More Than A Network: Distributed OLTP on Clusters of Hardware Islands

Danica Porobic* danica.porobic@epfl.ch
Pınar Tözün† ptozun@us.ibm.com
Raja Appuswamy* raja.appuswamy@epfl.ch
Anastasia Ailamaki*‡ anastasia.ailamaki@epfl.ch

*École Polytechnique Fédérale de Lausanne
†IBM Almaden Research Center
‡RAW Labs SA

ABSTRACT

Multisocket multicores feature hardware islands - groups of cores that communicate fast among themselves and slower with other groups. With high speed networking becoming a commodity, clusters of hardware islands with fast networks are becoming a preferred platform for high end OLTP workloads. While behavior of OLTP on multisockets is well understood, multi-machine OLTP deployments have been studied only in the geo-distributed context where network is much slower. In this paper, we analyze the behavior of different OLTP designs when deployed on clusters of multisockets with fast networks.

We demonstrate that choosing the optimal deployment configuration within a multisocket node can improve performance by 2 to 4 times. A slow network can decrease the throughput by 40% when communication cannot be overlapped with other processing, while having negligible impact when other overheads dominate. Finally, we identify opportunities for combining the best characteristics of scale-up and scale-out designs.

1. INTRODUCTION

Online Transaction Processing (OLTP) systems typically run on the highest performing servers of the day that feature many processing cores with large main memory and storage capacities. Until recently, such machines had uniform processor topologies with a constant communication latency between any pair of CPU cores regardless of their location. With multisocket multicores, however, communication latency between two cores can differ by an order of magnitude depending on their location [7]. Soon, with dozens of cores on the same chip, the communication latencies within a chip will also become non-uniform.

At the same time, fast network interconnects are reaching main memory bandwidths. Technologies such as Remote Direct Memory Access (RDMA) allow applications to access memory on a remote machine without involving either the operating system or the processor. High speed fabrics that support RDMA, such as Infiniband and converged Ethernet with bandwidths up to 100Gbps, are already standard in supercomputers and high-end appliances.

Abundant parallelism and fast commodity networks are leading to the emerging class of commodity cluster computing platforms that will offer performance comparable to today's large shared memory machines. These platforms achieve high compute density at low power budget and cost by eliminating unnecessary system components using highly customized system-on-a-chip (SoC) nodes. Individual nodes, containing only processing cores, memory, and I/O interfaces communicate using low-latency interconnect fabrics [8]. Data management on these platforms is an area of ongoing research with promising proposals for general purpose distributed system architectures [10, 17].

OLTP systems are typically designed to either scale out or scale up. Scale-out systems employ shared-nothing designs that run on clusters of machines and offer very high performance for easily partitionable workloads due to explicit data partitioning [26]. Scale-up systems typically use shared-everything designs that focus on minimizing the number and duration of critical sections [9, 14, 28]. None of the designs, however, attempts to scale across both dimensions at the same time due to conflicting requirements.

A recent study in the multisocket multicore environment shows that different deployment configurations are optimal for different types of workloads and hardware topologies [21]. Here we investigate whether multisocket topology matters when choosing deployment configuration in the cluster environment. In distributed deployments, distributed transactions are a major source of overheads. The faster communication, however, has a potential to significantly reduce the messaging overhead. Hence, we quantify the impact of fast communication on different workloads and deployment configurations. Finally, we evaluate how suitable the state-of-the-art scale-up OLTP designs are for the distributed deployments.

This paper presents an analysis of distributed OLTP deployments using established workloads. We compare scale-up and scale-out deployments of traditional and main-memory-optimized designs on different hardware platforms. We vary the communication channel to isolate the impact of network on different deployments depending on the workload properties. Overall, we quantify the challenges and opportunities for OLTP designs on clusters with fast interconnects.

Our experiments show the following:

1. Cluster deployments still benefit from choosing the optimal configuration within a node as scale-up deploy-

DaMoN'16, June 26-July 01 2016, San Francisco, CA, USA

ISBN 978-1-4503-4319-0/16/06…$15.00

DOI: http://dx.doi.org/10.1145/2933349.2933355

ments improve throughput by 2 to 4 times compared to the fine-grained scale-out ones. Furthermore, careful placement of threads to cores improves throughput by 10% to 60%.
2. Using a fast network increases throughput by a factor of 1.2 to 2 times for read-only workloads. When running update-heavy workloads, however, other overheads (i.e., logging for the traditional systems and increased contention for the in-memory optimized systems) dominate execution and overshadow any performance gains from fast communication.
3. Main-memory optimized designs exploit locality within an instance and achieve 1.2 to 1.8 times higher throughput in scale-up deployments when compared to the fine-grained scale-out ones. However, their concurrency control protocols are sensitive to delays in commit processing, causing reduction in throughput by a factor of 3.5, even if only 1% of the transactions are distributed.

The aforementioned experimental findings lead to the following insights. In order to use the full potential of clusters with fast interconnects, transaction processing systems need to scale up and out at the same time. Scale-up techniques are necessary but insufficient for scale-out deployments. To scale up without compromising the scale out dimension, concurrency control protocols need to become robust to the inter-node communication delays. At the same time, coordination protocols between the nodes cannot afford to add any overheads in the critical path of transaction execution.

