hash-based) to partition data among the nodes evenly. We use MySQL version 5.1.15. We use VoltDB version 5.3, running one VoltDB instance per server node.

All our tests are run using the client-server model. Dataset is generated using the YCSB framework and it consists of over 10 million records, for a total database size of over 12 GB. The dataset size was chosen so that the data fits into memory of the server nodes, which is the recommended operational setup for scale-out applications for better database performance [18]. In our tests, read operations retrieve an entire record, while updates modify one of the fields. We run 100,000 operations against the databases using 32 client threads. In order to evaluate microarchitectural performance, we use Linux perf tool [19] that provides an interface to the processor performance counters. To ensure greater data reliability, we repeat our experiments ten times over the entire execution of the applications.

IV. Evaluation and Analysis

In this section, we present a detailed evaluation of performance characteristics of modern NoSQL/SQL databases. The naming convention used for benchmarks in the following sections is:- Databases are represented as Wx-DB, where x is the YCSB workload type (A-E) and DB is the database name and "Avg-DB" is the average value across all databases. SPEC-INT and SPEC-FP represent the average value for SPEC integer and floating-point benchmarks respectively.

A. *Overall Micro-architectural Performance Efficiency*

We begin our discussion by evaluating the overall performance efficiency of the four databases on modern OOO processors. We are primarily interested to find out how the database cycles per instruction (CPI) correspond to the theoretical CPIs and identify the main performance bottlenecks. Figure 2 shows the IPC of all four databases running the YCSB workloads. IPC metric indicates how many instructions can be executed simultaneously and can be used to measure the instruction level parallelism of the application. The overall measured IPC of the databases is 0.58. Database workloads

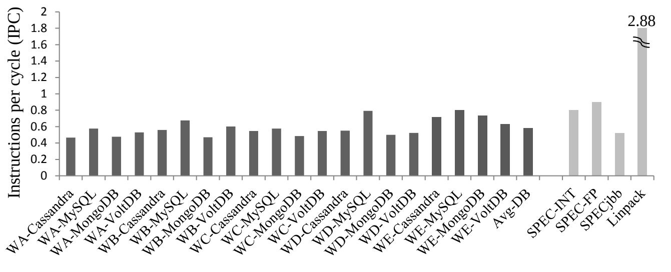

Fig. 2: Instructions per cycle for all workloads

have much lower IPCs (less than 0.6) than SPEC-INT, SPEC-FP and Linpack benchmarks but their IPC values are closer to the SPECjbb benchmark. MySQL has the highest IPC among the data-serving workloads, while MongoDB has the lowest. It is also interesting to observe that different types of operations lead to different IPCs in database applications, which implies that different operations have different performance requirements and bottlenecks. It is to be noted that scan workload (WorkloadE) IPC is higher for all four databases. But scan operations also use significantly more number of instructions to execute the same number of database operations, which translates to their poor performance. This is especially true for Cassandra which executes 10-20x more instructions for scan operations as compared to other read/write operations.

Figure 3 shows the breakdown of retired instructions into database/user and OS mode. We observe that all the databases spend 70 - 85% of their execution time in database code, with an average IPC of 0.62. Remaining time is spent in executing operating system code (approximately 15-30% of instructions with a lower average IPC of 0.34). Such high OS activity in data serving workloads arises because of handling a large number of data requests that lead to higher number of disk/network activity. It is also interesting to observe that NoSQL databases execute a larger fraction of OS instructions (25% of OS code) than SQL databases (15% of OS code) for all YCSB workloads.

For different workloads, IPC can be limited by several reasons: pipeline stalls, instruction dependencies and memory stalls. We show the breakdown of execution cycles in the four databases into useful computation and stall cycles in Figure 4. Comparing the useful computation and stall cycle breakdown of the workloads, we can observe that stall cycles dominate the overall execution. Poor cache/memory subsystem performance is the major contributor to the overall stall cycles, as we will discuss further in the the next section.

