SPADE: Tuning scale-out OLTP on modern RDMA clusters

Georgios Chatzopoulos
EPFL
georgios.chatzopoulos@epfl.ch

Aleksandar Dragojević
Microsoft Research
alekd@microsoft.com

Rachid Guerraoui
EPFL
rachid.guerraoui@epfl.ch

ABSTRACT

Distributed transactions on modern RDMA clusters promise high throughput and low latency for scale-out workloads. As such, they can be particularly beneficial to large OLTP workloads, which require both. However, achieving good performance requires tuning the physical layout of the data store to the application and the characteristics of the underlying hardware. Manually tuning the physical design is error-prone, as well as time-consuming, and it needs to be repeated when the workload or the hardware change.

In this paper we present SPADE, a physical design tuner for OLTP workloads in FaRM, a main memory distributed computing platform that leverages modern networks with RDMA capabilities. SPADE automatically decides on the partitioning of data, tunes the index and storage parameters, and selects the right mix of direct remote data accesses and function shipping to maximize performance. To achieve this, SPADE combines information derived from the workload and the schema with low-level hardware and network performance characteristics gathered through micro-benchmarks.

Using SPADE, the tuned physical design achieves significant throughput and latency improvements over a manual design for two widely used OLTP benchmarks, TATP and TPC-C, sometimes using counter-intuitive tuning decisions.

CCS CONCEPTS

• **Information systems** → **Relational parallel and distributed DBMSs**; *Distributed storage*; Record and block layout; • **Networks** → *Network performance modeling*;

KEYWORDS

SPADE, OLTP, RDMA, Tuning

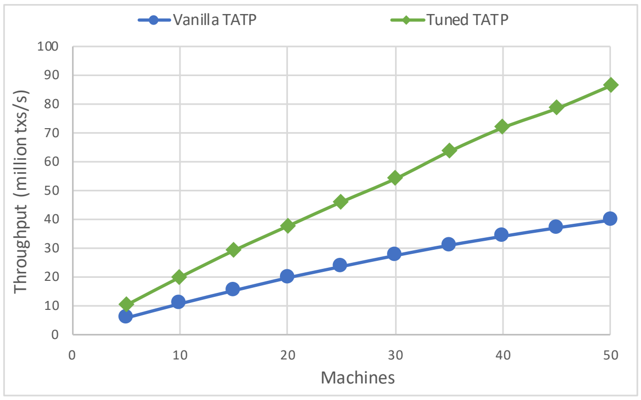


Figure 1: Effect of physical design tuning on TATP.

ACM Reference Format:
Georgios Chatzopoulos, Aleksandar Dragojević, and Rachid Guerraoui. 2018. SPADE: Tuning scale-out OLTP on modern RDMA clusters. In *19th International Middleware Conference (Middleware '18), December 10–14, 2018, Rennes, France.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3274808.3274815

1 INTRODUCTION

Modern hardware, with large amounts of main memory and high-speed networks with Remote Direct Memory Access (RDMA) capabilities, has already made its way into datacenters, enabling a new generation of distributed algorithms and systems. Several transactional systems have been devised to take advantage of this modern hardware [4, 8, 13, 14, 19, 26, 54]. These systems achieve high throughput, low latency and good scalability in On-Line Transaction Processing (OLTP) workloads. They can commit tens of millions of transactions per second, with latencies in the micro-second range.

However, achieving such performance is not straightforward. Implementing even a single *SELECT* statement of an OLTP workload involves physical design decisions that arise from the use of RDMA networks: (1) tuning the data indexes and storage, and (2) data partitioning and remote data access optimization. Previous work required manual tuning of such parameters to achieve performance. However, this process is both cumbersome and error-prone when performed for every parameter of a system. Additionally, even small changes in the workload, or the underlying hardware, require fully repeating the process. The impact of tuning can be significant. Figure 1 depicts 2.2x difference in performance between finely tuned and default physical designs of TATP [30].

In this paper, we present SPADE (Scale-out PhysicAl Design tunEr), a tuner for OLTP workloads in FaRM [13, 14], which automates these physical design configuration choices. First, SPADE tunes the index and storage parameters of a workload (1). Maximizing index performance on modern RDMA clusters requires configuration based on the size of rows and the operation mix. For example, inlining data in the primary index is beneficial for small row lookups, as it reduces the number of remote accesses. In contrast, for large rows, it is better to store the data outside of the index, as this reduces the amount of data transferred [13]. The exact threshold depends on the workload and the underlying hardware. Additional operations on tables (e.g., insertions or updates) and indexes (e.g. secondary) further complicate these decisions. The impact of an incorrect configuration on performance is significant: we noticed up to 15% impact on throughput and 22% on latency, when configuring indexes.

Additionally, SPADE optimizes partitioning and remote data accesses (2). While traditional systems rely heavily on partitioning, modern distributed transactional systems can take advantage of one-sided RDMA reads to avoid expensive network packet processing, improving throughput and lowering latency. Using RDMA reads is beneficial for a range of operations. However, message-passing (or Remote Procedure Calls – RPCs) can outperform direct accesses when data is collocated on the same machine, as well as for large write-intensive operations. SPADE identifies whether to use RDMA or message-passing based on the schema and workload characteristics. Correctly optimizing remote accesses not only improves the throughput, but also the median and tail latency of a workload. In our experiments, tail latency is reduced by up to 80% when correctly tuning remote accesses.

SPADE tunes OLTP workloads offline. Initially, SPADE takes query plans of the stored procedures of an OLTP workload as input. It examines these plans and combines them with information on the schema and workload characteristics. It identifies partitioning opportunities and creates a model for each procedure of the workload, consisting of low-level operations, targeting the FaRM API. Feeding these models with the performance characteristics of the cluster, gathered through microbenchmarks, enables SPADE to decide on a configuration for each table (i.e., data and index design), as well as for each stored procedure (i.e., using message-passing or one-sided remote memory accesses).

SPADE is designed and implemented as an additional level of tuning on top of FaRM [13, 14]. FaRM is a main memory distributed computing platform that uses one-sided RDMA memory accesses, battery-backed main memory and transaction, replication, and recovery protocols to achieve high availability, scalability and performance on modern datacenter hardware. We evaluate the performance of OLTP workloads tuned with SPADE on top of FaRM using two popular OLTP benchmarks: TATP and TPC-C. We study the extent to which tuning decisions affect performance, showing that index and storage tuning can improve throughput by up to 15%, partitioning by up to 50% and remote access tuning by up to 31%. Similarly, they reduce median latency by up to 22%, 25% and 23% respectively. Combined, these tuning decisions improve throughput by as much as 117%, while lowering latency by up to 75% compared to a workload with basic tuning. We highlight some counter-intuitive choices that SPADE makes, which would be difficult for a developer or administrator to identify. These choices improve throughput and reduce both median and 99th percentile latency.

In summary, the contributions of this paper are as follows:

- We identify RDMA-specific tuning opportunities that are crucial to the performance of OLTP workloads on modern RDMA clusters.
- We describe SPADE, a physical design tuner for OLTP workloads in FaRM. SPADE automates index tuning, data partitioning, and remote data access tuning, based on schema, workload and hardware characteristics.
- We evaluate the effect of physical design tuning on the throughput and latency of OLTP workloads.

It is important to note that SPADE is not meant to replace existing query optimizers. In this paper we identify and quantify the effect of RDMA-specific choices on the performance of OLTP workloads in FaRM, as an additional level of necessary optimizations. Similarly, SPADE might not be applicable to all distributed transaction systems, since not all of them offer the possible choices in SPADE. However, as such systems become popular, there are lessons to be learned for related future efforts.

The rest of the paper is organized as follows. In Section 2 we recall some background and motivate tuning on modern RDMA clusters. In Section 3 we present the architecture of SPADE and in Section 4 we describe how tuning is performed. We evaluate workloads tuned with SPADE in Section 5. We discuss related work in Section 6 and present some concluding remarks in Section 7.

2 BACKGROUND AND MOTIVATION

2.1 Modern hardware trends

Modern hardware clusters follow two main trends: large main memories and fast network interconnects. Indeed, commodity machines nowadays typically contain a few hundreds of gigabytes of main memory at very low prices. This enables handling tens of terabytes of data with only a small cluster of machines, benefiting from the low latency and high throughput of main memory and avoiding complex buffering mechanisms [41]. Ensuring data durability in main memory systems requires persistent logging. In order to avoid the overheads of logging to disks or SSDs, machines of a cluster can be equipped with a "distributed uninterruptible power supply" (distributed UPS) [14, 28] that writes the contents of the memory to an SSD when a power failure occurs, effectively making all DRAM non-volatile. New Non-Volatile RAM (NVRAM) technologies that have been proposed [24, 27, 38, 42] provide an alternative.