2. RELATED WORK

Non-uniformity in multisockets. A recent study analyzes the different transaction processing systems on modern multisocket hardware [21]. They conclude that there is no single optimal configuration for all combinations of workload properties and hardware topology. Multimed [25] treats multisocket multicore as a distributed system and uses middleware software and replication to improve scalability of shared-everything systems. A lot of recent work has focused on data analytics and analyzing the impact of memory bandwidth bottlenecks on the performance of join and sorting algorithms [1, 4, 15, 19]. This analysis is complementary as we take the network into account in addition to inter-socket communication for OLTP workloads.

Impact of network. As today's applications store more and more data, distributed data processing architectures are proliferating. They are typically deployed on commodity machines in datacenters and are using commodity ethernet networks. Even though conventional wisdom is that the network is a bottleneck in distributed data analytics, a recent study demonstrates that CPU is the bottleneck and that optimizing network performance can only improve median job completion time by 2% [18]. One recent proposal has demonstrated significant improvements in performance by focusing on improving CPU efficiency instead of network and disk I/O [6]. Specific data processing operations, such as joins have been a popular target with multiple studies analyzing the impact of network and proposing holistic solutions that balance network delays with CPU overheads to improve performance [5, 20, 23]. We also believe that transaction processing systems need to take the holistic view when optimizing for clusters with fast networks.

Datacenter OLTP. Finally, with the rise of web-scale applications, distributed transaction protocols have attracted a lot of attention both in the industry and academia, especially around weaker models such as eventual consistency [29]. A recent study characterizes the consistency trade-offs from the application perspective and analyzes when weaker consistency models are appropriate [3]. When they are not applicable, and strong consistency guarantees are required, deterministic execution is an appealing choice. A recent study explores different classes of workloads to quantify when are deterministic distributed transaction processing systems preferable to the traditional ones [22]. It would be interesting to extend these studies to clusters with significantly faster network compared to the datacenter deployments.

3. SETUP AND METHODOLOGY

Distributed OLTP deployments. We use two state-of-the-art open-source OLTP systems. We choose Shore-MT as the representative traditional storage manager[1] and Silo as the main-memory optimized one[2]. Both of these systems use scale-up designs that we extend with a thin distributed transaction layer. We implement different communication mechanisms to execute distributed transactions using the standard two phase commit (2PC) protocol. Distributed transactions in our deployment fit into predefined transaction classes and are one shot [26] with local and remote transaction parts known apriori, which removes the need for more than one message in the first phase of 2PC. Local transaction site acts as a coordinator in the 2PC protocol. Unless noted otherwise, we bind threads to cores and allocate memory in the local memory node when possible to improve locality.

Hardware Platforms. We use two different hardware platforms: a cluster and a large multisocket server. Our cluster consists of 8 machines with 2 Intel Xeon X5660 processors each, connected using 10Gbps Ethernet network. Each machine has 48GB of RAM that we use for both data and log files through memory mapped disks. All experiments are run using Ubuntu 12.04.4 LTS (kernel version 3.2.0-34) and the software is compiled using GCC 4.6.3 with maximum optimizations. The large multisocket server has 8 Intel Xeon E7-L8867 processors connected in a twisted cube topology ensuring that each pair of processors is at most 2 hops away. The server has 192GB of main memory. We run all experiments using Red Hat Enterprise version 6.7 (kernel version 2.6.32) and compile the system using GCC 5.1.0 with maximum optimizations.

Workloads. We use a microbenchmark and TPC-C [27]. In the microbenchmark, the data is logically partitioned into a number of sites where each site holds a range of rows. It comes in two flavors: read-only and update. Each transaction in the microbenchmark reads or updates N rows, and belongs to one of the two types:

- **Local transactions** pick randomly N rows located in the local site
- **Multisite transactions** pick randomly one row located in the local site and the remaining $N-1$ rows uniformly from the whole dataset. Multisite transactions are distributed if some of the input rows happen to be located in the remote instance.

To characterize the impact of more complex transactions, we use Payment and NewOrder transactions from the TPC-C benchmark that comprise 88% of the benchmark mix. Both

[1]https://sites.google.com/site/shoremt/
[2]http://github.com/stephentu/silo

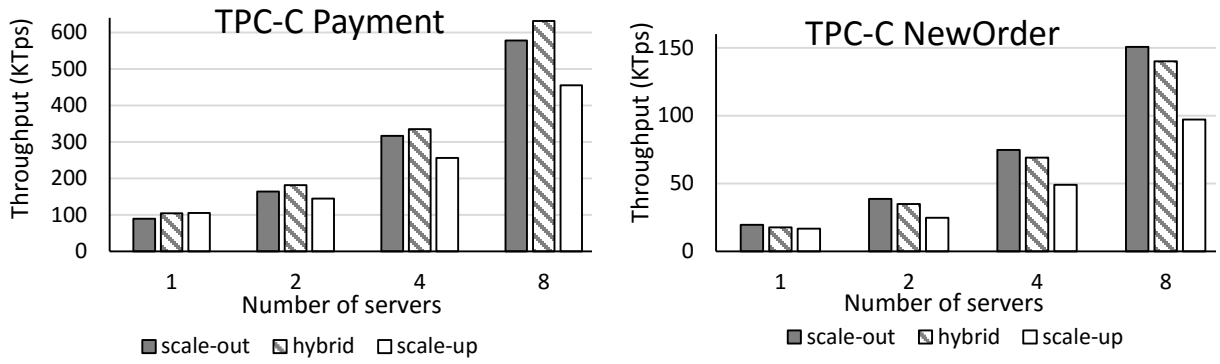

Figure 1: The impact of granularity on different deployments while running TPC-C.

