I-Store: Data Management for Fast Networks

Carsten Binnig, Ugur Cetintemel, Tim Kraska, Stan Zdonik,
Erfan Zamanian, Andrew Crotty
Brown University

1. INTRODUCTION

Motivation: Existing distributed data management systems typically assume that the network is a major bottleneck [10]. Consequently, avoiding remote data transfers is an important design aspect of existing systems. In extreme cases, this has lead to system designs, which explicitly do not support certain distributed operations (e.g., BigTable only supports joins if the inner table contains less than 8 MB of data). A common design principle, however, is to minimize remote data transfer by the two following techniques: First, existing systems try to find an optimal partitioning scheme to co-partition data in order to avoid remote data transfers for operations such as joins or to avoid distributed transactions. Second, locality-aware scheduling strategies aim to increase data-locality by shipping computation to the nodes where the data is stored. However, these techniques still result in major limitations: (1) For complex data models (e.g. graphical models) as well as complex operations expressed as user-defined functions (e.g., machine-learning tasks), partitioning the data optimally is not straightforward. (2) Existing partitioning schemes are designed for static workloads. Thus, ad-hoc workloads are not well-supported resulting often in a poor query performance. (3) Load balancing as well as adaptive parallelization that are necessary to optimally execute complex and long-running analytical programs become expensive operations since intermediate results might need to be re-distributed over the slow network connections.

With modern RDMA-capable networks such as Infiniband FDR/EDR the before mentioned design decisions become obsolete. Modern interconnects allow data to transfer across machines almost as fast as from the CPU to memory. For instance, DDR3 memory bandwidth currently ranges from 6.25 GB/s (DDR3-800) to 16.6 GB/s (DDR3-2133) for each memory channel [8], whereas Infiniband has a specified bandwidth of 1.7 GB/s (FDR 1×) to 37.5 GB/s (EDR 12×) [3] for each network port (see Figure 2).

Furthermore, recent advances in RDMA allow to send/read data directly to/from a remote host without involving the remote CPU at all and thus free up the CPU for other tasks.

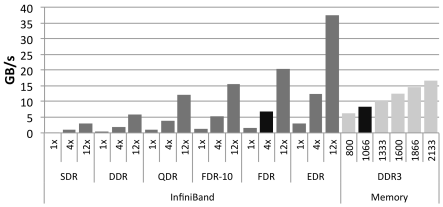

Figure 1: DDR3 memory vs. Infiniband bandwidth

With Intel Data Direct I/O technology [2], it is even possible to transfer data from one machine directly into the CPU cache of another. Thus, instead of finding an optimal partitioning scheme and shipping the computation to the data, a design where data is shipped to the computation is getting much more attractive since the before mentioned issues can be efficiently addressed.

Contributions and Outline: In this paper, we revisit design decisions for distributed data management systems for OLTP workloads (Section 2) as well as for OLAP workloads (Section 3), under the assumption that communication between servers is essentially free.

2. OLTP FOR FAST NETWORKS

Main Ideas: As mentioned before, RDMA provides one-sided read and write operations that implement an efficient access to remote memory without involving the remote CPU. Moreover, atomic RDMA operations such as fetch and add or compare and swap, implement distributed atomic access to remote memory. Based on these properties, we suggest a novel database architecture for transactional workload where the complete transaction logic is implemented on the client side. This enables much better scalability and elasticity properties and prevents hot database nodes.

Initial Results: As a fist step, we implemented a distributed snapshot isolation schemes where clients execute the complete transaction logic. In order to execute a transaction, the client first retrieves the version information of the last committed transaction (i.e., the version that the transaction reads). This is implemented as a global counter using atomic RDMA operations. Afterwards, the client retrieves the correct versions of database objects from the database nodes using RDMA reads (and updates them locally in its memory). For committing the new versions, the client again uses atomic operations to implement a distributed lock. Figure **??** shows our initial results of the TPC-W buy confirm transac-

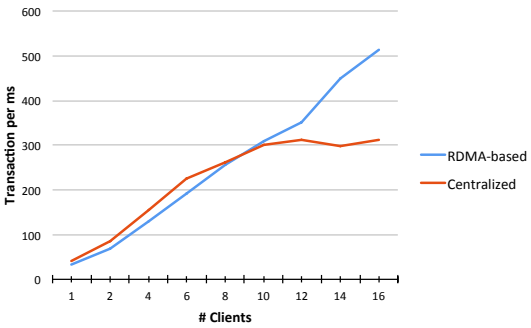

Figure 2: RDMA-based SI vs. Classical SI

tion using our RDMA-based snapshot isolation scheme and a classical centralized scheme where the transaction logic is implemented in the database. Compared to the centralized scheme, our scheme shows similar performance for a small number of clients but scales much better for an increasing number of clients.

