Online Replication Strategies for Distributed Data Stores

Niklas Semmler
SAP
niklas.semmler@sap.com

Georgios Smaragdakis
TU Berlin
georgios@inet.tu-berlin.de

Anja Feldmann
Max Planck Institute for Informatics
anja@mpi-inf.mpg.de

ABSTRACT

The rate at which data is produced at the network edge, e.g., collected from sensors and Internet of Things (IoT) devices, will soon exceed the storage and processing capabilities of a single system and the capacity of the network. Thus, data will need to be collected and preprocessed in distributed data stores—as part of a distributed database—at the network edge. Yet, even in this setup, the transfer of query results will incur prohibitive costs. To further reduce the data transfers, patterns in the workloads must be exploited. Particularly in IoT scenarios, we expect data access to be highly skewed. Most data will be store-only, while a fraction will be popular. Here, the replication of popular, raw data, as opposed to the shipment of partially redundant query results, can reduce the volume of data transfers over the network.

In this paper, we design online strategies to decide between replicating data from data stores or forwarding the queries and retrieving their results. Our insight is that by profiling access patterns of the data we can lower the data transfer cost and the corresponding response times. We evaluate the benefit of our strategies using two real-world datasets.

KEYWORDS

database, data replication, IoT, computer networks

1 INTRODUCTION

Sensors and Internet of Things (IoT) devices are more ubiquitous than ever and are increasingly integrated with decision-making procedures across many industries [14]. They are installed and operate at the Internet edge [31]. While each individual sensor may produce only a negligible data volume, together they generate massive data streams. Thus, we expect a growing disparity between the data volumes produced by these sensors and the capacity of a single system. Moreover, the network capacity to move all sensor data from the network edge to the cloud may become a bottleneck. Thus, we expect this data will have to be stored in the fog [9]—data stores at or close to the network edge—where it will form distributed mega-datasets [32]. Many companies have already started the design and deployment of complex network and cloud architectures to cope with the expected volume and management overhead of these datasets. Deutsche Telekom recently announced the introduction of the mobile edge cloud, a platform for deploying application functionality at the edge [6]. Volkswagen made a public statement that it will network its factories and use the Amazon cloud to improve its production line [29].

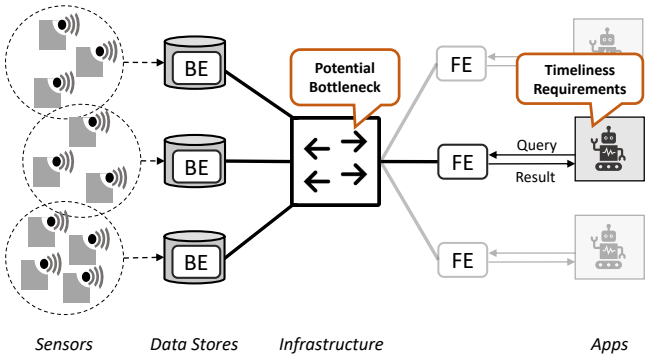


Figure 1: Architecture for IoT scenario with distributed mega-datasets. (Front-ends and back-ends are abbreviated as FE and BE respectively.)

The accumulation of sensor data gives rise to an increasing number of data-driven applications. These applications leverage the data to improve a multitude of processes ranging from pro-active maintenance to value-chain management to process optimization, which is expected to benefit many stakeholders, including end-customers, enterprises, vendors, and governments. To this end, each application needs to access the distributed data stores, which may create congestion in the network. Yet, we expect that each application will only need to access a small fraction of the data or pre-aggregated data, e.g., data from a limited number of sensors, data of a given type, data summaries, data to investigate errors and disruptions in operations. Typical for machine-generated data, most data will be very regular and therefore of little interest and never accessed. Overall, we expect that data access distributions are highly skewed, where a large fraction of the data is rarely accessed while a small fraction is very popular. We exploit this skewed distribution to minimize network traffic and ensure low response times and, thus, enable such data-driven applications for mega-datasets.

As part of our solution, we introduce a distributed platform involving potentially many front-ends and many back-ends, see Figure 1. The front-ends receive and answer incoming queries from the applications (or "Apps") using data from the back-ends. A front-end may forward a query to a back-end, which will process the query using its local data store and *ship* the result back. Alternatively, a front-end can ask a back-end to *replicate* the necessary data to its local store and, then, use this data to process queries locally.