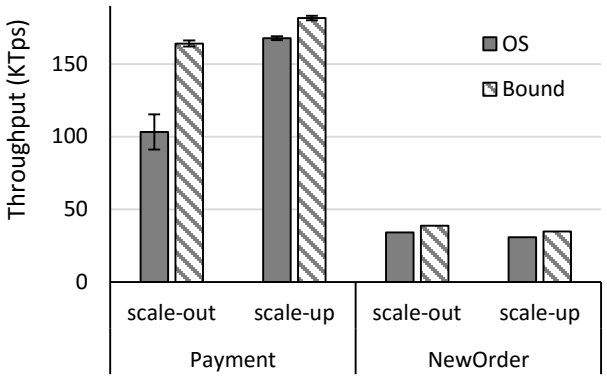

Figure 2: The impact of thread binding on different deployments while running TPC-C.

of them are read-write transactions that access data either from the local or the remote warehouse. The benchmark specifies that 15% of the Payment and 10% of the NewOrder transactions access remote warehouses. We partition the data using the well known scheme, where the data associated with a particular warehouse is placed in the same instance and the `Item` table is replicated in every instance.

4. HARDWARE ISLANDS IN A CLUSTER

The network poses much higher communication overheads across the cluster of machines compared to a single multisocket which potentially overshadows the impact of multisocket topology in cluster deployments. In this section, we use a cluster of machines and different workloads to quantify the impact of multisocket topology on the performance of different deployments. As we require TCP/IP communication channel for cluster deployment, we use the traditional OLTP system.

4.1 TPC-C

In all experiments, we choose a configuration for a machine and deploy it across all machines in the cluster. We use *scale-out* (one per core), *scale-up* (one per machine), and *hybrid* (one per socket) deployment configurations with 12, 1, and 2 instances per machine, respectively. We scale dataset sizes to 1 warehouse per core for TPC-C and 10 000 rows per core for the microbenchmark.

The impact of instance granularity. We start by analyzing scalability of TPC-C as we increase the number of machines from 1 to 8. We plot the throughput for both Payment Figure 1 (left) and NewOrder Figure 1 (right) transactions. On a single server, the larger instances perform better for Payment transactions, while the smaller ones perform better for NewOrder transactions. The difference stems from the type of write operations done by a transaction. For Payment transactions, larger instances profit from constructive sharing of a single log whereas each scale-out instance needs to write its own log and issue expensive system calls. On the other hand, the NewOrder transactions perform many insertions to the `OrderLine`, `Order`, and `NewOrder` tables, which require a lot of synchronization among threads in the same instance. Also, the scale-up deployment greatly benefits from the fact that it does not need to execute any distributed transactions.

When we increase the number of servers, smaller instances scale better than the scale-up deployment which requires executing distributed transactions when deployed over multiple servers. Scale-out deployments scale better than the hybrid ones for NewOrder, while the situation is reversed for Payment. All configurations scale linearly for NewOrder, while scalability is poorer for Payment. The difference in scalability comes from the type of updates performed by each transaction. Namely, Payment transactions update one row from the `Warehouse` table, which limits the number of concurrent transactions in the system to the number of warehouses. In contrast, NewOrder updates a row from the `District` table, that has 10 rows for each warehouse, thus permitting 10 times more concurrent transactions. Distributed transactions holding locks on the updated rows until the end of the second phase of the 2PC protocol lead to lower concurrency in Payment, which severely limits scalability.

Impact of thread binding. Careful thread binding is an important prerequisite for achieving predictable high performance on multisockets as it maximizes locality and avoids thread migrations [21]. In this experiment we investigate whether thread binding has any impact on performance of cluster deployments. We use 2 servers and either bind the instances to specific cores or sockets, or leave the placement to the operating system. We repeat the experiments with both Payment and NewOrder transactions.

The left hand side of Figure 2 shows throughput for the Payment transaction with solid bars representing threads placed by the operating system and striped bars representing manual binding with each core-sized scale-out instance on a separate core and each scale-up socket-sized instance on a separate socket. We run the experiment three times and show standard deviation on the bars. Binding instances to sockets improves performance of the scale-up deployment by 8%. This effect is more pronounced for the scale-out