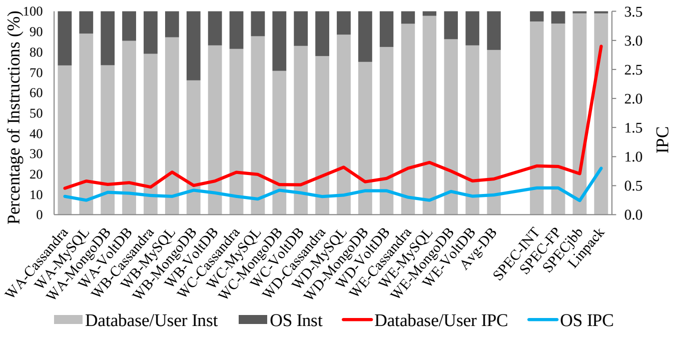

Fig. 3: Percentage breakdown of user/database and OS instructions for each workload

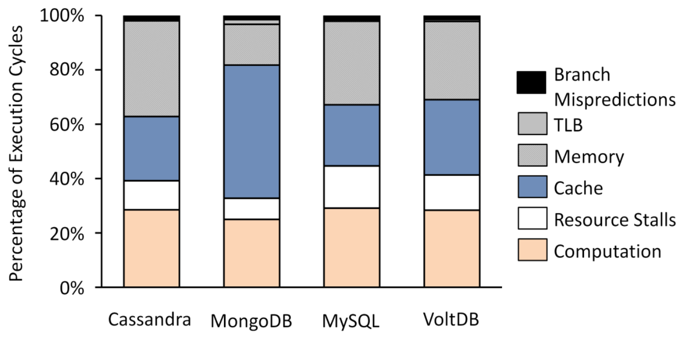

Fig. 4: Percentage breakdown of total execution cycles

B. *Cache and Memory System Behavior*

Memory system related stalls comprise a significant portion of the total execution time in modern databases. In this section, we evaluate the cache/memory (both for instruction and data) performance of different databases.

Instruction caches and TLBs are critical components as they are used to fetch and feed instructions to the core. Higher instruction cache or TLB misses are detrimental to performance, because these misses cause front-end stalls in the pipeline and reduce pipeline's efficiency. Figures 5 and 6 show the L1 ICache and TLB misses (ITLB induced page walks) per thousand instructions for each workload respectively. Both NoSQL and SQL databases experience high ICache (27 MPKI average) and TLB misses. MongoDB experiences the worst ICache and ITLB performance, which results in significant front-end pipeline stalls for MongoDB. In contrast, ICache misses are relatively rare in SPEC, SPECjbb and Linpack benchmarks. This can be attributed to several reasons. First, database applications have much higher application footprints than typical SPEC/Linpack applications which exceeds the on-chip cache capacity of modern processors. Second, third-party libraries used by the databases further increase the size of their application binaries. Third, databases execute significantly higher fraction of OS code and as a result, the limited on-chip ICache/TLB capacity is shared by both OS and database code. Cassandra's bigger codebase may also be attributed to its JVM dependency. Modern processors are equipped with simple next-line instruction prefetchers, which are not quite effective for the tested databases. Intelligent prefetching schemes targeting specific instruction execution patterns is likely to improve the front-end performance/power-efficiency of databases significantly.

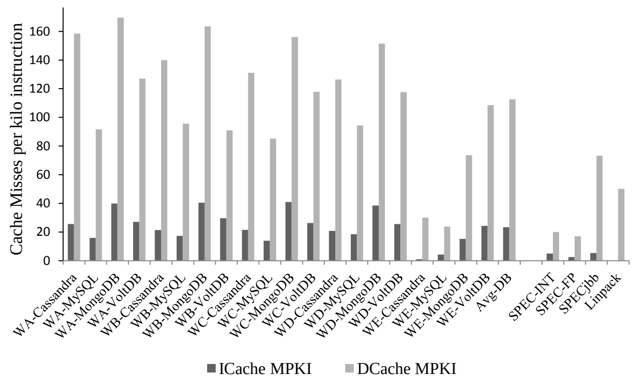

Fig. 5: L1 Instruction and Data Cache Misses per kilo instructions (MPKI) for each workload