Datacenter networking has also evolved. Commodity Network Interface Controllers (NICs) nowadays support RDMA. Current implementations include InfiniBand, RoCE (RDMA over Converged Ethernet) and iWARP (internet Wide Area RDMA Protocol). Connection-based implementations support one-sided RDMA requests. Applications register memory regions with the NIC, which can then serve one-sided RDMA requests (e.g., remote read or write requests) without involving the remote CPU. Remote reads and writes are currently only supported in reliable connection-based implementations. RDMA has been used in a number of different systems and scenarios [6, 17, 18, 25, 29, 40, 43, 49].

2.2 Distributed transactions

Distributed database transactions are an active area of research [20, 45, 52]. Recently, various systems have utilized fast networks and modern hardware to achieve millions of OLTP transactions at sub-millisecond latencies [4, 8, 13, 19, 26, 54]. Most of these systems use (or rely on) message-passing abstractions (typically Remote Procedure Calls – RPCs), one-sided RDMA operation, or a combination of the two (e.g., one-sided RDMA reads and writes with RPCs).

There are two main execution models in distributed transaction systems. In the first [8, 13, 19], (referred to as *symmetric*), all the machines store data and execute transactions. Each server is responsible for storing parts of the data (and possibly replicas of the data stored by other machines of the cluster), as well as running distributed transactions. The second model [26, 54] distinguishes between computation and storage nodes. The former run transactions, utilizing RDMA to access the data stored in the memory of the latter.

In this paper, we use FaRM[13, 14] as our target platform. In FaRM, objects are stored in the memory of the cluster with a global address. FaRM uses one-sided RDMA accesses for lock-free reads, as well as message-passing for read-write transactions. It offers an API that allows applications to create and execute transactions. It also includes implementations of a distributed hashtable and a distributed B-Tree. FaRM implements a symmetric execution model.

2.3 The cost of performance on modern RDMA networks

Distributed transaction systems typically offer APIs for low-level transactions. Some of them also include implementations of data structures that can be used for OLTP workloads. More specifically, [4, 8, 13, 19, 26, 54] have a hashtable implementation, while [8, 13, 26, 54] have a B-Tree [9] implementation. Even with the available APIs and data structures, implementing OLTP workloads using such an API is not straightforward. For example, given only two of the queries of TATP benchmark [30], *GET_SUBSCRIBER_DATA* and *UPDATE_LOCATION* (shown in Listing 1 and Listing 2), there are a few crucial choices to get performance out of the underlying hardware. For the *Subscriber* table, these are:

(1) Table storage. This includes data partitioning and deciding on how the data is stored, for example inlining data in an index, or flat storage of data.
(2) Indexes. By examining the queries, in addition to a primary hash index on *s_id*, a secondary hash index on *sub_nbr* is necessary. Based on the previous step, each index needs additional tuning to get the best performance of the underlying network, based on characteristics such as the record size, the operation mix and the neighbourhood size of the hash index.
(3) Remote data access. Possible choices here are: a) directly accessing the data using RDMA, b) doing an RPC to the server holding the data, or c) a combination of the two. Choosing between directly accessing data and sending a message can be tuned based on the length of the operations and the size of transfers necessary.

```
SELECT s_id, sub_nbr,
    bit_1, bit_2, bit_3, bit_4, bit_5,
    bit_6, bit_7, bit_8, bit_9, bit_10,
    hex_1, hex_2, hex_3, hex_4, hex_5,
    hex_6, hex_7, hex_8, hex_9, hex_10,
    byte2_1, byte2_2, byte2_3, byte2_4,
    byte2_5, byte2_6, byte2_7, byte2_8,
    byte2_9, byte2_10,
    msc_location, vlr_location
FROM Subscriber
WHERE s_id = <rnd>;
```

Listing 1: TATP's *GET_SUBSCRIBER_DATA* query.

```
UPDATE Subscriber
SET vlr_location = <rnd>
WHERE sub_nbr = <rnd>;
```

Listing 2: TATP's *UPDATE_LOCATION* query.

Making the correct choices is not only cumbersome, but also error-prone. Workloads typically contain multiple operations, and queries touch data on multiple tables. Deciding on a set of parameters for each table quickly becomes overwhelming, and making the wrong choices can significantly impact performance, as we show in Figure 1. Changing the workload requires repeating the process, which is time-consuming. Previous work has implemented and studied the performance of OLTP transactions on modern RDMA clusters, but has not looked at the trade-offs of tuning, or quantified the effect that these choices have on the performance of an OLTP workload on modern RDMA networks.

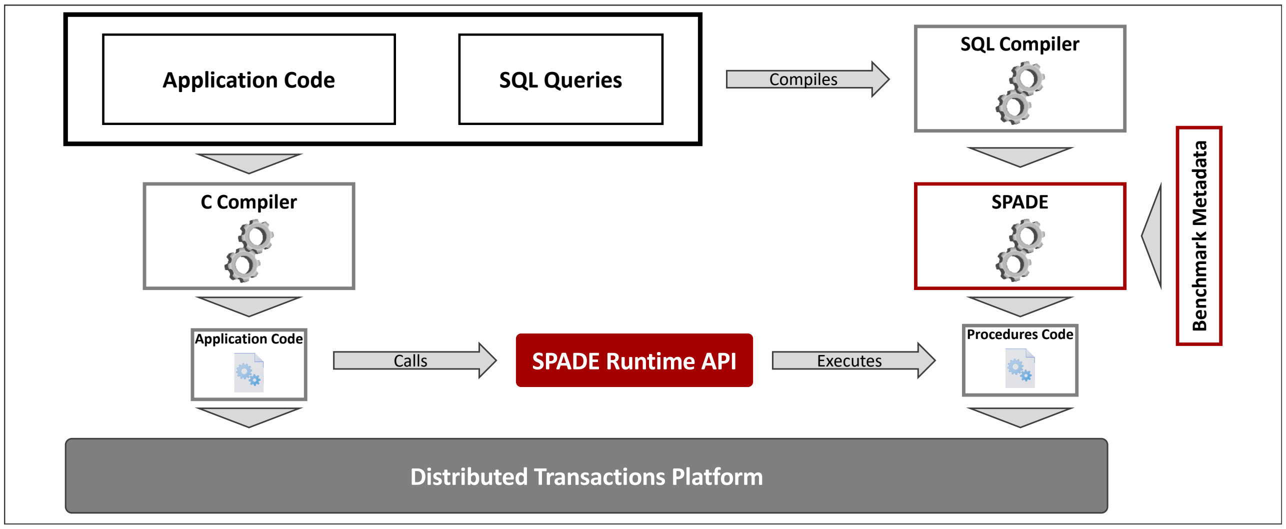


Figure 2: The architecture of SPADE.

3 SPADE

In this section, we first describe SPADE's architecture and how it fits in the execution of an OLTP workload in FaRM. We then present how we compile query plans into low-level transactions, how SPADE tunes the physical design, as well as the models it uses and how these are fed with data.

3.1 Architecture overview

Figure 2 presents SPADE's architecture. An application consists of application code and stored procedures. Stored procedures are written in a SQL dialect and compiled by a SQL compiler offline. In our implementation, we used the PostgreSQL compiler to produce physical execution plans of the workloads (see Section 5.1). SPADE then analyzes the physical plans, tunes the indexes used based on the workload queries, and makes per-procedure choices regarding remote accesses. We leave support for online tuning as future work. The stored procedures are then compiled into low-level code that utilizes the distributed transactional platform's API to access data and indexes, similarly to the in-memory component of SQL Server [11]. The stored procedures are uploaded to an RDMA cluster together with the application. During execution, the application uses the SPADE runtime API to bind arguments and execute the stored procedures of the workload, as well as to retrieve the results of the queries.

Queries are compiled in two steps. Physical query plans, which describe the operators that need to be executed, the type of joins to use, and similar, are produced first using a SQL compiler. We use existing techniques for query compilation and do not impose any specific requirements on the physical plan. SPADE takes as input the query execution plan produced by an optimizer. With these plans as input, SPADE produces low-level code that implements the stored procedures, tuning their physical design, using the API offered by the distributed transactional platform.

```
void get_subscriber_data(int s_id) {
  auto transaction_object =
  new SQLTransaction("GET_SUBSCRIBER_DATA");
  transaction_object->AddArgument(0, s_id);
  auto results = transaction_object->Execute();
  if (results->status == TxStatus::Committed)
  {
  // Application code to use the results
  }
}
```

Listing 3: TATP's *GET_SUBSCRIBER_DATA* invoked in an application.