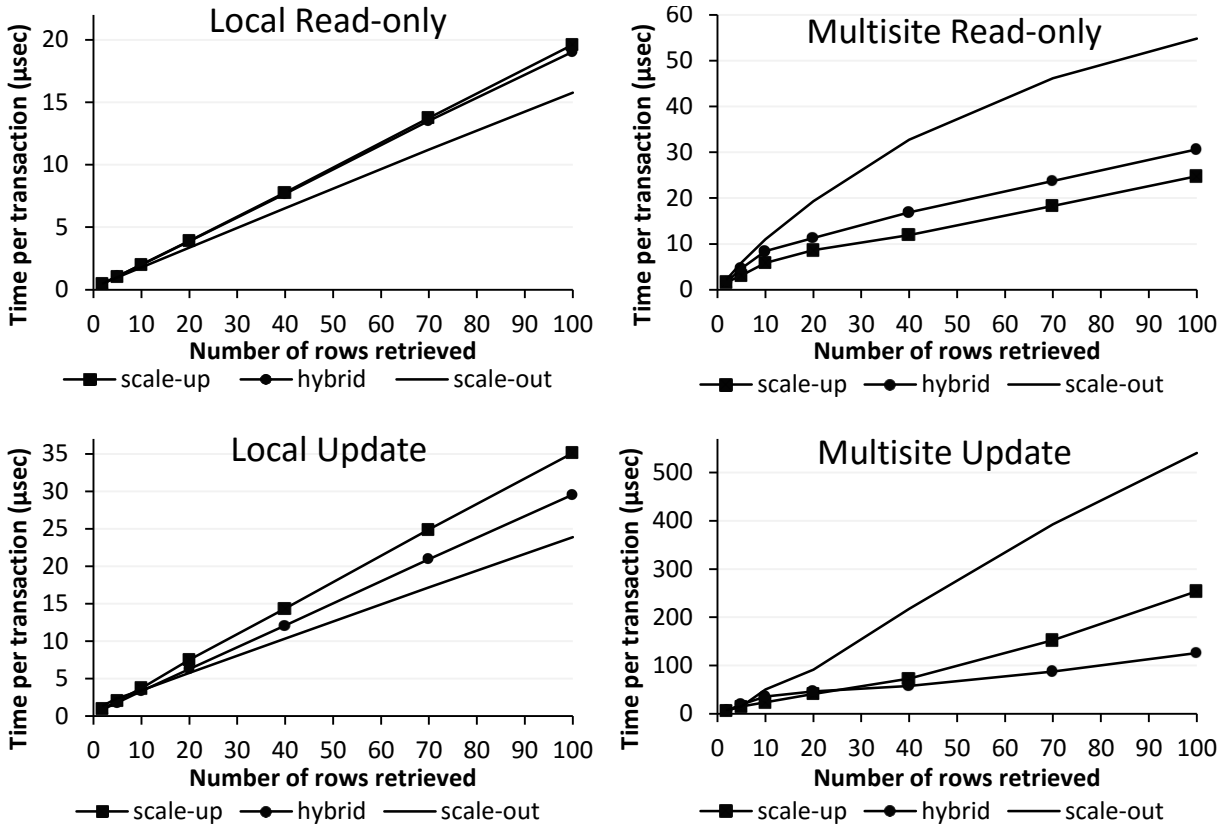

Figure 3: The cost of local and multisite transactions for different deployment configuration as the number of rows updated increases.

deployment, where binding instances to cores improves performance by 60% and reduces variability from 11.7% to 1.2%. The NewOrder transaction is much more predictable with standard deviations of less than 1% in all cases. Still, binding the instances improves performance by 13.5% and 12.9% respectively for scale-out and scale-up deployments.

4.2 Microbenchmarks

In easily partitionable TPC-C benchmark, each distributed transaction involves at most two instances, all transactions involve updates, and the percentage of distributed transactions is fixed. To better quantify the costs of arbitrary distributed transactions we perform a sensitivity analysis using microbenchmarks. The cost of a transaction is expressed as the time it takes to execute a single transaction. We use a 4 machine cluster and measure the cost of local and multisite transactions, in read-only and update versions, as we increase the number of rows accessed from 2 to 100. With the higher number of rows per transactions, multisite transactions require data from multiple instances and thus have to exchange more messages to complete a single transaction. We plot results in Figure 3.

Read-only case. The scale-out deployment has the lowest cost since each instance runs single-threaded and, hence, pays no thread synchronization overhead. Larger instances have higher costs due to these overheads. The cost trend is reversed for the multisite case where scale-out instances have significantly higher costs compared to the larger ones. The increase in cost is primarily due to the number of messages needed for a multisite transaction. Since these transactions are read-only, we use the optimized version of the 2PC protocol that requires only one roundtrip per participating site. For every deployment configuration, after the number of rows surpasses the number of instances in the system, every multisite transaction involves all instances in the system. This results in the flattening lines as the distributed transaction overheads become constant.

Update case. For the local transactions, the increase in cost is linear with the number of rows per transaction with larger instances having higher cost. The differences between configurations are more pronounced due to the higher synchronization overhead involved in the operations that modify data. In the multisite case, while the number of instances involved in a transaction increases at the same rate as in the read-only case, the costs increase faster. This effect is due to the higher communication costs (as update transactions require both rountrips in the 2PC protocol) and increased contention since locks are held until the end of a transaction. Even though scale-up and hybrid deployments, with multiple threads per instance, use optimized logging, their cost trends do not flatten out for higher number of rows due to increased contention. The increase is higher for the scale-up deployment because of more threads in the instance.

Summary. For the workloads that access many rows, overheads such as communication, logging and additional contention, make distributed transactions 2-4x more expensive for the fine-grained scale-out deployments compared to the coarser ones. When the number of instances required in a distributed transactions is small, as it is the case for the TPC-C, the optimal configuration depends on the trade-off

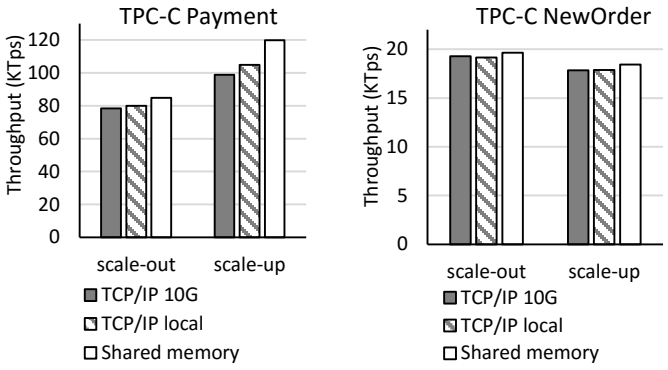

Figure 4: The impact of communication channel on TPC-C.

