A Demonstration of Rubato DB: A Highly Scalable NewSQL Database System for OLTP and Big Data Applications

Li-Yan Yuan
Department of Computing Science
University of Alberta
yuan@cs.ualberta.ca

Lengdong Wu
Department of Computing Science
University of Alberta
lengdong@cs.ualberta.ca

Jia-Huai You
Department of Computing Science
University of Alberta
you@cs.ualberta.ca

Yan Chi
Shanghai Shifang Software
chiyan@rubatodb.com

ABSTRACT

We propose to demonstrate Rubato DB, a highly scalable NewSQL system, supporting various consistency levels from ACID to BASE for big data applications. Rubato DB employs the staged grid architecture with a novel formula based protocol for distributed concurrency control. Our demonstration will present Rubato DB as one NewSQL database management system running on a collection of commodity servers against two of benchmark sets.

The demo attendees can modify the configuration of system size, fine-tune the query workload, and visualize the performance on the fly by the graphical user interface. Attendees can experiment with various system scales, and thus grasp the potential scalability of Rubato DB, whose performance, with the increase of the number of servers used, can achieve a linear growth for both OLTP application with the strong consistency properties and key-value storage applications with the weak consistency properties.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Concurrency, Transaction processing*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems*

Keywords

Scalability; Concurrency Control; ACID; BASE

1. INTRODUCTION

Today, data is flowing into organizations at an unprecedented scale in the world. The ability to scale out for processing an enhanced workload has become an important factor for the proliferation and popularization of data management system. The pursuit for tackling the big data challenges has given rise to a plethora of data management systems characterized by high scalability. Diverse systems for processing big data explore various possibilities in the infrastructure design space. The trend of using lower-end, commodity servers to scale out configurations has become popular, due to the drop in prices for the hardware and the improvement in performance and reliability.

A notable phenomena is the NoSQL movement that began in early 2009 and is growing rapidly. NoSQL systems are characterized as simplified highly scalable systems addressing properties of schema-free, simple-API, horizontal scalability and relaxed consistency [2, 5, 8].

The recently proposed NewSQL systems (relative to NoSQL) aim to achieve the high scalability and availability same as NoSQL, while preserving the ACID properties for transactions and complex functionality of relational databases [11]. NewSQL database systems are appropriate in applications where traditional RDBMS have been used, but requiring additional scalability and performance enhancement.

According to the core requirement differences between NoSQL systems and NewSQL systems, there are challenges in the implementation of NewSQL systems in the following aspects:

- **Data model.** NoSQL systems usually merely support non-relational data model such as key-value stores [5] and Bigtable-like stores [1]. NoSQL systems represent a recent evolution by making trade-off between scalability and complexity of the data model. Thus the implementation of NewSQL systems faces with the first issue:

  Is the complex relational data model an obstacle of scalability?

- **Consistency model.** The NoSQL systems represent an evolution in building scalable infrastructure by sacrificing strong consistency and opting for weak consistency [2, 5, 8]. However, strong consistency levels such as ACID properties are essential for NewSQL systems to manage critical data which requires both liveness and safety guarantees. The implementation of NewSQL systems meets the second challenge:

  Is it possible to achieve high scalability with ACID?

- **Architecture.** Traditional database vendors employ new techniques to explore the scalability of the symmetric multiple processing (SMP) architecture and the massively parallel processing (MPP) architecture. However, because of the inherent deficiencies due to the resources contention, the scalability of such architectures is not comparable with NoSQL systems. Some NoSQL systems based on the Bigtable [1] and MapReduce framework [4] utilize the shared-nothing infrastructure to provide high scalability. A set of systems with high-level declarative languages, including Yahoo! Pig [9] and Facebook Hive [13], are realized to compile queries into the MapReduce framework on the Hadoop platform.

  To achieve high performance and scalability, innovative software architecture should be applied to NewSQL systems, then we need to consider:

  Is it possible to scale out commonly used single server database system design?

We have developed a highly scalable NewSQL system, called Rubato DB [17] with satisfactory solutions for the challenges as mentioned above, by applying the principles as following:

- Exploring the close integration of data partitioning and relational data model to eschew data skew for transactions.
- Obtaining high scalability with the MapReduce [4] sevice-oriented architecture.
- Scaling out the concurrency control protocol for ACID based on timestamps rather than any locking mechanism.
- Overcoming the weakness of BASE through stronger consistency models such as BASIC [16].