To tune the physical design of queries, SPADE models each operation (e.g., lookups, updates, etc.) as a set of low-level distributed operations (e.g., remote reads and writes). To quantify the performance of these operations, SPADE executes a set of microbenchmarks on the target cluster. These include single operations (lookups, updates, insertions and deletions) executed both through a primary and a secondary index, for different cluster and row sizes. These microbenchmarks need to be run only once. SPADE executes three different scenarios for each operation: a) the operation accessing local data, b) the operation accessing remote data through RDMA, and c) the operation executing locally at a remote machine through an RPC. The results of these benchmarks are used by SPADE. We detail how SPADE uses this metadata for each tuning decision it automates in the following.

We present how the *GET_SUBSCRIBER_DATA* query (Listing 1) can be invoked in an application in Listing 3. The code creates an object based on the name of the stored procedure (given by the developer) and invokes the *Execute* method. The application can check whether the transaction committed, and use the results of the query (in this case the subscriber data). If the SQL transaction has aborted, the application can retry the same transaction, based on whether the abort was due to contention or lack of data (not shown here).

4 TUNING

In this section, we discuss the tuning decisions we consider in SPADE. For each one we first present the possible choices and some intuition on how these affect performance. We then show examples of how these different options impact performance based on our microbenchmarks, and how we use this information in SPADE to tune a workload. We distinguish two categories of design choices, which we describe in the following subsections. We assume a row-store format, which is the prevalent format for OLTP workloads.

4.1 Index tuning

Indexes are crucial to databases and can significantly speed up data accesses. SPADE currently supports hash indexes and B-Tree indexes. Modern distributed transaction systems provide at least one of the two. Tuning these indexes can optimize both the size and the number of network accesses. Typically a trade-off exists–increasing the size of accesses reduces their number, but also the request rate the hardware can support. In SPADE we identified two opportunities for tuning indexes. The first is choosing between storing records inside the data structures (e.g., as the value of a hash index) or storing only a pointer to the record in the index. We refer to the first as *inlined indexes* and to the second as *non-inlined* indexes. The second choice is on the neighbourhood sizes for hash indexes and node sizes for B-Trees.

We use a hash index as an example of performance impact, but similar observations hold for B-Trees. Hash indexes in FaRM are implemented using *chained associative*

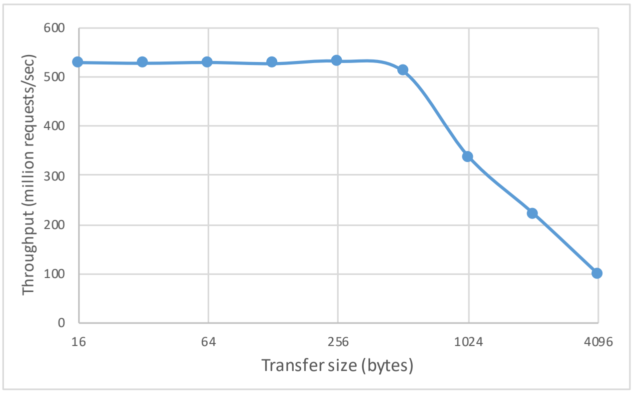
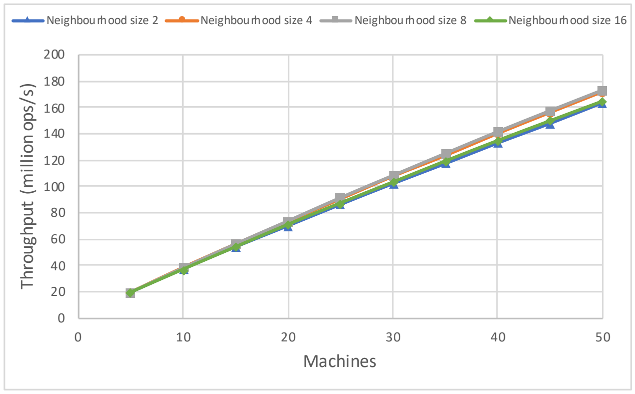
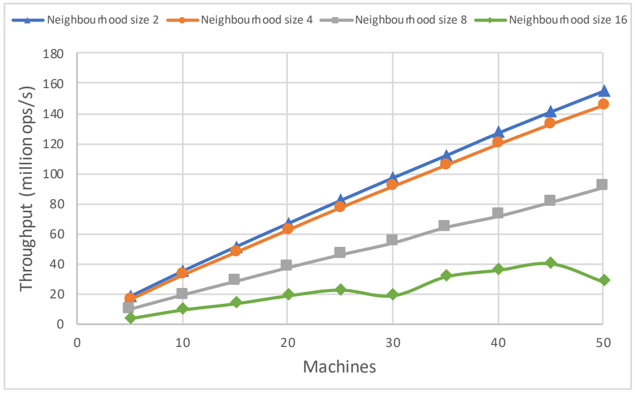


Figure 3: Performance of random RDMA reads.



(a) 16-byte rows.

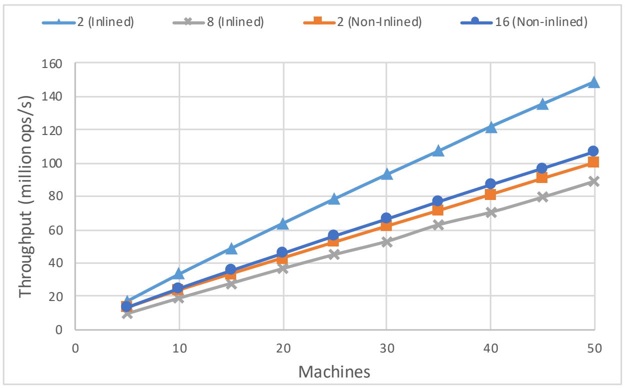


(b) 256-byte rows.

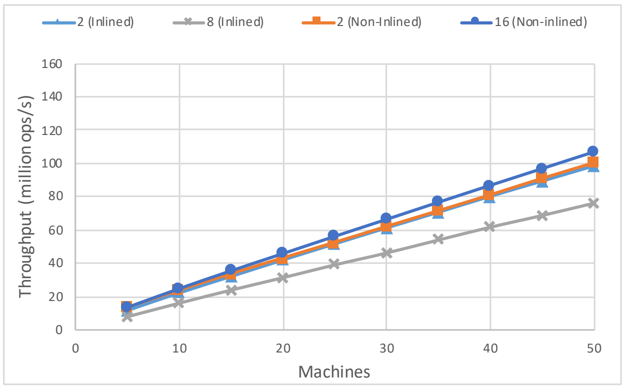
Figure 4: Performance of inlined lookups.

hopscotch hashing [13]. Each bucket in the hashtable has an overflow chain, which stores key-value pairs that do not fit into the bucket. Lookups use one-sided RDMA reads to access a bucket, as well as to fetch the overflow chain blocks. A small bucket will typically lead to more RDMA accesses for the overflow chain. In contrast, a large bucket will result in fewer RDMA accesses, but its performance might suffer because of the large RDMA transfers. In Figure 3 we present the throughput of RDMA requests for different transfer sizes on a cluster of 50 machines (our setup is described in Section 5.2). For transfers of up to 256 bytes, the request rate remains constant. However, for sizes above 512 bytes the request rate degrades significantly. The specific numbers change based on the network hardware available.

Storing the records in a hashtable (and similarly in a B-Tree) raises the question of inlining. When the data is inlined, lookups can be as fast as a single one-sided RDMA. A larger neighbourhood size for an inlined table minimizes the number of RDMA transfers. At the same time, however, it results in larger RDMA transfers, which as we showed can reduce performance. There is a trade-off between keeping the neighbourhood small and reading as many records per hash index



(a) 90% primary index - 10% secondary index lookups.



(b) 10% primary index - 90% secondary index lookups.

Figure 5: Performance of 256-byte row lookup mix.

lookup as possible. Figure 4 presents the performance of primary index lookups using one-sided RDMA on two inlined tables, with rows of 16 and 256 bytes. For the smaller row size, a large neighbourhood of 8 rows per hash bucket achieves the highest throughput, while for larger records the best performance is achieved with a neighbourhood size of 2, and the throughput gains over choosing a neighbourhood of size 8 is 70% on 50 machines. Inlining does not create multiple copies of the data: if the data resides in the primary index, it is stored only there, and any secondary index merely holds the key of the data in primary one. This is possible in non-traditional systems, such as the ones we study, where there is no notion of "tables" and data can be stored anywhere.