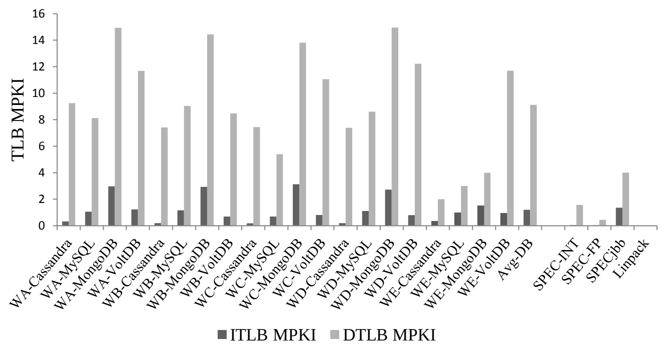

Fig. 6: Instruction and Data TLB Misses per kilo instructions for each workload

Figures 5, 6 and 7 show the data cache, data TLB and unified last-level cache (LLC) performance of the workloads respectively. We can see that data serving applications have significantly higher data cache/TLB misses than SPEC applications. MongoDB suffers significantly due to its poor L1 data cache performance, its data does not fit adequately into L1 resulting in over 160 DCache MPKI. Despite MongoDB's poor instruction and data cache performance, its instructions and data fit into the LLC, which results in lower LLC cache MPKI as shown in Figure 7. MongoDB's lower IPC is partly attributed to its poor cache performance as a result of which it is constantly waiting for memory operations missing its L1 cache. Cassandra and MySQL have better L1 data cache and TLB performance as compared to MongoDB, but they both experience higher LLC cache misses. Cassandra has higher LLC MPKI than MySQL for all the workloads. Optimizations targeting to improve the performance of instruction and data caches can substantially improve the performance of all databases, especially MongoDB.

C. *Processor Pipeline Issues*

1) Control flow Performance: Figure 8 shows the control flow performance of NoSQL/SQL databases. Control instructions add up to 15-20% of their overall instruction mix. We also observe that databases experience higher branch misprediction rates (over 6%) as compared to SPEC applications. Linpack experiences the lowest branch misprediction rate due to its regular control flow structure. Branch predictor performance is critical because incorrect predictions, especially in today's deep, complex pipelines lead to expensive pipeline flushes and stalls. MongoDB experiences the highest misprediction rates (over 5.5%) for most workloads. MySQL is a close second sitting around 5%. In contrast to prior studies on OLTP-like workloads, we observed that branch misprediction rates are lower than prior reported numbers (over 15-20% in [1]) owing to improvements in branch predictor designs. As a result,

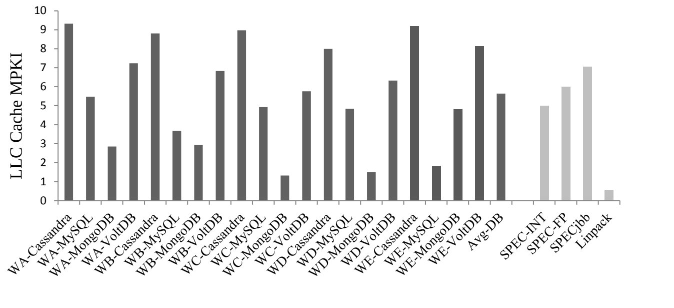

Fig. 7: LLC Misses per kilo instructions for each workload

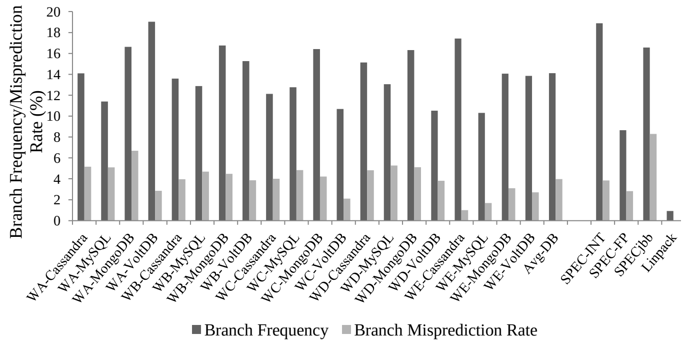

Fig. 8: Branch Misprediction Rate for each workload

branch-misprediction related stalls contribute much less to the overall stall cycles.