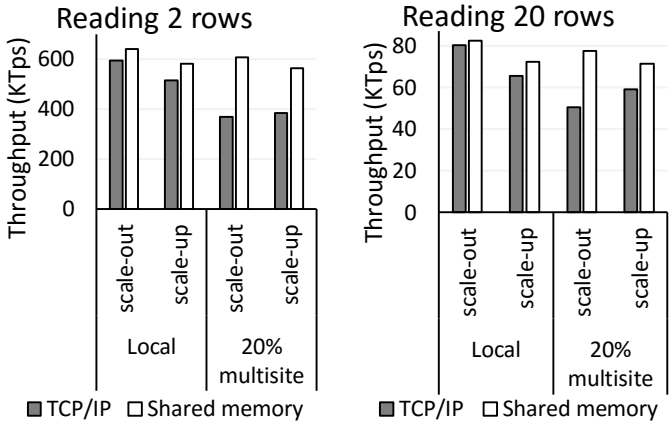

Figure 5: The impact of communication channel on read-only microbenchmarks.

between the overheads of thread synchronization and the opportunities for constructive sharing within an instance. Finally, adjusting the placement of the individual instances within a machine can significantly improve performance, especially for scale-out deployments, by improving locality.

5. THE IMPACT OF NETWORK

Network communication is a significant component of the cost of distributed transactions. Its performance depends on two factors: the hardware channel and the software stack. With high-speed low latency interconnects that enable RDMA-based messaging that bypasses the operating system, communication overheads significantly diminish. This potentially makes distributed transactions much cheaper and particular system designs more appealing. In this section, we quantify the impact of network performance on the throughput of different distributed deployments across various workloads and communication mechanisms.

5.1 TPC-C

We first use TPC-C to isolate the impact of different components of the communication channel on the distributed transactions. We compare three communication mechanisms: 1) TCP/IP over Ethernet network, 2) TCP/IP in a single machine, and 3) shared memory communication in a single machine. The first case represents today's mainstream option. The second case is the scenario with the fastest possible way of communication that still employs unmodified TCP/IP

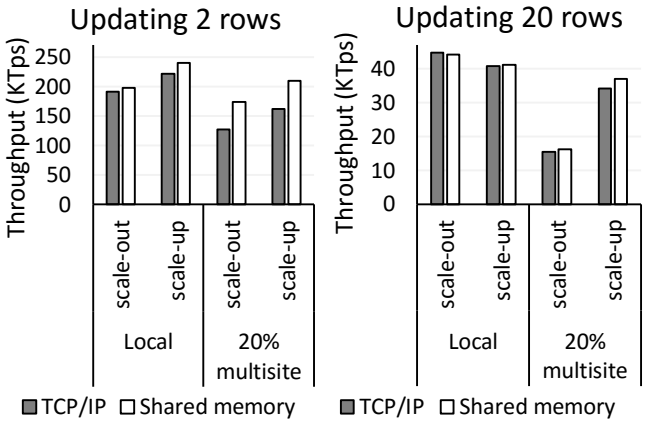

Figure 6: The impact of communication channel on update microbenchmarks.

software stack. We use shared memory communication to emulate the best RDMA scenario where accessing remote machine's memory has the same latency as accessing local memory. We use a dataset with 12 warehouses (1.8GB) and compare *scale-out* and *scale-up* deployments. For the first setting, we use two servers and deploy half of the instances (6 scale-out and 1 scale-up) on each server, while for the other two settings we deploy all instances on the same server.

The left hand side of Figure 4 shows throughput for the Payment transaction. Faster communication increases the performance of Payment transactions for both configurations. The magnitude of the increase depends on the size of the instance and the workload type. However, faster communication does not change the relative performance: the scale-up configuration has higher throughput than the scale-out one. The right hand side of Figure 4 shows experiment for the NewOrder transactions. In this case, communication speed has a negligible impact on the performance since NewOrder does many more operations per transaction than Payment and the cost of messaging is amortized.

5.2 Microbenchmarks

To better understand the impact of communication latency on the cost of distributed transactions depending on the type of operations, we run a series of experiments with microbenchmarks using TCP/IP and shared memory communication mechanisms. In all graphs, the dark bars show the case when we use TCP/IP for communication and the white bars represent shared memory. We use a single server and a dataset with 12 sites (120 000 rows) and compare *scale-out* and *scale-up* deployments over 12 cores. We study read-only and update cases separately and repeat the experiments with only local transactions and with a mix containing 20% multisite transactions. Also, we repeat microbenchmarks for 2 and 20 rows to assess the impact of 1) the different percentage of multisite transactions that are executed as distributed transactions and 2) the different number of instances involved in the execution of a single distributed transaction.