The implementation of Rubato DB combines and extends the ideas from:

> hybrid data partitioning, staged-event driven architecture (SEDA) [15], data-intensive MapReduce framework [4], and distributed timestamp-based concurrency control;

and using innovative technologies such as

> Formula based protocol, software instruction, lazy loading, dynamic timestamp ordering, multiple communication protocols.

Rubato DB is a highly scalable NewSQL database system running on a collection of commodity servers that supports various consistency guarantees[1] including the traditional ACID and BASE, and a novel consistency level in between, i.e., BASIC [16]; and conforms with the SQL2003; and has been used in some commercial applications.

The proposed demo is to demonstrate and verify the performance and scalability of Rubato DB under both TPC-C benchmark [14] and YCSB benchmark [3]. More specifically, the demonstration will allow demo attendees to try out Rubato DB for the following objectives:

[1]The name Rubato, (from Italian rubare, "to rob"), in music, means subtle rhythmic manipulation and nuance in performance. It describes the practice of Rubato DB for supporting various consistencies with freedom.

1. **Feasible Configuration**. To deploy Rubato DB on a collection of commodity servers, ranging from 1 to 16.
2. **Data Loading**. To load large size of data distributed across numbers of nodes automatically according to pre-defined data partitioning schema.
3. **High Scalability**. To demonstrate the linear scalability of Rubato DB for typical OLTP applications requiring the ACID properties under TPC-C benchmark.
4. **Various Consistency Levels**. To compare performances of Rubato DB with either BASE or BASIC properties and other NoSQL systems with BASE properties under YCSB benchmark.

A friendly Graphical User Interface will be provided in the proposed demo to enable demo attendees to choose different experiments and deployment configurations, and to display performance results and internal working loads.

The rest of the paper is organized as follows. Section 2 describes the architecture of Rubato DB and some of its key features; and Section 3 outlines the experiment setups and the demonstration we plan to show. The conclusion is in Section 4.

2. OVERVIEW OF RUBATODB

Rubato DB is a scalable NewSQL database system supporting relational data model for data-centric applications. The main architectural components of Rubato DB are depicted in Figure 1 [17]. We firstly give a brief overview of the several essential Rubato DB components as below, before describing the contents of our demonstration.

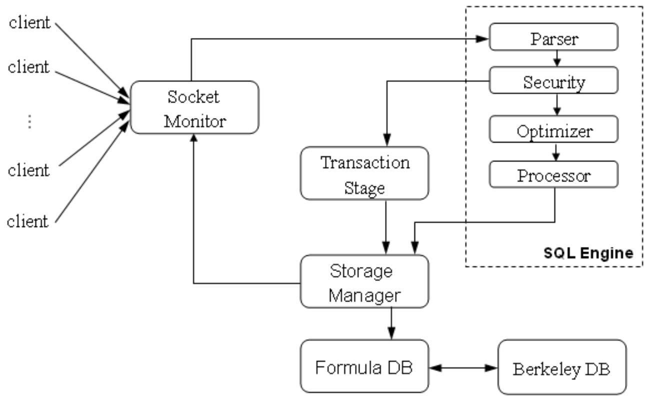


Figure 1: RubatoDB system architecture

2.1 Socket Monitor

Rubato DB runs on a collection of servers as one database management system, with a single socket monitor to establish and record the connection states for all client requests.

The socket monitor adopts a loading control schema, called *lazy loading*, to reduce data contention and to minimize unnecessary rollbacks. The socket monitor maintains two lists of clients: a list of active clients with active transactions, and a list of requesting clients whose requests are waiting to be evaluated. The socket monitor will not process any request when the oldest waiting transaction is not among the oldest 20% ones base on the following practical principle:

If all the current requests have higher potential to conflict with other requests, the system should rather wait awhile for new requests with lower conflict potential to arrive.

2.2 SQL Engine

Rubato DB's SQL engine is used to process all the SQL queries, including aggregate functions and nested queries, updates, and all other requests according to SQL2003.

The SQL engine is composed as a set of staged grid modules, each of which is a self-contained software module consisting of an incoming request queue, as illustrated in Figure 2. Threads within each staged grid module operate by pulling a sequence of requests, one at a time, off the input queue, invoking the application-supplied event handler (e.g. parser, optimizer, query, update, etc) to process requests, and dispatching outgoing results by enqueuing them on the request queues of outgoing staged grid module, located either in the same server or another server within LAN. Each request to the system will be processed in a relay-style by being passed from one staged grid module to the next one until it is completed. Both parallelism and pipeline execution are supported by the engine.