Alternatively, the data could be stored outside of the index. Pointers to data are typically smaller than the rows (e.g. they are 8 bytes in FaRM). Holding the data outside the index incurs extra memory accesses. However, records are not only accessed through a primary index (i.e. through the primary key of a relational table), and operations are not always read-only. For example, records can be accessed through secondary indexes, with the results of a lookup corresponding to multiple rows. Retrieving these rows involves either looking up the records in the primary index (if data is inlined), or reading them directly from the memory (if data is non-inlined), which is more efficient.

Figure 5 compares the performance of a read-only workload with two different transaction mixes: (1) 90% accesses through the primary index and 10% through a secondary one, and (2) 90% accesses through a secondary index and 10% through the primary. Rows are 256 bytes. The best configuration for the first transaction mix is to inline data and use a neighbourhood size of 2. Using the same configuration parameters for the second transaction mix yields up to 12.4% lower throughput than the optimal one, which is to not inline the data and use a neighbourhood size of 16.

To choose a configuration for a given table and transaction mix, SPADE works as follows. After translating all the queries of a workload into low-level distributed operations, SPADE groups operations per relational table (based on the workload schema). For each table T, SPADE calculates the weighted average of both throughput and latency:

$$Throughput(T) = \sum_i r_i(T) * Thr_i(T)$$

$$Latency(T) = \sum_i r_i(T) * Lat_i(T)$$

Where $r_i(T)$ is the ratio of the low-level operation i (e.g., lookup through a hash index, update through a secondary index, etc.) across all the operations on table T:

$$r_i(T) = \frac{N_i(T)}{\sum_j N_j(T)}$$

$N_i(T)$ is the number of occurrences of the operation i on table T. If metadata about the query mix is available (we use such metadata in our evaluation), $N_i(T)$ is adjusted by the frequency of each query. SPADE iterates over the parameter space, using the microbenchmark performance numbers for each operation – $Thr_i(T)$ and $Lat_i(T)$. It then tries to maximize $Throughput(T)$ or minimize $Latency(T)$. We leave more elaborate heuristics in SPADE for future work.

4.2 Partitioning and remote access tuning

An important decision in scale-out systems is partitioning: data should be sharded in such a way that all transactions access local data. With modern datacenter networks, partitioning is no longer the main performance factor but rather an optimization: accessing remote memory is fast, but still an order of magnitude slower than accessing local memory [54].

There is a large body of work on data partitioning [2, 3, 31, 33, 36, 51]. SPADE is not meant to replace these systems. However, as we use a shared-memory compiler, we implement a simple partitioning scheme in SPADE. Before the tuning process, SPADE groups table accesses per transaction. It then identifies inputs that are used to access more than one tables, creating groups of tables accessed with the same primary key, or common parts of their primary keys. SPADE keeps these common keys that act as *links* in this grouping. It then identifies intermediary outputs of a stored procedure used to access tables (e.g., foreign keys) and further groups tables. Finally, SPADE groups commonly accessed groups, and uses the *links* identified as the partitioning key. In FaRM, the distributed data structures used as indexes (the hashtable and B-Tree) support partitioning keys, which allows SPADE to collocate parts of tables that will be accessed together.

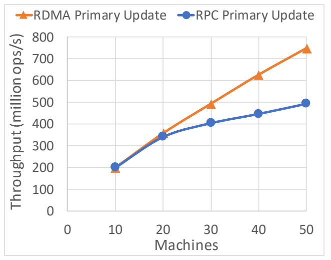
For example, in Listing 4, by examining the *UPDATE_SUBSCRIBER_DATA* transaction of TATP, SPADE can identify that records of the *Subscriber* and *Special_Facility* tables are accessed together based on the *s_id*. Thus, a grouping between the two is created, with *s_id* being the *link*. After all the tables are processed, SPADE partitions the *Subscriber* and *Special_Facility* tables with *s_id* as the partitioning key of the hash indexes used for both tables. Thus, accesses local to a specific subscriber, will always result in local accesses to the special facilities that correspond to that subscriber.

We believe that more elaborate partitioning and collocation heuristics can be included in the SQL compiler and provide additional inputs to SPADE. We thus focus on tuning remote accesses for transactions.

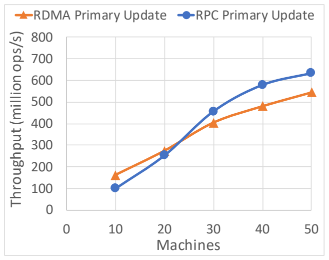
```
UPDATE Subscriber
SET bit_1 = <rnd>
WHERE s_id = <rnd subid>;
UPDATE Special_Facility
SET data_a = <rnd>
WHERE s_id = <s_id value subid>
  AND sf_type = <rnd>;
```

Listing 4: TATP's *UPDATE_SUBSCRIBER_DATA*

In the general case, it is unavoidable for some workloads to access remote data. SPADE distinguishes two cases: a) a transaction refers to remote data, which is all on the same machine or b) a transaction refers to a combination of local and remote data residing on multiple machines. SPADE further optimizes for case (a). There are two possible ways to execute a transaction that only accesses data on a single remote machine. The first is to utilize RDMA to fetch the necessary data, invoking the remote protocol at the end of the transaction. The second approach is to do an RPC to the remote machine, have the transaction executed on that machine, and any results shipped to the invoking machine. The performance of even simple operations differs between these two choices. Figure 6 shows the performance of a single-row update for two different row sizes, 16 and 256 bytes. In these measurements, we include the cost of replication (we use a replication factor of 3, similarly to our evaluation setup). Persistence is also included, since we assume main memory that has power support, as described in 2.1 and implemented in FaRM. This example assumes that we are not updating the full row, but a single field in it, which is the case for many workloads. While for small row-sizes accessing the row through RDMA and invoking the distributed commit protocol achieves higher throughput, for the 256-byte rows sending an RPC and invoking the commit protocol at the machine where the data resides is 16% faster. The difference in performance only becomes bigger as the transactions to be executed become longer and touch more data.



(a) 16-byte rows.



(b) 256-byte rows.

Figure 6: Performance of RDMA and RPC for a single-row update operation.

SPADE optimizes remote transaction execution using a similar technique as before. Instead of grouping operations per relational table, SPADE examines low-level operations (including distributed data structure accesses) of a single workload transaction, possibly consisting of multiple queries. For each transaction Tx, it calculates the weighted average of both throughput and latency:

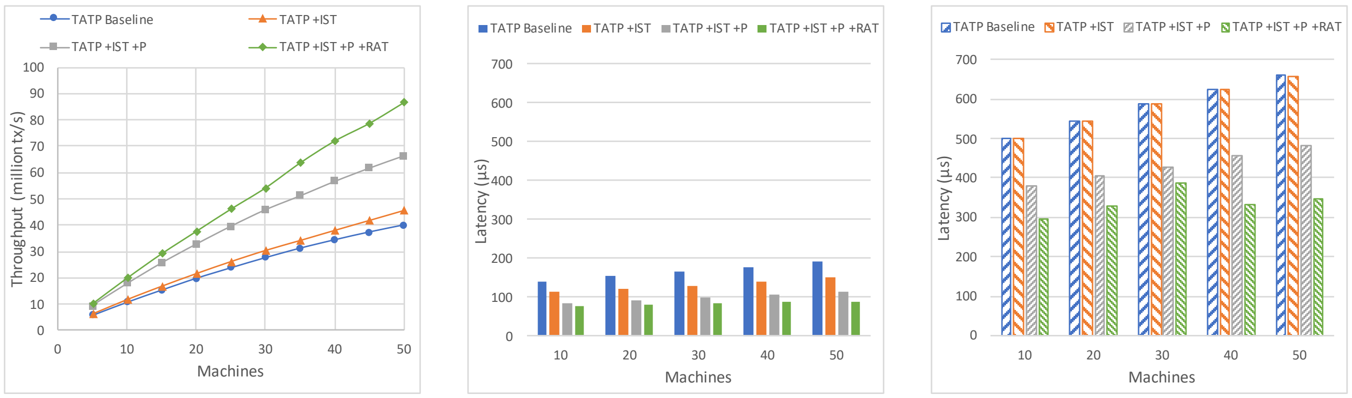
$$Throughput(Tx) = \sum_i r_i * Thr_i(Tx)$$

$$Latency(Tx) = \sum_i r_i * Lat_i(Tx)$$

Where r_i^T is the ratio of the low-level operation i (e.g., lookup through a hash index, update through a secondary index, etc.) over all the operations in transaction Tx:

$$r_i = \frac{N_i(Tx)}{\sum_j N_j(Tx)}$$

$N_i(Tx)$ is the number of occurrences of the operation i in transaction Tx. SPADE iterates over the two available choices (RDMA accesses or RPCs) and chooses the one that either maximizes $Throughput(Tx)$ or minimizes $Latency(Tx)$. If SPADE chooses an RPC call for a transaction, it generates the code necessary for the RPC calls.