*2) **Does Out-of-Ordering help?**:* Out-of-ordering and speculative execution engines provide significant scope for performance optimizations, but also lead to complex and power-hungry micro-architecture designs. As database performance is limited significantly by the memory system, many may argue to employ simpler in-order execution engines to achieve better power efficiency without degrading performance. In this section, we investigate how effective are the dynamic execution techniques in improving performance of modern databases. In order to do so, we compare the measured CPI of the applications with the non-overlapped CPI using stall and execution cycles. The intent is that if OOO execution is effective, then individual components of the non-overlapped CPI should be overlapped and measured CPI should be much lower than non-overlapped CPI. Figure 9 shows the results for different databases aggregated over all the YCSB workloads. The stacked bars include stalls corresponding to pipeline resources, memory and computation cycles. We can observe that the measured CPI is lower than the non-overlapped stalls in all cases, measuring around 53-80% of the non-overlapped CPI. Similar observations hold for individual YCSB workloads as well. MongoDB has the least overlapping of stall components among all the databases as it experiences significantly higher number of cache misses than the other databases. Cassandra, on the other hand, shows the highest overlapping. In general, OOO execution is effective at hiding the non-overlapped CPI components to achieve a lower actual CPI. In contrast, actual CPI of the SPEC applications is 45-55% of the non-overlapped CPI. Thus, we can conclude that although OOO execution and speculation are less effective for the SQL/NoSQL databases than SPEC-like applications, they are still effective at improving performance of databases like Cassandra and VoltDB.

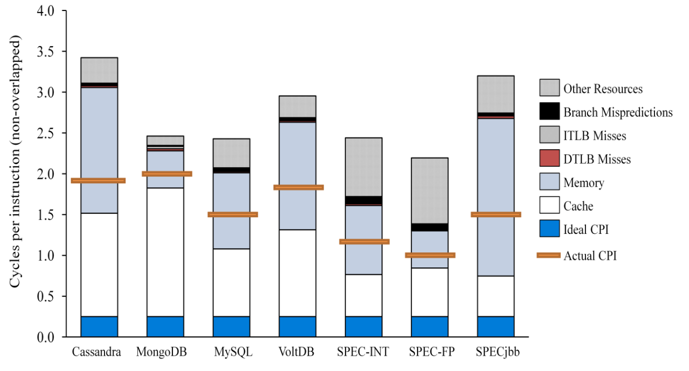

Fig. 9: Non-overlapped and Actual CPI for database workloads

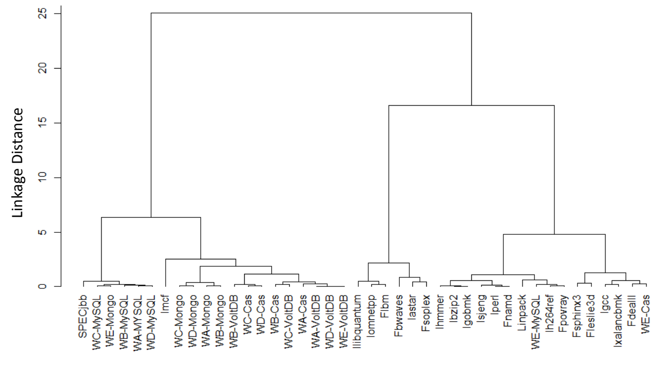

Fig. 10: Dendrogram comparing databases with SPEC, SPECjbb and Linpack benchmarks based on cache performance

D. *Workload Similarity Analysis*

In this section, we study the similarities between data-serving applications and SPEC, SPECjbb and Linpack benchmarks. Our analysis shows where different workloads lie in the workload space and can help performance analysts to choose representative benchmarks while evaluating future hardware designs targeting database applications.