3. OLAP FOR FAST NETWORKS

Main Ideas: The main focus of distributed query processing algorithms is to minimize network costs. Therefore, existing parallel query processing systems are typically implemented using a data parallel execution scheme (i.e., the same operation is executed over different partitions of the data) [6, 5] while re-partitioning operators which would require remote data transfers are minimized [11]. However, existing distributed database operations (e.g., joins) do not pay much attention of efficiently leveraging the caches of individual machines nor do they optimally load balance the execution over all nodes of the cluster. With fast RDMA-capable networks, where the network is no longer a bottleneck and an efficient remote memory access is available via RDMA, cache efficiency is becoming a relevant aspect for distributed database operations. Thus, as a major aspect we want revisit typical distributed database operations. One idea is to adapt parallel main-memory algorithms for query operators on single multi-core machines (e.g., [4]) that are much more efficient with regard to load balancing and cache usage.

Initial Results: As a first step, we implemented a distributed RDMA-based version of the radix join. A recent paper has shown that the radix join [4] is the main-memory join algorithm which has (in most cases) the best query performance on a single node. In our RDMA-based version, we make use of efficient one-sided RDMA write operations to implement the radix partitioning phase. Moreover, we use unsignaled RDMA writes in most cases, which means that the client does not need to wait for the confirmation. Only, the last RDMA write of the radix partitioning phase on each node is executed in a signaled way. Since, RDMA operations are not reordered on a connection, we know that all data has been written to the remote side if the last write operation succeeds. Our initial implementation shows, that our RDMA-based version of the radix join is as efficient as the local radix join on the same data sizes using the same computing power (i.e., number of threads).

4. RELATED WORK

There are different existing approaches that address the problem of distributed data management on fast networks with RDMA capabilities. FaRM [7], for example, exposes the memory of machines in a cluster as a shared address space and offer random memory access as well as transactional support. In contrast to FaRM, our storage manager does not support random remote memory access, but offers only restricted access patterns that can be used efficiently to implement typical distributed database operations.. RAMCloud [9] is designed from scratch for fast interconnects (i.e., Infiniband) but it solves a different problem; RAMCloud's goal is to lower the latency for key/value operations as much as possible rather than considering typical data management operations. For Oracle's cluster product, Oracle RAC, it is suggested to use Infiniband [1]. However, the query processing algorithms and the storage layer of Oracle RAC is neither build from scratch to optimally support fast networks nor does it support more complex analytical workloads.

5. REFERENCES

[1] Delivering Application Performance with OracleâĂŹs InfiniBand Technology. `http://www.oracle.com/technetwork/server-storage/networking/documentation/o12-020-1653901.pdf`, 2012.

[2] Intel Data Direct I/O Technology. `http://www.intel.com/content/www/us/en/io/direct-data-i-o.html`, 2014.

[3] I. T. Association. InfiniBand Roadmap. `http://www.infinibandta.org/`, 2013.

[4] C. Balkesen, G. Alonso, J. Teubner, and M. T. Özsu. Multi-core, main-memory joins: Sort vs. hash revisited. *PVLDB*, 7(1):85–96, 2013.

[5] A. Bezerra, P. Hernández, A. Espinosa, and J. C. Moure. Job scheduling for optimizing data locality in hadoop clusters. In *20th European MPI Users's Group Meeting, EuroMPI '13, Madrid, Spain - September 15 - 18, 2013*, pages 271–276, 2013.

[6] R. Chaiken, B. Jenkins, B. Larson, Per Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: Easy and efficient parallel processing of massive data sets. *Proceedings of VLDB Endowment*, 1(2):1265–1276, 2008.

[7] A. Dragojevic, D. Narayanan, M. Castro, and O. Hodson. FaRM: Fast Remote Memory. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014, Seattle, WA, USA, April 2-4, 2014*, pages 401–414, 2014.

[8] JEDEC. DDR3 SDRAM Standard. `http://www.jedec.org/standards-documents/docs/jesd-79-3d`, 2013.

[9] J. K. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, D. Ongaro, G. M. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman. The case for ramcloud. *Commun. ACM*, 54(7):121–130, 2011.

[10] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. Sparrow: distributed, low latency scheduling. In *SOSP*, pages 69–84, 2013.

[11] M. T. Ozsu. *Principles of Distributed Database Systems*. Prentice Hall Press, 3rd edition, 2007.