Read-only case. Figure 5 plots the results of the experiment for the read-only transactions with the 2 rows case (left) and 20 rows (right). In all cases, the deployments that use shared memory communication have higher throughput than the ones using TCP/IP. Since the read-only transactions are short, higher static communication overheads in the TCP/IP

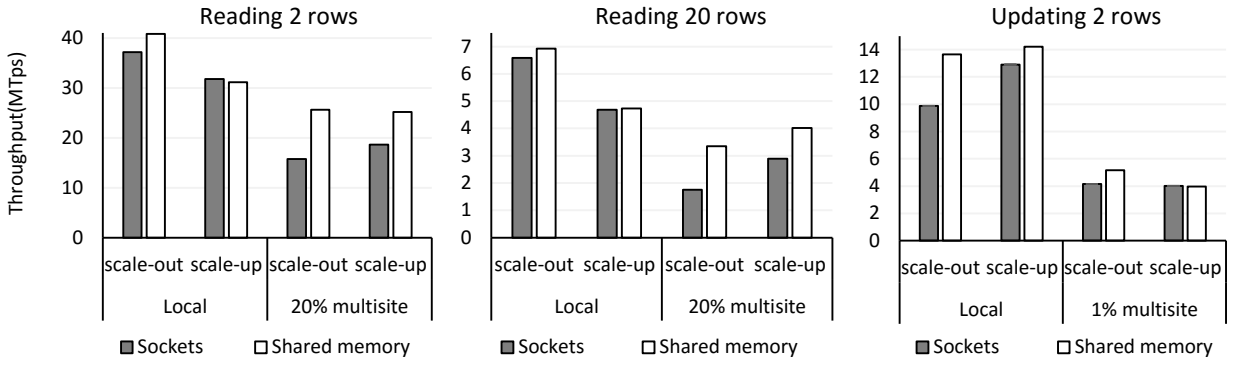

Figure 7: The impact of communication channel on distributed deployments of the main memory system.

case lead to noticeable difference in throughput for local only transactions. In order to fully exploit fast network, we need to avoid expensive system calls required for TCP/IP communication. For both types of transactions, we observe that the communication channel has a significant impact on relative performance of two deployments. Namely, in the presence of multisite transactions, scale-out deployment has higher performance than the scale-up one for shared memory communication, while the situation is reversed for TCP/IP. The impact of communication is higher for heavier transactions since every distributed transaction involves all instances which means scale-out configuration needs to exchange messages with 11 instances, compared to only one instance in the scale-up case. However, even in that case, fast network communication makes scale-out configurations faster than the scale-up one for all scenarios.

Update case. We plot the result of the update experiment in Figure 6. The impact of communication is less pronounced compared to the read-only case as distributed update transactions are significantly more expensive than their read-only counterparts. The difference comes from the ability to overlap logging and communication overheads. For example, transactions that update 2 rows generate less log, hence, they cannot overlap static communication overheads as effectively as the larger transactions. This effect is particularly evident in the presence of distributed transactions where the choice of communication mechanism has almost no effect on the throughput for the 20 row case.

Summary. The impact of network communication depends on the type of operations performed by distributed transactions. For the read-only transactions, communication has direct impact proportional to the number of instances involved in a transaction. On the other hand, the impact of network on the update workloads is much smaller due to other factors that dominate the cost of distributed update transactions. In the traditional system, the communication can be overlapped with logging or other processing. Furthermore, constructive sharing among threads makes scale-up instances preferable for many update workloads.

6. MAIN MEMORY SYSTEM

Each node in a cluster is a multisocket multicore with a large main memory and is connected to other nodes using low latency network. This section investigates how distributed deployments of the scale-up main-memory-optimized design compare to the scale-out deployments of the same system. We use Silo main memory OLTP system and approximate the cluster using the multisocket multicore server with each socket representing a node.

Distributed main memory system. Silo uses optimistic concurrency control (OCC) protocol that scales well on multicores as it avoids any centralized synchronization points. In order to adapt Silo for distributed deployment, we split its commit processing into two phases: 1) the validation phase that we perform at the end of the first phase of 2PC and 2) the actual commit that we perform in the second phase. Between these two phases, the updated rows are locked and any transactions attempting to read them is aborted.

We deploy a distributed version of Silo using a shared memory communication channel as well as UNIX domain sockets. The dataset has 8 million rows and is partitioned equally among all instances in the deployment. We compare *scale-out* deployments with one instance per processor core (80 instances) and *scale-up* deployment with one instance per processor socket (8 instances). We distinguish the behavior of the distributed deployment for the read-only and update workloads and identify the factors that cause differences between the two.

Read-only case. We start with the read-only microbenchmark for 2 and 20 rows for local only transactions and a mix with 20% multisite transactions and plot the results in Figure 7 (left and middle). In both cases, the fine-grained scale-out deployment has higher throughput for local transactions due to better locality of data accesses and absence of thread synchronization. For the experiment with the lightweight transactions, the choice of communication channel can reverse the relative performance of different deployments. Namely, scale-out deployment has higher throughout because of the locality of data accesses for the local transactions in both cases. However, with higher overhead sockets, especially for the case of heavier transactions in scale-out deployment, the relative performance reverses and scale-up deployment performs better. For the heavier transactions, scale-up deployment has higher throughput for multisite transactions in both cases due to fewer instances that participate in a single transaction causing lower communication overheads.