(a) Throughput. (b) Median latency. (c) 99th latency.

Figure 7: Performance of all TATP versions.

5 EVALUATION

We used SPADE to implement two popular OLTP workloads: TATP and TPC-C. We first discuss TATP, establishing a baseline implementation and then showing how the different tuning decisions included into SPADE affect performance, what wrong choices would mean for both throughput and latency. Then we connect these results to the observations of Sections 4.1 and 4.2. We also present the final results for both workloads in terms of throughput, median and 99th latency when scaling out the workloads to a cluster of 50 machines. We focus on the performance of TATP because of the different types of queries it contains: TATP has both read-only and read-write transactions, as well as transactions that touch a varied amount of data. As such, a single configuration for all tables and indexes will be far from optimal. We use such examples throughout our evaluation.

5.1 Implementation

We implement SPADE as a standalone application written in F#. Our microbenchmarks are written in C++ and executed once for a FaRM cluster. We use PostgreSQL to generate plans for our workloads in XML format, which are then used as input to SPADE. SPADE produces executable code for the queries of a workload, which is invoked by applications using a C++ API we developed for SPADE. Both applications and compiled transaction code are uploaded to a FaRM cluster for execution. At runtime, the compiled code of a stored procedure invokes the FaRM API. Fault-tolerance and partition guarantees are the same as described in [14]. Additionally, due to the power-backed memory in a FaRM cluster, data is durable at all times. As a result, our system and execution model provides ACID transactions.

5.2 Experimental setup

We run our experiments on a cluster of 50 machines. Each machine has 256 GB of DRAM and two Intel E5-2650 CPUs (16 cores / 32 threads in total). Each machine has two Mellanox ConnectX-3 56Gbps Infiniband NICs, one shared by the threads of each socket. The machines are connected to a single Mellanox SX6512 switch with full bisection bandwidth. The cluster runs Windows Server 2016. FaRM is configured with 3-way replication. All experiments are run for 60 seconds (excluding warm-up) and 3 iterations. In the graphs presented we report the average of the 3 iterations (we do not notice significant deviations).

5.3 TATP

Overview. The Telecommunications Application Transaction Processing (TATP) [30] benchmark simulates a typical Home Location Register (HLR) database in a mobile phone network. The benchmark consists of a set of seven pre-defined transactions that query, update and insert data in 4 tables, following a fixed probability for each transaction. TATP is read dominated, with 80% of read-only transactions. It also involves small transactions, with the rows read/written being between 8 and 72 bytes. Throughout our experiments, we use 50 million subscribers per machine (250 million subscribers for a cluster of size 5 and 2.5 billion subscribers for a cluster of size 50), for a total size of 1TB of data on 50 machines.

In all our experiments, we scale TATP to 50 machines. To showcase the effect of tuning decisions, we write a baseline implementation of TATP and then use SPADE to tune the different parts of the workload. We repeat the experiments and report the difference in throughput and latency (median and 99th percentile). Figure 7 shows the throughput and latency results of our experiments.

Baseline. We begin by implementing a first version of TATP, using the transactions as described in the specification. Our sample application invokes a random number generator, and based on the workload mix described in the TATP specification, executes the different transactions.

For this baseline, the data is not partitioned, with tables distributed across the cluster in a round-robin fashion. Remote accesses use one-sided RDMA reads and writes. All indexes are configured with the same parameters, which favor small row sizes. We refer to this version as *TATP Baseline*. *TATP Baseline* achieves a throughput of 39.9 million committed transactions per second on 50 machines, with a median latency of 193 microseconds and a 99th percentile latency of 660 microseconds.

Index tuning. We use SPADE to tune the indexes in TATP, as described in Section 4.1. We refer to this version as *TATP +IST* (TATP with Index and Storage Tuning) in the remainder of this section. SPADE uses hash indexes for the primary keys. The 4 tables of TATP have different characteristics in terms of row sizes and accesses. For example, the *Access_Info* table has a row size of 10 bytes, and is only read by the queries. In contrast, the *Call_Forwarding* rows are 20 bytes and the rows are read, inserted, and deleted. Correctly configuring each index improves throughput by 10%-15% and latency by 19%-22% across different cluster sizes – for a cluster size of 50, the difference in throughput is 14.7% (45.79 million tx/s versus 39.91).

Partitioning. Next, we partition the TATP workload with SPADE. Most accesses are done based on a specific subscriber (the *s_id* of the subscriber table). Thus, SPADE partitions the tables so that the rows of all the tables that refer to a subscriber are stored on the same machine. Transactions access local data when invoked on the machine that stores the subscriber affected. We hardcode 10% of the transactions to reference remote subscribers, ensuring that there are remote accesses across the cluster. All remote accesses are executed using RPCs: a machine first looks up the machine storing the data, and then does an RPC to that machine to execute the transaction and return the result. We refer to this implementation as *TATP +IST +P* (TATP +IST with Partitioning).

In Figure 7, we show the improvement over *TATP +IST*. The results are intuitive: By avoiding remote accesses, the majority of transactions is executed locally, avoiding communication through RPCs. In our implementation, when a transaction accesses remote data, the machine processing the transaction does an RPC to the machine holding all the data necessary to execute the transaction, so the accesses to memory are local (except for the shipping of results). The improvement in throughput is between 45% and 50%, while latency is improved between 23% and 25% for the median and between 22% and 28% for the 99th percentile latency – for a cluster size of 50, throughput increases by 20.58 million tx/s and median and 99th latency are lower by 37 and 174 us respectively. The improvement is significant. However, it is comparable to that between *TATP Baseline* and *TATP +IST*. In the past crossing partitions and accessing remote data used to be prohibitively expensive. As this experiment shows, with modern network interconnects, the effect of remote accesses becomes smaller, and partitioning can be viewed as an optimization, rather than a requirement.

Transactions benefit from SPADE's tuning to different extents. For instance, for the *GET_ACCESS_DATA* transaction (shown in Listing 5) the median latency is 25% lower, but the 99th percentile latency is increased by 40-53% (Figure 8). This is because the query is read-only, and accesses a single row of the table. Doing an RPC for remote data adds an extra overhead to the transaction: the data is shipped to the calling machine and by using an RPC we additionally involve the remote CPU. However this is the case only because of the size of the returned data (10 bytes) and the length of the

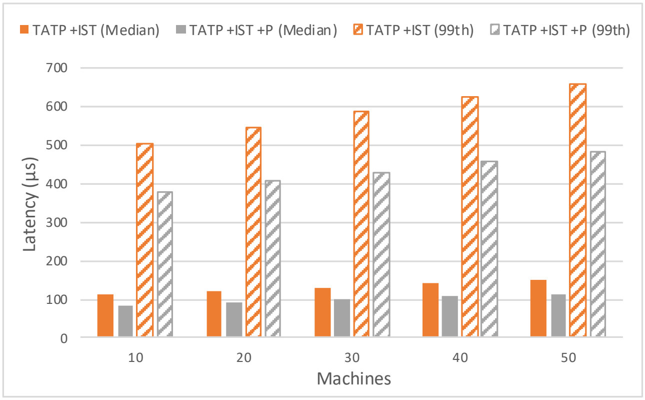


Figure 8: TATP's *GET_ACCESS_DATA* transaction latency after partitioning.