A set of software instructions are employed to carry the operation's backpack with its private state and data. The instruction with a uniform format is the only packet flowing through different staged grid modules. To improve the performance, multiple communication protocols are utilized depending on the source and destination locations of stages and/or system resources.

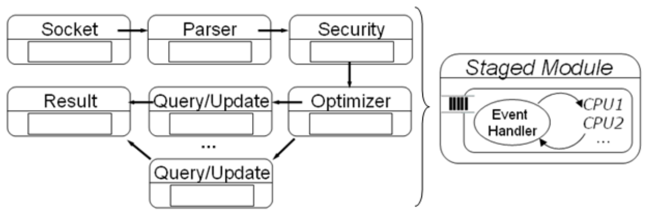


Figure 2: Staged Modules of SQL Engine

2.3 Transaction Manager

The transaction manager, consisting of *Transaction stage* and *Formula DB*, is responsible for coordinating data access on multiple nodes based on a novel formula protocol for concurrency (FPC) to ensure serializability. It is the transaction manager that performas all the transactional operations, including pre-commit (necessary for distributed concurrency), commit, and rollback.

The FPC is a novel implementation of the classical Multiversion Timestamp Concurrency Control Protocol [12], with two distinct features:

1. Instead of using multiple versions of updated data items, FPC uses formulas stored in memory (associated with the updated data items) to represent the multiple values of updated data items, which will reduce overhead of storing multiple data versions.

2. The timestamp ordering of transactions may be altered to allow a transaction with older timestamp to read the data item updated by a later transaction, as long as the serializability is respected, which will increase the degree of concurrency.

The two parts of the transaction manager perform their respective responsibilities:

(a) *Transaction stage.* A dedicated stage grid module that is located in every grid server and is responsible for all the basic transactional operations including *pre_commit*, *commit* and *rollback* of the transaction manager.

(b) *Formula DB.* A thread-free layer on the top of the Berkeley DB such that all disk accesses in Rubato DB are through Formula DB.

Three different consistency levels are supported by Rubato DB:

1. **ACID.** The strongest end of the consistency spectrum for the transactional functionalities.

2. **BASE.** The most notable weak consistency model used by NoSQL systems [2, 5, 8]. The BASE can be summarized as: the system responses basically all the time (Basically Available), is not necessary to be always consistent (Soft-state), but has to come to a consistent state eventually (Eventual consistency) [10].

3. **BASIC.** Rubato DB is not limited to merely ACID that is too strong and BASE which is too weak, but rather supports BASIC, a spectrum between these two extreme. BASIC stands for Basic Availability, Scalability, Instant Consistency [16]. BASE and BASIC provide different choices between the model that is faster but requiring extra efforts to deal with inconsistent results and the model that delivers consistent results but is relatively slower with higher latency.

Figure 3 illustrates the various consistency levels supported by Rubato DB together with different tolerance for fault or network partition.

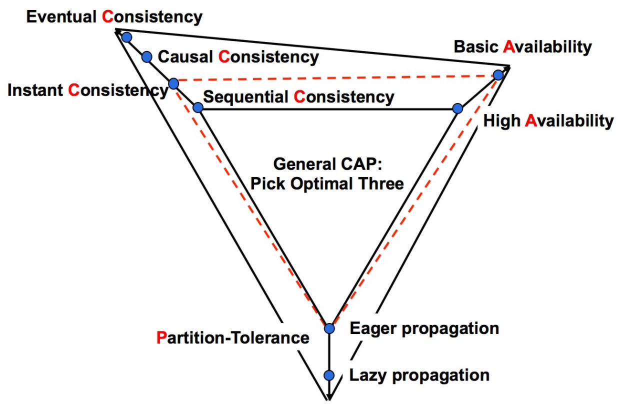


Figure 3: Generalized CAP Theorem [16]

2.4 Storage Manager

Rubato DB uses a hybrid data storage partition solution to allow a table to be partitioned vertically and horizontally and stored separately over different grid nodes.