Update case. To quantify the impact of updates, we run a microbenchmark that updates 2 rows and compare two settings: 1) only local transactions and 2) 1% multisite transactions. We plot the results in Figure 7 (right). We use significantly smaller percentage of multisite transactions compared to the previous experiment since distributed update transactions have much higher cost. In contrast to the

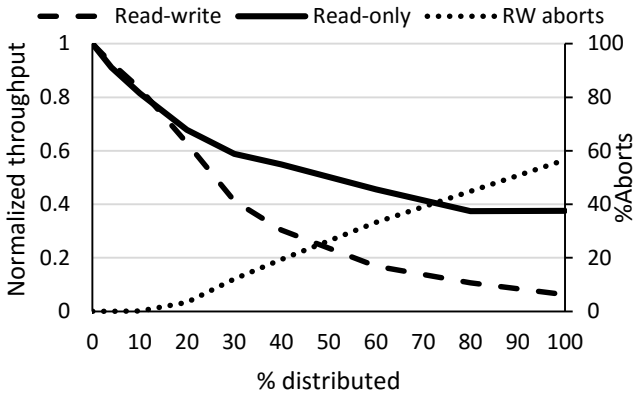

Figure 8: Increasing the duration of commit processing in distributed transactions significantly increases abort rates.

read-only distributed transactions, the update ones increase contention due to prolonged commit phase that leads to abort rates of 11.5% for 1% of multisite transactions. While the pessimistic choice to abort transactions attempting to update locked row has a negligible effect when running only local transactions as the commit phase is very short, its impact is much bigger in the presence of distributed transactions. Communication channel has much lower impact on the update transactions compared to the read-only ones. For all combinations of deployment configuration and communication mechanisms, even small percentage of distributed transactions significantly affects contention. The increased contention leads to abort rates of 8% to 11.5% with slightly lower abort rates when using UNIX domain sockets.

Sensitivity analysis. In order to characterize the different effect of the read-only and update distributed transactions on the throughput, we distinguish the impact of communication overheads and increased contention. We use a modified microbenchmark on a deployment that has only 2 instances, each using a single core. We vary the percentage of distributed transactions from 0 to 100%. The dataset contains 200 000 rows evenly split between the instances. We use two microbenchmarks: 1) the one that reads 2 rows and 2) the one that reads 1 row and updates 1 row. For the local transactions, both rows are chosen from the local instance. For the distributed transactions, one row is chosen from the local and the other is chosen from the remote instance. In the read-only case, distributed transactions will incur only the communication overheads. In the read-write case, we choose the row that is updated from the local instance and the row that is read from the remote instance. In both cases, distributed transaction have the same communication overheads since the remote fragment is read-only and does not require any processing in the second phase. However, for the read-write case, the update row is locked until the remote fragment is processed causing any concurrent requests for that row to conflict.

We plot the normalized throughput of both microbenchmarks as well as the abort rates for the read-write one in Figure 8. For small percentages of distributed transactions, the relative throughputs of the read-only and the read-write microbenchmarks follow the same trend as long as abort rates are negligible. However, with 10% or more of distributed transactions in the workload, the throughput of read-write microbenchmark starts dropping faster. At the same time, abort rates steadily increase reaching 55% when all transactions are distributed while the throughput plummets to 6% of the peak.

Summary. For the read-only distributed transactions, communication is the main overhead. Hence, distributed transactions affect the coarser grained configurations less since they potentially involve fewer instances in the execution of a transaction. The impact of distributed transactions is much higher for the transactions that contain updates. Main-memory optimized systems achieve high performance by accessing only a small number of short critical sections in the critical path of transaction execution. Adding communication step in the middle of the commit processing of the efficient OCC protocol increases abort rates significantly and has detrimental effect on performance.

7. CONCLUSIONS

The current state-of-the-art transaction processing system designs either focus on scaling up or scaling out. In this paper, we analyze different deployments of the traditional and main-memory-optimized designs and characterize the impact of communication latency, multisocket topology, and workload properties on throughput. We show that the fast communication improves throughput for the read-only workloads while having negligible impact for the update ones where other overheads, i.e., logging for the traditional system and increased contention for the main memory one, dominate. Regardless of the network performance, optimal configuration for the cluster deployment of the OLTP designs requires taking into account the topology of a multisocket node as well as the workload characteristics.

Challenges. Inadequacies of the state-of-the-art concurrency control and coordination protocols stem from scale-up and scale-out design requirements respectively. On the one hand, concurrency control protocols for main memory optimized scale up designs need to minimize the duration of any critical sections so as not to introduce any scalability bottlenecks. This makes them sensitive to delays introduced in the critical path of transaction execution. On the other hand, coordination protocols aim to minimize the number of messages between nodes in the distributed system as communication latencies dominate all other delays in the system. However, this allows them to add significant local processing overhead that is prohibitive for lean main memory optimized systems.

Opportunities. In order to be applicable to clusters with fast networks, concurrency control protocols need to become resilient against communication delays and the coordination protocols need to become more lightweight to capture the best of both worlds. One approach for making concurrency control protocols more amenable to distributed execution is using techniques such as controlled lock violation to shorten commit processing by tracking dependencies [13]. This optimistic approach may lead to a chain of aborts. However, such behavior is restricted to the situations where there are many read/write conflicts on the hot data. A complementary set of techniques rely on application semantics to enable phase reconciliation and knowing transaction write-set apriori to increase concurrency [12, 16]. Similar ideas that rely on application semantics to relax coordination requirements in distributed deployments have shown good results in datacenter deployments [2, 24]. We believe that the judicious use

of semantic information from the application enables design of resilient concurrency control and lightweight coordination protocols required for efficient rack-scale OLTP designs. Two recent proposals leverage RDMA and modern hardware, namely non-volatile RAM and hardware transactional memory, to achieve good scalability for easily partitionable workloads, such as TPC-C, on clusters with fast networks [11, 30]. They present a good initial step toward designing efficient systems for arbitrary transaction processing workloads that scale up and out.