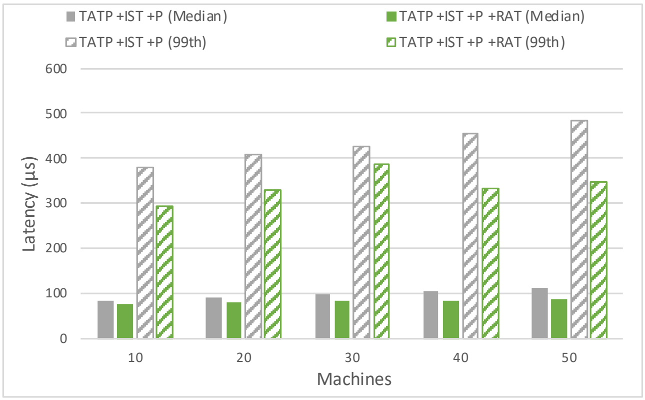
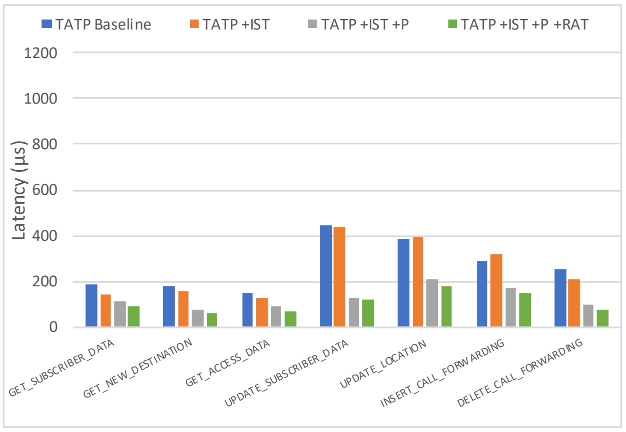
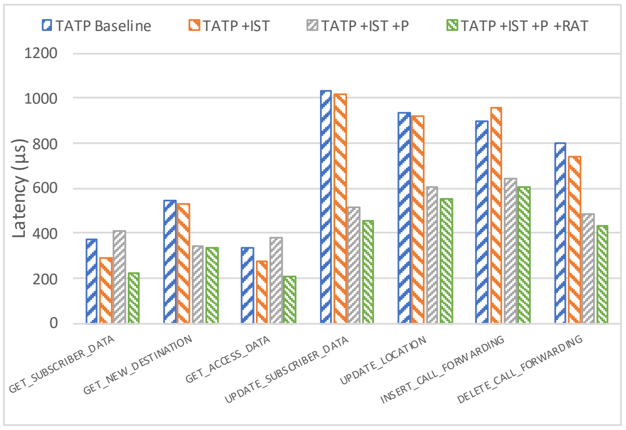


Figure 9: TATP's *GET_ACCESS_DATA* transaction latency with tuned remote accesses.



(a) Median latency.



(b) 99th latency.

Figure 10: Overview of the impact of different physical design choices to TATP operation latency.

transaction itself (a single lookup). Because of the partitioned workload, in *TATP +IST +P* version, 90% of the invocations of this transaction will access local data, lowering the median latency, while the 10% of the cases where remote accesses are necessary will be slower than before (our *TATP +IST* version uses one-sided RDMA operations for all remote accesses). This transaction is a perfect example of a physical design choice that is tied to the characteristics of the data and the underlying hardware, but also one that would require extensive experimentation to identify. A uniform configuration for all the remote accesses clearly is not optimal for TATP.

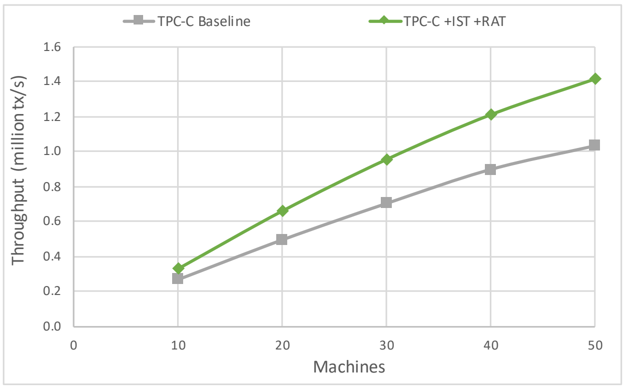
```
SELECT data1, data2, data3, data4
FROM Access_Info
WHERE s_id = <rnd> AND ai_type = <rnd>;
```

Listing 5: TATP's "GET_ACCESS_DATA" in SQL

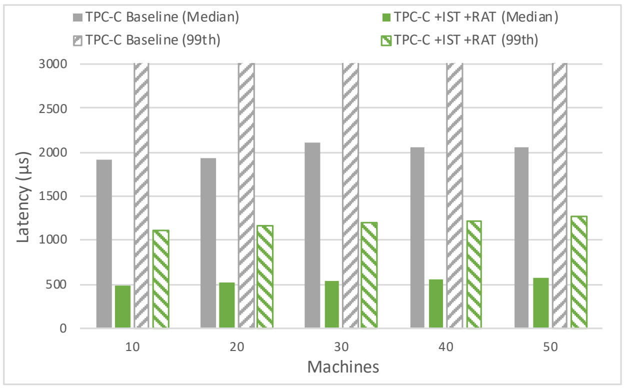
Remote access tuning. Finally, we enable the cost models of SPADE which decide on whether an RPC should be preferred over RDMA accesses (we refer to this implementation as *TATP +IST +P +RAT* (TATP +IST +P with Remote Access Tuning). Throughput is improved between 9% and 30% and latency between 5% and 22% – for a cluster of size 50, throughput is improved by 30.7% (from 66.37 to 86.72 million tx/s) and median and tail latency reduced by 26 and 135 us respectively. This is a direct result of addressing the cases such as *GET_ACCESS_DATA*, which benefit from directly accessing remote data. Figure 9 shows the improvement for this particular query (introduced before). Optimizing parts of the workload to use RDMA improves median latency between 5% and 23% and tail latency by up to 45%. Doing so manually is far from trivial. Using a cost model like the one we use in SPADE is not only easier, but also more accurate, since it can calculate the optimal setup for a set a stored procedures and a specific transaction mix.

Effect on individual operations. Figure 10 presents the median and 99th percentile latencies 50 machines on of all 7 procedures in TATP across the four versions. For 4 of the 7 transactions, each set of tuning decisions improves median latency, lowering it by as much as 70%. For both *UPDATE_SUBSCRIBER_DATA* and *UPDATE_LOCATION*, the differences when optimizing indexes (*TATP +IST)* are small: for the Subscriber table, SPADE stores data outside of the primary hash index. Thus, both the primary index and the secondary index on *sub_nbr* store pointers to the objects. As a result, lookups through the secondary index are significantly faster, but updates become slower, since they require more operations (as opposed to inlined data). However, the total effect on the workload is positive. If the transaction mix was different, SPADE might choose to inline the *Subscriber* table, in order to achieve an overall better performance.

Similarly, each set of tuning decisions improves the 99th percentile latency of transactions, with two exceptions. For *GET_SUBSCRIBER_DATA* and *GET_ACCESS_DATA*, using only RPCs for transactions accessing remote data increases 99th percentile latency by 40.9% and 40.1% respectively. Both transactions are read-only, touching small amounts of data. Doing an RPC and receiving the results translates to higher latency than directly reading the necessary data using one-sided RDMA reads. The transfer latency for the data will be the same, however, using message-passing will additionally involve the remote CPU, which will have to process the request and send the response. For both transactions, when switching to RDMA reads for remote accesses, median latency improves by 23% (from 116 to 89 us for *GET_SUBSCRIBER_DATA* and from 94 to 72 us for *GET_ACCESS_DATA*), and at the same time tail latency is lowered by 45% (from 410 to 224 us and from 384 to 208 respectively for the two queries).



(a) Throughput.



(b) Median and 99th latency.

Figure 11: Performance of *TPC-C* and remote access tuning.

5.4 TPC-C

Overview. The Transaction Processing Performance Council Benchmark C (TPC-C) [48] is a long-studied OLTP benchmark. TPC-C emulates the activity of a wholesale supplier, with tables that contain data on stock, customers, new orders, and more. The workload consists of five pre-defined transactions of different types and complexity, which can access up to hundreds of rows. In contrast to TATP, TPC-C is write-intensive, with 35.8% of transactions modifying data. Throughout our experiments, we use 120 warehouses per machine, for a total of 6000 warehouses on 50 machines.

We scale TPC-C to 50 machines. Since TPC-C is partitionable by design (the granularity of partitioning is a warehouse), we do not use SPADE to partition the workload, and focus on remote transaction processing, using either RPCs or RDMA accesses. We collect statistics for all the transactions of the workload and report throughput and latency for "new order" transactions only.