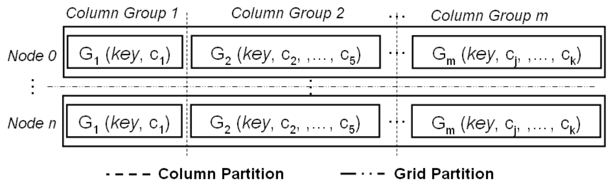


Figure 4: Hybrid partition of Storage Manager

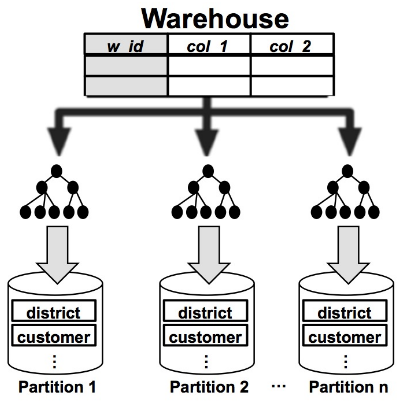


Figure 5: Grid partition illustration

Storage manager uses Berkeley DB for all disk accesses to direct attached storages (DAS). All tables and their partitions if any are stored on local disk as a Berkeley DB file. To obtain maximum performance, when a table is created according to the schema, the storage manager provides facilities for users to specify fine-gained hybrid storage partitioning based on application semantics and query workload.

As depicted in Figure 4, the column partition enables users to store a single relation as a collection of disjoint (none-key columns) vertical partitions of different groups. Columns frequently accessed together are clustered into the same frequent column group, while columns seldom accessed are categorized into static column groups. Compression can be also applied to take the benefits of column-oriented layout. The grid partition is used to deploy data on multiple nodes and thus provides scale-out capability.

Rubato DB applies a tree-based schema for grid partitioning, as demonstrated in Figure 5. Tuples in every descendent table are partitioned according to the ancestor that they descend from. As a consequence, data accessed within a transaction will be located in the same data node [6, 7]. This tree structure implies that corresponding to every row in the root table, there are a group of related rows in the descendant tables. All rows inherent from the same root are guaranteed to be co-located.

3. DEMONSTRATION

The proposed demo is to demonstrate and verify the performance of Rubato DB under both TPC-C and YCSB benchmark tests [3, 14]. The demonstration will allow demo attendees to try out Rubato DB for the following objectives:

1. **Feasible Configuration**. How to deploy Rubato DB on a collection of commodity servers?
2. **Data Loading**. How to load large size of data distributed across numbers of nodes automatically according to the predefined data partitioning schema?
3. **High Scalability**. What is the scalability of Rubato DB for the typical OLTP applications requiring the ACID properties?
4. **Various Consistency Levels**. What are performance comparisons between Rubato DB with either BASE or BASIC properties and other NoSQL systems with BASE property for big data applications?

A graphical user interface for Rubato DB will be provided to facilitate users to explore Rubato DB.

3.1 Setup Details

All the demo will be conducted on a collection of four (4) commodity servers with 4 quad-core Intel Xeon CPUs, 64 GB of main memory, SATA disks configured in RAID0, running Linux Ubuntu 12.04 LTS.

For simplicity, each server may be used as 1 to 4 nodes, that is, Rubato DB may be configured and then deployed on all four servers as 16 nodes such that all nodes will be connected only through the TCP/IP protocol.

Our demo is based on two well-known representative benchmarks. We use the TPC-C [14] benchmark's set of transactions, database schema, and pre-generated data as our target domain for demo about OLTP workloads. All the demos conducted are according to the TPC-C specification. In addition, demo attendees can also take the Yahoo! Cloud Serving Benchmark (YCSB) [3] on Rubato DB. The number of operations per second will be reported with varying distribution of read/write operations.

In order to demonstrate most important features of Rubato DB within a limited time, we have pre-load all the initial databases as per the benchmark specifications.

For the TPC-C benchmark, we have populated 2000 warehouses, and all TPC-C tables are column-partitioned in to frequent and static column groups; and then grid-partitioned across different nodes according to a schema structure. The number of warehouses deployed is proportional to the system size chosen by the attendee.

For YCSB which is relatively simple, we define a multi-column structure for each record, which consists of one key column of integer type and 20 data columns of text type. We pre-load 100 million 1KB records on each node, resulting 120GB of raw data per node.