Acknowledgments

We would like to thank Eric Sedlar, Ippokratis Pandis, and the members of the DIAS laboratory for their insightful feedback throughout this work. This work is partially funded by the Swiss National Science Foundation (Grant No. 200021-146407/1).

8. REFERENCES

[1] M.-C. Albutiu, A. Kemper, and T. Neumann. Massively parallel sort-merge joins in main memory multi-core database systems. *PVLDB*, 5(10):1064–1075, 2012.

[2] P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Coordination avoidance in database systems. *PVLDB*, 8(3):185 – 196, 2015.

[3] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica. Quantifying eventual consistency with PBS. *The VLDB Journal*, 23(2):279–302, 2014.

[4] C. Balkesen, G. Alonso, J. Teubner, and M. T. Ozsu. Multi-Core, Main-Memory Joins: Sort vs. Hash Revisited. *PVLDB*, 7(1), 2014.

[5] C. Barthels, S. Loesing, G. Alonso, and D. Kossmann. Rack-Scale In-Memory Join Processing using RDMA. In *SIGMOD*, pages 1463–1475, 2015.

[6] A. Crotty, A. Galakatos, K. Dursun, T. Kraska, U. Cetintemel, and S. Zdoni. Tupleware:"Big" Data, Big Analytics, Small Clusters. In *CIDR*, 2015.

[7] T. David, R. Guerraoui, and V. Trigonakis. Everything You Always Wanted to Know About Synchronization but Were Afraid to Ask. SOSP, pages 33–48, 2013.

[8] A. Dhodapkar, G. Lauterbach, S. Li, D. Mallick, J. Bauman, S. Kanthadai, T. Kuzuhara, G. Xu, and C. Zhang. SeaMicro SM10000-64 Server: Building Datacenter Servers Using Cell Phone Chips. In *HotChips*, 2011.

[9] C. Diaconu, C. Freedman, E. Ismert, P.-A. Larson, P. Mittal, R. Stonecipher, N. Verma, and M. Zwilling. Hekaton: SQL Server's Memory-Optimized OLTP Engine. In *SIGMOD*, pages 1243–1254, 2013.

[10] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro. FaRM: Fast Remote Memory. In *NSDI*, pages 401–414, 2014.

[11] A. Dragojevic, D. Narayanan, E. B. Nightingale, M. Renzelmann, A. Shamis, A. Badam, and M. Castro. No compromises: distributed transactions with consistency, availability, and performance. In *SOSP*, 2015.

[12] J. M. Faleiro and D. J. Abadi. Rethinking serializable multiversion concurrency control. *PVLDB*, 8(11):1190–1201, 2015.

[13] G. Graefe, M. Lillibridge, H. Kuno, J. Tucek, and A. Veitch. Controlled lock violation. In *SIGMOD*, pages 85–96, 2013.

[14] H. Kimura. FOEDUS: OLTP Engine for a Thousand Cores and NVRAM. In *SIGMOD*, pages 691–706, 2015.

[15] Y. Li, G. Lohman, I. Pandis, R. Mueller, and V. Raman. Numa-aware algorithms: the case of data shuffling. In *CIDR*, 2013.

[16] N. Narula, C. Cutler, E. Kohler, and R. Morris. Phase reconciliation for contended in-memory transactions. *OSDI*, pages 511–524, 2014.

[17] S. Novakovic, A. Daglis, E. Bugnion, B. Falsafi, and B. Grot. Scale-Out NUMA. In *ASPLOS*, 2014.

[18] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, B.-G. Chun, and V. ICSI. Making sense of performance in data analytics frameworks. In *NSDI*, pages 293–307, 2015.

[19] O. Polychroniou and K. A. Ross. A comprehensive study of main-memory partitioning and its application to large-scale comparison- and radix-sort. In *SIGMOD*, pages 755–766, 2014.

[20] O. Polychroniou, R. Sen, and K. A. Ross. Track join: distributed joins with minimal network traffic. In *SIGMOD*, pages 1483–1494, 2014.

[21] D. Porobic, I. Pandis, M. Branco, P. Tözün, and A. Ailamaki. OLTP on hardware islands. *PVLDB*, 5(11):1447–1458, 2012.

[22] K. Ren, A. Thomson, and D. J. Abadi. An evaluation of the advantages and disadvantages of deterministic database systems. *PVLDB*, pages 821–832, 2014.

[23] W. Rodiger, T. Muhlbauer, P. Unterbrunner, A. Reiser, A. Kemper, and T. Neumann. Locality-sensitive operators for parallel main-memory database clusters. In *ICDE*, pages 592–603, 2014.

[24] S. Roy, L. Kot, G. Bender, B. Ding, H. Hojjat, C. Koch, N. Foster, and J. Gehrke. The homeostasis protocol: Avoiding transaction coordination through program analysis. In *SIGMOD*, pages 1311–1326, 2015.

[25] T.-I. Salomie, I. E. Subasu, J. Giceva, and G. Alonso. Database engines on multicores, why parallelize when you can distribute? In *EuroSys*, pages 17–30, 2011.

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

[27] TPC. TPC benchmark C (OLTP) standard specification, revision 5.9, 2007. Available at http://www.tpc.org/tpcc.

[28] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden. Speedy Transactions in Multicore In-memory Databases. In *SOSP*, pages 18–32, 2013.

[29] W. Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, 2009.

[30] X. Wei, J. Shi, Y. Chen, R. Chen, and H. Chen. Fast in-memory transaction processing using RDMA and HTM. In *SOSP*, 2015.