Effect of remote accesses. Figure 11 depicts the effect of optimizing remote accesses for TPC-C. In TPC-C warehouses are the granularity of partitioning: 90% the queries reference data that corresponds to a local warehouse, as such they result in local data accesses. Thus, we begin our tuning process from the *TPC-C Baseline* implementation, which uses the same configuration for all indexes and RDMA accesses for all remote data, and then invokes the distributed commit protocol. Since we are using FaRM for our implementation, during commit, read values are validated to have not changed, and modified values are shipped to the machine storing the data (and then replicated to backups). The second version, named *TPC-C +IST +RAT*, is the result of using SPADE to tune indexes and remote accesses. SPADE decides that RPCs are better for all the transactions accessing remote data in TPC-C. Using RDMA for transactions that reference remote data yields throughput that is lower by up to 37.7% (1.03 versus 1.42 million tx/s on 50 machines), while median and tail latency are up to 298% and 395% higher respectively (1479 and 4457 us higher respectively on 50 machines). All the queries executed in TPC-C involve numerous data accesses. Using RDMA decreases throughput and increases latency. Additionally, due to longer transactions, throughput is further lowered, due to the closed-loop architecture of the TPC-C benchmark. Adding concurrent transactions improves throughput slightly, but at the cost of higher latency.

6 RELATED WORK

6.1 Distributed transactions and modern hardware

Distributed transactions on traditional hardware and networks has a long history of research. However, in the past, networks imposed a significant bottleneck that dominated performance. Thus, a large body of research has focused on partitioning schemes [10, 32, 34, 37, 39] to avoid crossing partitions, or relaxation of the consistency of transactions [22].

Recently, a number of research projects and commercial systems have taken advantage of modern network characteristics, such as high throughput and low latencies, as well as RDMA capabilities, to implement fast distributed transactions. FaRM [13, 14] uses one-sided RDMA verbs and optimistic concurrency control to implement serializable distributed transactions. FaRM uses primary-backup replication to ensure fault-tolerance, a fast failure detection mechanism, and a recovery protocol that ensures that transactions are not blocked when failures occur. FaSST [19] implements serializable distributed transactions on top of unreliable, unordered, non-congestion-friendly RPC implementation. FaSST also uses primary-backup replication and a similar optimistic commit protocol. Tell [26] decouples processing nodes from storage nodes, using one-sided RDMA accesses to ship data to processing nodes. Tell implements a distributed snapshot isolation protocol that uses a centralized commit manager. NAM-DB [54] also separates computation from storage, however it uses a distributed snapshot isolation algorithm that relies on a scalable timestamp oracle. DrTM+R [8] provides serializable transactions by using both RDMA accesses and Hardware Transactional Memory (HTM), relying on primary-backup and an optimistic replication scheme.

All the above systems [8, 13, 14, 19, 26, 54] utilize modern hardware to implement distributed transactions. Moreover, they all include distributed index structures. SPADE optimizes for the physical design of data, such as tuning indexes and remote accesses, which is applicable to distributed systems that leverage modern RDMA networks in general.

6.2 Index tuning

Databases require a significant amount of tuning to achieve performance. Research on automatically tuning indexes has been meticulous [2, 16, 21, 35, 50]. Additionally, DBMS vendors provide tools that aid administrators in tuning a database. Microsoft's SQL Server has the Database Tuning Advisor [1], which can both tune existing indexes for performance, as well as recommend new ones. Oracle Server has both self-tuning [7, 12, 55] and performance analysis capabilities [53]. DB2 has offered the Performance Wizard tool [23]. Recent research projects have also looked into machine learning [5], adaptive sampling [15] and regression models [46] to choose the best parameters for DBMSs.

In SPADE we focus on a distributed setting. We argue that, in addition to previous efforts, modern distributed transaction systems require an additional level of tuning. SPADE takes into account the network and hardware characteristics to tune index parameters. This way, applications take full advantage of the performance of modern networks, which, as we show, can have a significant impact on performance.

6.3 Partitioning and remote accesses

Partitioning has played a role of paramount importance in the past for distributed transactions. Schism [10] is an off-line approach to data partitioning for distributed shared-nothing databases, which minimizes the number of distributed transactions. SWORD [37] employs an incremental data repartitioning technique for OLTP workloads in database-as-a-service cloud settings. Pavlo et al. [34] automatically partition workloads using large neighborhood search and analytical models. Clay [39] is an on-line partitioning approach that uses dynamic blocks to incrementally partition data minimizing the number of distributed transactions. JECB [47] uses a divide-and-conquer strategy to partition workloads for large clusters. Sun et al. [44] use a bottom-up approximate approach to partition data.

With the advent of fast networks that support one-sided remote memory operations, partitioning is no longer the determining factor, but rather an optimization. With new ways to access remote data, partitioning now imposes a new problem, that of determining how to do so in a way that takes full advantage of the network characteristics. As such, SPADE's optimization of remote accesses plays an important role, as we show in our evaluation.

7 CONCLUSIONS

This paper introduces SPADE, a physical design tuner for OLTP workloads in FaRM. SPADE automates choices regarding the tuning of indexes, the partitioning of data, and remote accesses, shipping data, or the computation through RPCs for different parts of the workload. SPADE achieves this by analyzing the physical plan of a workload, and using hardware and network performance characteristics gathered through microbenchmarks.

Our evaluation of SPADE showed that these choices have a significant impact on both throughput and latency. Index and storage tuning can improve throughput by up to 15%, partitioning by up to 50% and remote access tuning by up to 31%. Combined, these tuning decisions improve throughput by as much as 117%, while lowering latency by up to 75% compared to a workload with basic tuning.

ACKNOWLEDGEMENTS

We would like to thank our shepherd, José Orlando Pereira, as well as the anonymous reviewers for their helpful comments on improving the paper. We would also like to thank Mirjana Pavlovic, Georgios Psaropoulos, Danica Porobic and Manos Karpathiotakis for their feedback on the paper. Part of this work was supported by the European Research Council (ERC) Grant 339539 (AOC) and the "Revisiting Transactional Computing on Modern Hardware" project of the Microsoft-EPFL Joint Research Center.

REFERENCES

[1] Agrawal, S., Chaudhuri, S., Kollár, L., Marathe, A. P., Narasayya, V. R., and Syamala, M. Database tuning advisor for microsoft SQL server 2005. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004* (2004), pp. 1110–1121.

[2] Agrawal, S., Chu, E., and Narasayya, V. R. Automatic physical design tuning: workload as a sequence. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006* (2006), pp. 683–694.

[3] Agrawal, S., Narasayya, V. R., and Yang, B. Integrating vertical and horizontal partitioning into automated physical database design. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004* (2004), pp. 359–370.

[4] Aguilera, M. K., Leners, J. B., and Walfish, M. Yesquel: scalable sql storage for web applications. In *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015, Monterey, CA, USA, October 4-7, 2015* (2015), pp. 245–262.

[5] Aken, D. V., Pavlo, A., Gordon, G. J., and Zhang, B. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017* (2017), pp. 1009–1024.

[6] Barthels, C., Loesing, S., Alonso, G., and Kossmann, D. Rack-scale in-memory join processing using RDMA. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015* (2015), pp. 1463–1475.

[7] Belknap, P., Dageville, B., Dias, K., and Yagoub, K. Self-tuning for SQL performance in oracle database 11g. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China* (2009), pp. 1694–1700.

[8] Chen, Y., Wei, X., Shi, J., Chen, R., and Chen, H. Fast and general distributed transactions using RDMA and HTM. In *Proceedings of the Eleventh European Conference on Computer Systems, EuroSys 2016, London, United Kingdom, April 18-21, 2016* (2016), pp. 26:1–26:17.

[9] Comer, D. The ubiquitous b-tree. *ACM Comput. Surv. 11*, 2 (1979), 121–137.

[10] Curino, C., Zhang, Y., Jones, E. P. C., and Madden, S. Schism: a workload-driven approach to database replication and partitioning. *PVLDB 3*, 1 (2010), 48–57.

[11] Diaconu, C., Freedman, C., Ismert, E., Larson, P., Mittal, P., Stonecipher, R., Verma, N., and Zwilling, M. Hekaton: SQL server's memory-optimized OLTP engine. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013* (2013), pp. 1243–1254.

[12] Dias, K., Ramacher, M., Shaft, U., Venkataramani, V., and Wood, G. Automatic performance diagnosis and tuning in oracle. In *CIDR 2005, Second Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2005, Online Proceedings* (2005), pp. 84–94.

[13] Dragojevic, A., Narayanan, D., Castro, M., and Hodson, O. 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* (2014), pp. 401–414.

[14] Dragojevic, A., Narayanan, D., Nightingale, E. B., Renzelmann, M., Shamis, A., Badam, A., and Castro, M. No compromises: distributed transactions with consistency, availability, and performance. In *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015, Monterey, CA, USA, October 4-7, 2015* (2015), pp. 54–70.

[15] Duan, S., Thummala, V., and Babu, S. Tuning database configuration parameters with ituned. *PVLDB 2*, 1 (2009), 1246–1257.