We begin by comparing the applications based on cache and memory performance related characteristics only. Figure 10 shows the dendrogram plot. The horizontal scale lists all the applications and the vertical scale corresponds to the linkage distances between the applications indicating relative similarity. Shorter linkage distance implies greater similarity. We can see that database workloads form a separate cluster from SPEC benchmarks with the exception of Workload E (Cassandra and MySQL). Cache/memory performance of data-serving applications is closest to the SPECjbb benchmark and *mcf* benchmark from SPEC CPU2006. However, once all the characteristics are considered, mcf and SPECjbb move away from the database cluster as shown in Figure 11. No single non-database benchmark comes close in representing the full spectrum of database applications. Cassandra's scan functionality (WE) has a lot of associated overhead and executes 10-20x more instructions than other operations and databases. As a result, it's behavior is significantly different from other database workloads and it is the only database program grouped in the SPEC cluster.

Putting it all together, modern databases suffer from several bottlenecks which limit their overall performance on contemporary hardware systems. This is exemplified in the kiviat diagram plots in Figure 12 for a few selected work-

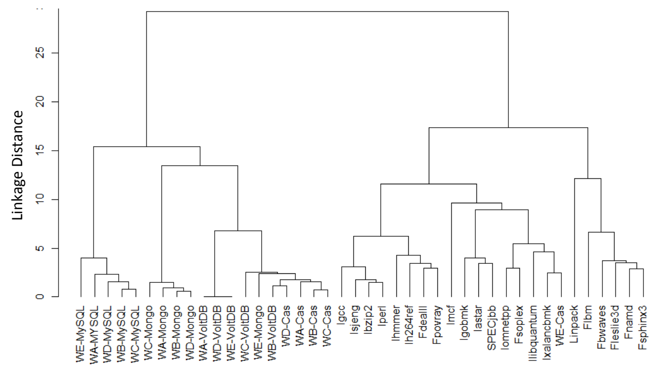

Fig. 11: Dendrogram comparing databases with SPEC, SPECjbb and Linpack benchmarks based on overall performance

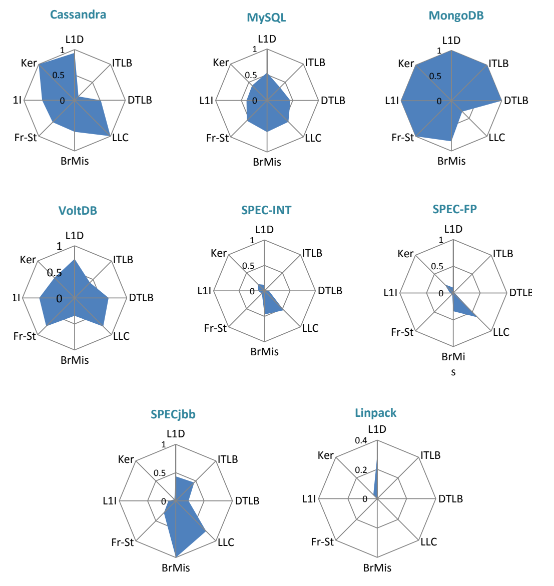

Fig. 12: Kiviat diagrams of the representative workloads. Database runs are for workload A. [Note the change of scale for Linpack]