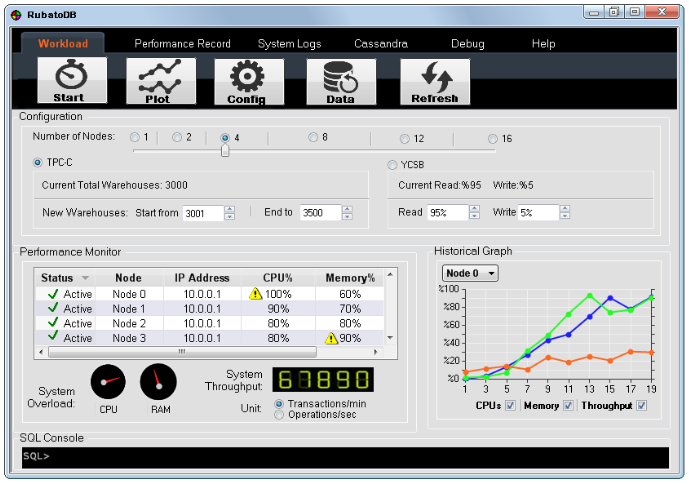
3.2 Demonstration Details

The proposed demonstration will perform both TPC-C and YCSB benchmarks on various configurations, including the number of nodes (scalability), the number of warehouses (the work load for TPC-C), and weights in read/write (the workload for YCSB). The following table summarizes the results displayed for various tasks and/or configurations.

Task	Nodes	Work load	Display
TPC-C	1-16	100-2000 (warehouses)	tmpC CPU/Memory Usage
YCSB	1-12	0% - 10% (write operations)	Throughput Latency

3.3 User Interaction

To facilitate interaction with Rubato DB during the demonstration, a friendly graphical user interface is provided, as shown in the mock-up in Figure 6.



Number of Nodes: ◯ 1 ◯ 2 [✓ 4] ◯ 8 ◯ 12 ◯ 16

[✓ TPC-C] ◯ YCSB

Current Total Warehouses: 3000

New Warehouses: Start from 3001 End to 3500

Current Read:%95 Write:%5

Read 95% Write 5%

Status	Node	IP Address	CPU%	Memory%
✓ Active	Node 0	10.0.0.1	⚠ 100%	60%
✓ Active	Node 1	10.0.0.1	90%	70%
✓ Active	Node 2	10.0.0.1	80%	80%
✓ Active	Node 3	10.0.0.1	80%	⚠ 90%

System Overload: CPU RAM

System Throughput: 67890

Unit: [✓ Transactions/min] ◯ Operations/sec

CPUs ✓ Memory ✓ Throughput ✓

SQL>

Figure 6: Graphical User Interface of Rubato DB

The interface consists of the following panels.

1. Configuration Panel:

   This panel enable users to choose

   (a) the demo tasks: TPC-C or YCSB tests;

   (b) the number of nodes, from 1 to 16 nodes;

   (c) the workload: the number of warehouses for TPC-C and the read/write weight for YCSB;

   (d) the different consistency levels: ACID, BASIC, BASE.

2. Performance Monitor Panel:

   This panel displays the demo performance, including the number of transactions/minute (TPC-C) or the number of operations/second (YCSB), the CPU time and the memory usage, and the running status of each node.

3. Historical Graph Panel:

   It is used to visualize the change curve of throughput as well as the CPU time and memory usage of each node, as specified by the user.

4. SQL Panel:

   An SQL console is provided for users to query the Rubato DB as well to manipulate workload control.

3.4 Performance Expectation

Demo attendees can obtain the performance of Rubato DB using the GUI. Here we provide the expected performance results so that attendees can make the comparison and verify the features of Rubato DB.

Figure 7 demonstrates that the performance (tpmC) of the TPC-C benchmark test of Rubato DB increases from 5000 concurrent clients (500 warehouses) running on a system instance with 1-node, to 67,000 concurrent clients (6700 warehouses) running on a system instance with 16 nodes. The result under the TPC-C benchmark test clearly shows that:

> **The performance of Rubato DB scales up linearly with the increase of the number of nodes deployed.The rollback ratio is stable at a low level.**

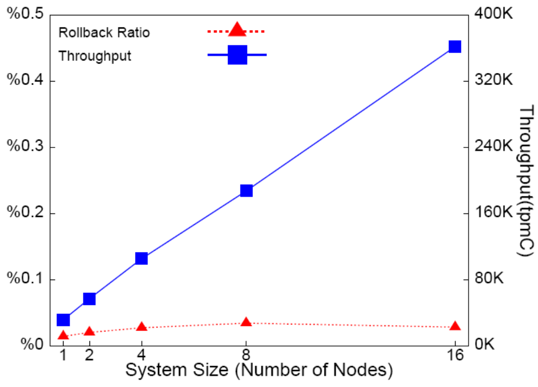


Figure 7: Expected Results of TPC-C Benchmark

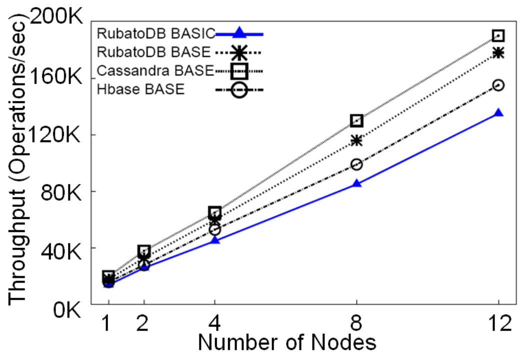


Figure 8: Expected Throughout Comparison of YCSB

For the results of YCSB, though users can set any proportion for read/write operation, we present the performance for a typical read-intensive workloads: read intensive (90% read and 10% write operations).