[16] Gankidi, V. R., Teletia, N., Patel, J. M., Halverson, A., and DeWitt, D. J. Indexing HDFS data in PDW: splitting the data from the index. *PVLDB 7*, 13 (2014), 1520–1528.

[17] Huang, J., Ouyang, X., Jose, J., Wasi-ur-Rahman, M., Wang, H., Luo, M., Subramoni, H., Murthy, C., and Panda, D. K. High-performance design of hbase with RDMA over infiniband. In *26th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2012, Shanghai, China, May 21-25, 2012* (2012), pp. 774–785.

[18] Kalia, A., Kaminsky, M., and Andersen, D. G. Design guidelines for high performance RDMA systems. In *2016 USENIX Annual Technical Conference, USENIX ATC 2016, Denver, CO, USA, June 22-24, 2016.* (2016), pp. 437–450.

[19] Kalia, A., Kaminsky, M., and Andersen, D. G. Fasst: Fast, scalable and simple distributed transactions with two-sided (RDMA) datagram rpcs. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.* (2016), pp. 185–201.

[20] Kallman, R., Kimura, H., Natkins, J., Pavlo, A., Rasin, A., Zdonik, S. B., Jones, E. P. C., Madden, S., Stonebraker, M., Zhang, Y., Hugg, J., and Abadi, D. J. H-store: a high-performance, distributed main memory transaction processing system. *PVLDB 1*, 2 (2008), 1496–1499.

[21] Kargin, Y., Kersten, M. L., Manegold, S., and Pirk, H. The DBMS - your big data sommelier. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015* (2015), pp. 1119–1130.

[22] Kraska, T., Hentschel, M., Alonso, G., and Kossmann, D. Consistency rationing in the cloud: Pay only when it matters. *PVLDB 2*, 1 (2009), 253–264.

[23] Kwan, E., Lightstone, S., Storm, A., and Wu, L. Automatic configuration for ibm db2 universal database. In *Proc. of IBM Perf Technical Report* (2002).

[24] Lee, B. C., Ipek, E., Mutlu, O., and Burger, D. Architecting phase change memory as a scalable dram alternative. In *36th International Symposium on Computer Architecture (ISCA 2009), June 20-24, 2009, Austin, TX, USA* (2009), pp. 2–13.

[25] Li, F., Das, S., Syamala, M., and Narasayya, V. R. Accelerating relational databases by leveraging remote memory and RDMA. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016* (2016), pp. 355–370.

[26] Loesing, S., Pilman, M., Etter, T., and Kossmann, D. On the design and scalability of distributed shared-data databases. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015* (2015), pp. 663–676.

[27] Micron. 3D XPoint Technology, 2017. https://www.micron.com/products/advanced-solutions/3d-xpoint-technology.

[28] Microsoft. OCS Open CloudServer Power Supply v2.0, 2015. http://www.opencompute.org/wiki/Server/SpecsAndDesigns.

[29] Mitchell, C., Geng, Y., and Li, J. Using one-sided RDMA reads to build a fast, cpu-efficient key-value store. In *2013 USENIX Annual Technical Conference, San Jose, CA, USA, June 26-28, 2013* (2013), pp. 103–114.

[30] Neuvonen, S., Wolski, A., Manner, M., and Raatikka, V. Telecom Application Transaction Processing Benchmark. http://tatpbenchmark.sourceforge.net/.

[31] Olma, M., Karpathiotakis, M., Alagiannis, I., Athanassoulis, M., and Ailamaki, A. Slalom: Coasting through raw data via adaptive partitioning and indexing. *PVLDB 10*, 10 (2017), 1106–1117.

[32] Paiva, J., Ruivo, P., Romano, P., and Rodrigues, L. E. T. AUTO-PLACER: scalable self-tuning data placement in distributed key-value stores. In *10th International Conference on Autonomic Computing, ICAC'13, San Jose, CA, USA, June 26-28, 2013* (2013), pp. 119–131.

[33] Papadomanolakis, S., and Ailamaki, A. Autopart: Automating schema design for large scientific databases using data partitioning. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM 2004), 21-23 June 2004, Santorini Island, Greece* (2004), pp. 383–392.

[34] Pavlo, A., Curino, C., and Zdonik, S. B. Skew-aware automatic database partitioning in shared-nothing, parallel OLTP systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012* (2012), pp. 61–72.

[35] Petraki, E., Idreos, S., and Manegold, S. Holistic indexing in main-memory column-stores. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015* (2015), pp. 1153–1166.

[36] Porobic, D., Liarou, E., Tözün, P., and Ailamaki, A. Atrapos: Adaptive transaction processing on hardware islands. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014* (2014), pp. 688–699.

[37] Quamar, A., Kumar, K. A., and Deshpande, A. SWORD: scalable workload-aware data placement for transactional workloads. In *Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18-22, 2013* (2013), pp. 430–441.

[38] Qureshi, M. K., Srinivasan, V., and Rivers, J. A. Scalable high performance main memory system using phase-change memory technology. In *36th International Symposium on Computer Architecture (ISCA 2009), June 20-24, 2009, Austin, TX, USA* (2009), pp. 24–33.

[39] Serafini, M., Taft, R., Elmore, A. J., Pavlo, A., Aboulnaga, A., and Stonebraker, M. Clay: Fine-grained adaptive partitioning for general database schemas. *PVLDB 10*, 4 (2016), 445–456.

[40] Shi, J., Yao, Y., Chen, R., Chen, H., and Li, F. Fast and concurrent RDF queries with rdma-based distributed graph exploration. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.* (2016), pp. 317–332.

[41] Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hachem, N., and Helland, P. The end of an architectural era (it's time for a complete rewrite). In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007* (2007), pp. 1150–1160.

[42] Strukov, D. B., Snider, G. S., Stewart, D. R., and Williams, R. S. The missing memristor found. *Nature 453*, 7191 (2008), 80–83.

[43] Stuedi, P., Trivedi, A., and Metzler, B. Wimpy nodes with 10gbe: Leveraging one-sided operations in soft-rdma to boost memcached. In *2012 USENIX Annual Technical Conference, Boston, MA, USA, June 13-15, 2012* (2012), pp. 347–353.

[44] Sun, L., Franklin, M. J., Krishnan, S., and Xin, R. S. Fine-grained partitioning for aggressive data skipping. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014* (2014), pp. 1115–1126.

[45] Thomson, A., Diamond, T., Weng, S., Ren, K., Shao, P., and Abadi, D. J. Calvin: fast distributed transactions for partitioned database systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012* (2012), pp. 1–12.

[46] Tran, D. N., Huynh, P. C., Tay, Y. C., and Tung, A. K. H. A new approach to dynamic self-tuning of database buffers. *TOS 4*, 1 (2008), 3:1–3:25.

[47] Tran, K. Q., Naughton, J. F., Sundarmurthy, B., and Tsirogiannis, D. JECB: a join-extension, code-based approach to OLTP data partitioning. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014* (2014), pp. 39–50.

[48] Transaction Processing Performance Council (TPC). TPC Benchmark C: Standard specification. http://www.tpc.org/tpcc/.

[49] Tsai, S., and Zhang, Y. LITE kernel RDMA support for datacenter applications. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017* (2017), pp. 306–324.

[50] Voigt, H., Kissinger, T., and Lehner, W. SMIX: self-managing indexes for dynamic workloads. In *Conference on Scientific and Statistical Database Management, SSDBM '13, Baltimore, MD, USA, July 29 - 31, 2013* (2013), pp. 24:1–24:12.

[51] Wu, E., and Madden, S. Partitioning techniques for fine-grained indexing. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany* (2011), pp. 1127–1138.

[52] Xie, C., Su, C., Kapritsos, M., Wang, Y., Yaghmazadeh, N., Alvisi, L., and Mahajan, P. Salt: Combining ACID and BASE in a distributed database. In *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014.* (2014), pp. 495–509.

[53] Yagoub, K., Belknap, P., Dageville, B., Dias, K., Joshi, S., and Yu, H. Oracle's SQL performance analyzer. *IEEE Data Eng. Bull. 31*, 1 (2008), 51–58.

[54] Zamanian, E., Binnig, C., Kraska, T., and Harris, T. The end of a myth: Distributed transaction can scale. *PVLDB 10*, 6 (2017), 685–696.

[55] Zilio, D. C., Rao, J., Lightstone, S., Lohman, G. M., Storm, A. J., Garcia-Arellano, C., and Fadden, S. DB2 design advisor: Integrated automatic physical database design. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004* (2004), pp. 1087–1097.