loads. The kiviat plots are based on selected raw performance metrics (L1D, L1I, LLC, I/D TLB MPKI, %kernel instructions executed (Ker), branch misprediction rate (BrMis), fraction of front-end stalls(Fr-St)), normalized by their maximum observed values. The plots illustrate significant diversity in the performance and bottlenecks of different databases. We can also observe that as compared to their SQL counterparts, NoSQL databases suffer from worse performance inefficiencies. For example, NoSQL databases have significantly worse instruction/data cache and TLB performance than SQL databases for the same operations. MongoDB, in spite of executing the least number of instructions for every workload, suffers from the highest cache miss rates among all the databases. It also executes a significantly greater fraction of OS instructions. MySQL executes more instructions than MongoDB for same number of database operations, yet its design and implementation allows it to exploit the underlying hardware more efficiently. This also results in better performance of MySQL compared to NoSQL databases for several workloads. Many recently proposed big-data benchmarking suites [8], [9] include at most a single (or two) database run(s) to represent the entire class of data-serving applications. Our evaluation shows that this is not enough to represent the entire class of data-serving applications. It is also interesting to note that even though SPECjbb has IPC values closer to MongoDB and Cassandra, it stresses a different set of system components (branch misprediction rate and LLC cache misses) which lead to its overall poor performance. Even with a comparable dataset size (over 10GB), Linpack does not encounter similar memory subsystem issues as the database applications, demonstrating that Linpack program behavior is different from databases even when the dataset is big.

E. *Overall Database Performance and Scalability*

After analyzing the microarchitectural characteristics, we finally evaluate the overall performance and scalability of NoSQL and SQL databases by varying the number of client threads (see Figure 13). We can observe that MongoDB performs much worse than the other databases for all YCSB workloads. MySQL offers the best throughput and latency for workloads A, B and C, and it is outperformed by Cassandra and VoltDB with 64 client threads only. However, MySQL performs significantly worse (in terms of throughput and latency) than the other databases for the insert workload D. MySQL database has more constraint checking for adding data elements and has greater overhead in creating and maintaining relational meta-data (which also translates to higher instruction counts). VoltDB and Cassandra have comparably good performance for workloads A/B/C and perform better than both MySQL and MongoDB for insert operations. Cassandra is more optimized for write-intensive workloads and it is evident in its better performance for the write-heavy workload. For scan operations, all four databases experience higher execution latencies. Cassandra has especially poor scan performance as compared to the other databases. Cassandra did not originally support scan operations, but later added the functionality. From its poor performance, it appears that the scan capability of Cassandra was added with much overhead. VoltDB offers the best scan performance. Summarizing, although NoSQL databases are designed for optimized key-value stores, we observe that SQLs outperform NoSQLs for most operations and are beaten by NoSQLs only in a few cases (read-intensive and insert workloads at higher thread counts). Similar obser-

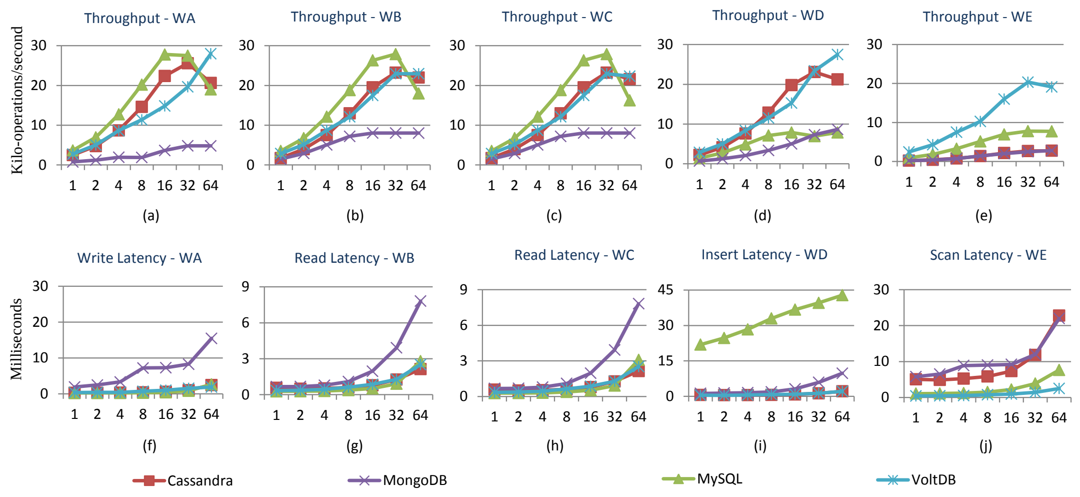

Fig. 13: Throughput and Latency graphs: (a, f) - WA, (b, g) - WB, (c, h) - WC, (d, i) WD, and (e, j) WE