Also, we present the comparison with Cassandra[2] [8]: an open-source key-value store clone of Dynamo [5], HBase[3]: an open-source bigtable-like system [1]. The experiments are conducted with the number of nodes as 1, 2, 4, 8 and 12. This result sets shown in Figure 9 can clearly demonstrate that:

> **The performance and scalability of Rubato DB is comparable with that of other popular scalable NoSQL system.**

By demonstrating the performance of Rubato DB with BASIC properties, we can also show that RubatoDB with

[2]http://cassandra.apache.org/
[3]http://hbase.apache.org/

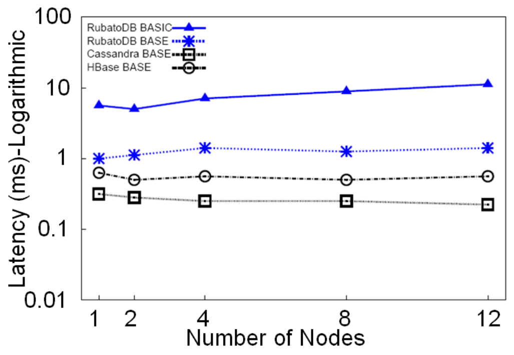


Figure 9: Expected Latency Comparison of YCSB

BASIC properties still preserves near-linear scalability with increasing throughput and flat latency, same as systems with BASE. We can conclude that:

> **The cost induced by BASIC is acceptable comparing with the extra efforts needed to manipulate the inconsistent soft states for BASE. BASIC pays a reasonable price for a higher consistency than BASE.**

4. CONCLUSION

We present a demonstration of Rubato DB, a scalable NewSQL system, showcasing the main feature of scalability with various consistency properties from ACID to BASE, which provides a positive answer to the question on the trade-off between scalability and consistency.

The development of NewSQL database systems is a topic with strong interest over recent years and with a great potential impact on data management, especially in the face of big data challenges. With experiments and explorations on our pioneering Rubato DB system, we are exploiting road for the design and implementation of the new generation of database systems.

5. REFERENCES

[1] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems*, 26(2):4, 2008.

[2] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proceedings of the VLDB Endowment*, 1(2):1277–1288, 2008.

[3] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proc. the 1st ACM Symposium on Cloud computing*, pages 143–154. ACM, 2010.

[4] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[5] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review*, 41(6):205–220, 2007.

[6] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden. Hyrise: a main memory hybrid storage engine. *Proc. the VLDB Endowment*, 4(2):105–116, 2010.

[7] E. P. Jones, D. J. Abadi, and S. Madden. Low overhead concurrency control for partitioned main memory databases. In *Proc. the 2010 ACM SIGMOD International Conference on Management of Data*, pages 603–614. ACM, 2010.

[8] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[9] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proc. the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1099–1110. ACM, 2008.

[10] D. Pritchett. Base: An acid alternative. *Queue*, 6(3):48–55, 2008.

[11] M. Stonebraker. New opportunities for new sql. *Communications of the ACM*, 55(11):10–11, 2012.

[12] R. Thomas. A solution to the concurrency control problem for multiple copy databases. In *Digest of papers IEEE COMPCON spring*, pages 56–62, 1984.

[13] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: a warehousing solution over a map-reduce framework. *Proc. the VLDB Endowment*, 2(2):1626–1629, 2009.

[14] TPC-C. http://www.tpc.org/tpcc/. 2010.

[15] M. Welsh, D. Culler, and E. Brewer. Seda: an architecture for well-conditioned, scalable internet services. *ACM SIGOPS Operating Systems Review*, 35(5):230–243, 2001.

[16] L. Wu, L. Yuan, and J. You. Basic, an alternative to base for large-scale data management system. In *2014 IEEE International Conference on Big Data*, 2014.

[17] L. Yuan, L. Wu, J. You, and Y. Chi. Rubato db: a highly scalable staged grid database system for oltp and big data applications. In *Proc. the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 1–10. ACM, 2014.