vations are also made in [20]). NoSQL databases suffer from greater performance inefficiencies (poor I/D cache and TLB performance), which limits their performance as compared to their SQL counterparts. Nonetheless, NoSQL databases are promising and competitive alternatives to SQL-style databases, still they are yet to be optimized to fully reach the performance of contemporary SQL databases.

V. Related Work

Several prior studies [1], [2], [3], [21], [22], [23], [24] have evaluated the performance of OLTP applications. But they were performed on machines of more than 12 to 15 years ago. Significant advances have taken place in computer system designs since then. Moreover, the big-data revolution has caused a surge in the number and diversity of data management applications. In its light, we evaluate the performance of four popular NoSQL/SQL databases on modern computer systems.

Few recent research studies [8], [9], [10], [25] have evaluated the microarchitectural performance of big-data applications. Cloudsuite [8] and BigDataBench [9] present a range of big-data applications and evaluate them using an extensive set of microarchitectural parameters. However, they restricted the representation of the data-serving class of applications using one or two databases running at most a single operation. This paper expands performance characterization of data-serving databases to greater detail and to the best of our knowledge, no prior work has compared the hardware performance implications of different NoSQL/SQL databases.

Many researchers [26], [27] have compared SQL and NoSQL systems qualitatively by exploring their respective technologies and impact. Several other research studies [26], [20], [14], [28], [29], [30] have compared client-side execution time performance of SQL and NoSQL systems, making observations on their overall performance at a higher level. But there is no clear consensus among the studies regarding which system (SQL or NoSQL) is better.

VI. Conclusion

In this paper, we evaluated four modern NoSQL/SQL databases: Cassandra, MongoDB, MySQL and VoltDB in order to analyze their performance characteristics on contemporary processors. In general, we found that cache and memory system are the biggest limiters of database performance. Databases commonly exhibit data access patterns and working set sizes that reduce performance of the data caches and processor back-end. We also observed that as compared to other application classes, data-serving applications suffer from significantly poor front-end performance. What is most interesting to observe is that even though database performance is severely limited by the memory subsystem, dynamic execution techniques (e.g. out-of-ordering and speculative execution) help in hiding a significant fraction of the overall stalls. We identify several such performance bottlenecks and make recommendations to improve database performance efficiency.

We also showed that NoSQL databases face greater bottlenecks (poor I/D cache and TLB performance) as compared to their SQL counterparts. As a result, although NoSQL databases are designed for optimized key-value stores, SQL databases still outperform NoSQL databases for most operations and are beaten by NoSQL databases only in a few cases. Nonetheless, NoSQL databases are promising, competitive alternatives to SQL-style databases, however, NoSQLs are yet to be optimized to fully reach the performance of contemporary SQL databases.

Also, as significant performance variability exists between different database implementations even for the same operations, we conclude that multiple benchmarks are necessary to encapsulate the full performance spectrum of data-serving workloads. We also compared SQL/NoSQL databases with three popular benchmark suites. Cache/memory performance of data-serving applications is closest to the SPECjbb benchmark and *mcf* benchmark from SPEC CPU2006.

VII. Acknowledgement

The researchers are supported partially by SRC under Task ID 2504, National Science Foundation (NSF) under grant numbers 1337393, 1218474 and 1117895, Oracle, AMD and Huawei. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or other sponsors.

References

[1] K. Keeton, D. A. Patterson, Y. Q. He, R. C. Raphael, and W. E. Baker, "Performance characterization of a quad pentium pro smp using oltp workloads," in *ISCA*, Washington, DC, USA, 1998, pp. 15–26.
[2] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood, "Dbmss on a modern processor: Where does time go?" in *VLDB*, 1999, pp. 266–277.
[3] P. Ranganathan, K. Gharachorloo, S. V. Adve, and L. A. Barroso, "Performance of database workloads on shared-memory systems with out-of-order processors," in *ASPLOS*, 1998, pp. 307–318.
[4] "Cassandra," wiki.apache.org/cassandra/FrontPage.
[5] "MongoDB," mongodb.org.
[6] "MySQL," http://www.mysql.com.
[7] "VoltDB," http://voltdb.com.
[8] M. Ferdman *et al.*, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," in *ASPLOS*. New York, NY, USA: ACM, 2012, pp. 37–48.
[9] W. Gao *et al.*, "Bigdatabench: a big data benchmark suite from web search engines," *CoRR*, vol. abs/1307.0320, 2013.
[10] Z. Jia, L. Wang, J. Zhan, L. Zhang, and C. Luo, "Characterizing data analysis workloads in data centers," *CoRR*, vol. abs/1307.8013, 2013.
[11] "SPEC CPU2006," https://www.spec.org/cpu2006.
[12] "SPECjbb 2005," https://www.spec.org/jbb2005/.
[13] "HP Linpack," http://icl.eecs.utk.edu/hpl/.
[14] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *SoCC*, 2010, pp. 143–154.
[15] G. Dunteman, *Principal Component Analysis*. Sage Publications, 1989.
[16] L. Eeckhout, H. Vandierendonck, and K. D. Bosschere, "Workload design: Selecting representative program-input pairs," *PACT*, vol. 0, p. 83, 2002.
[17] A. Phansalkar, A. Joshi, L. Eeckhout, and L. John, "Measuring program similarity: Experiments with spec cpu benchmark suites," *ISPASS*, vol. 0, pp. 10–20, 2005.
[18] "Mongodb architecture guide," http://www.mongodb.com.
[19] "Linux perf tool," https://perf.wiki.kernel.org/index.php/Main_Page.
[20] A. Floratou, N. Teletia, D. J. DeWitt, J. M. Patel, and D. Zhang, "Can the elephants handle the nosql onslaught?" *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1712–1723, Aug. 2012.
[21] M. Shao, A. Ailamaki, and B. Falsafi, "Dbmbench: Fast and accurate database workload representation on modern microarchitecture," in *CASCON*, 2005, pp. 254–267.
[22] R. Hankins *et al.*, "Scaling and characterizing database workloads: Bridging the gap between research and practice," in *MICRO*, 2003, pp. 151–162.
[23] S. Manegold, P. A. Boncz, and M. L. Kersten, "Optimizing database architecture for the new bottleneck: Memory access," *The VLDB Journal*, vol. 9, no. 3, pp. 231–246, Dec. 2000.
[24] S. H. Anastassia and A. Ailamaki, "Stageddb: Designing database servers for modern hardware," in *In IEEE Data*, 2005, pp. 11–16.
[25] A. Yassin, Y. Ben-Asher, and A. Mendelson, "Deep-dive analysis of the data analytics workload in cloudsuite," in *IISWC*, 2014, pp. 202–211.
[26] M. Indrawan-Santiago, "Database research: Are we at a crossroad? reflection on nosql," in *NBIS*. IEEE Computer Society, 2012, pp. 45–51.
[27] D. Bartholomew, "Sql vs nosql," *Linux Journal*, p. 195, July 2010.
[28] A. Boicea, F. Radulescu, and L. I. Agapin, "Mongodb vs oracle – database comparison," in *EIDWT*. IEEE Computer Society, 2012, pp. 330–335.
[29] A. Pavlo *et al.*, "A comparison of approaches to large-scale data analysis," in *SIGMOD*. ACM, 2009, pp. 165–178.
[30] S. M. Y. Li, "A performance comparison of sql and nosql databases," in *PACRIM*, August 2013.