To take full advantage of replication and to reduce the overhead, it is often more desirable to replicate the partition that contains a superset of the data necessary for processing queries instead of only the data itself. A partition, typically, contains related data and is a fixed data unit related to the storage organization. Then, the replication option has one advantage and one disadvantage. It has the advantage that it can respond to future queries that access the same data at no additional cost. In contrast, it has the disadvantage that replicated partitions requires storage at the front-end, which may only be able to store a limited number of replicas.

Depending on the size of the accessed data and size of the query result, either option may reduce the required transfer data volume over the network. Decomposing the system into distributed front-ends and back-ends enables many additional, orthogonal optimizations, such as caching and approximate query processing (AQP) [10]. These can be used to further improve application performance.

In this paper, we propose online strategies to decide when to transfer data from a remote data store to a front-end to improve the system's overall efficiency. Our contributions can be summarized as follows:

- We formalize the problem of choosing between shipping or replicating data for distributed IoT data stores with application front-ends.
- We show that this problem can be reduced to a variation of the ski-rental problem. Based on this insight, we propose two online replication strategies that decide when replicating partitions should be chosen over shipping query results.
- We evaluate the performance of our online replication strategies by applying them to two real-world query traces. Our results show that our proposed online strategies perform close to the optimal offline strategy.

2 PROBLEM FORMALIZATION

Traditionally, distributed databases rather ship query results than replicate partitions. Even modern databases make only light use of replication. They rely either on a static replication scheme or require explicit configuration by a human administrator [1, 15]. Here, we use replication as a tool to reduce the *transferred data volume*, i.e., the data volume that is moved over the network to answer queries and reduce query response times as a side effect. In this preliminary work, we focus on the interaction between a single front-end and a single back-end. We also restrict this model to a read-only workload, which is consistent with the immutable machine and sensor data often produced in the Internet of Things.

Var.	Description
Q	Set of queries over the whole runtime
Q_t	Queries received at time t
$\hat{Q}_t$	Sequence of query sets up to time t: $(Q_1, Q_2, \ldots, Q_t)$
$Q_{(t)}$	Set of queries observed up to time t: $\bigcup_{i=1}^{t} Q_i$
P	Set of partitions of the data
P_t^R	Set of partitions replicated at time t
$\hat{P}_t^R$	Sequence of partition sets replicated up to time t: $(P_1^R, P_2^R, \ldots, P_t^R)$
$P_{(t)}^R$	Set of partitions replicated up to time t: $\bigcup_{i=1}^{t} P_i^R$

Table 1: Variables

In the following, we specify, for every function, its domain and the target set (the co-domain) using the $\rightarrow$ symbol. We specify the function itself using the $\mapsto$ symbol. Where unambiguous, we omit the domain and codomain.

2.1 Query Model

In our model, the front-end receives queries over time from the set of queries Q. At any point in time t, it receives query sets Q_t where $Q_t \subseteq Q$. We refer to the history of query sets received up to the point t as $\hat{Q}_t$ and to the set of all queries received up to the point t as $Q_{(t)}$ where $Q_{(t)} \subseteq Q$. We assume that the back-end stores its data in partitions P of fixed size and that the queries can be answered using data stored in the partitions at the back-end. Partitions can be replicated to the front-end. At any point in time t we denote the set of replicated partitions as P_t^R where $P_t^R \subseteq P$. Similar to the queries, we describe the history of replicated partition sets up to time t with $\hat{P}_t^R$ and all partitions replicated at the time t with $P_{(t)}^R$ so that $P_{(t)}^R \subseteq P$. For now, we assume that partitions, which have been replicated at some point in time remain replicated. For convenience, we summarize all variables in Table 1.

We assume the use of partitions since partitions are often the result of optimizing for memory/disk accesses. The common assumption is that if a row in a partition is accessed other rows within the partition are likely to be accessed as

well. For the same reason, we see it as an appropriate transfer unit for our model.

In this paper, we assume that the cost of replicating a partition p is approximated by the size of the partition $size(p)$.

$$size\text{:}\ P \rightarrow \mathbb{N}^+ \tag{1}$$

A query q is answered using data from one or several partitions. We refer to the contributions of each partition p to the query result as the *partition transfer volume* and $result(q, p)$:

$$result\text{:}\ Q \times P \rightarrow \mathbb{N}_0^+ \tag{2}$$

Thus, we can determine the overall contribution of each partition p, or *aggregate partition transfer volume*, to queries up to time t as $record(Q_{(t)}, p)$:

$$\begin{aligned} record\text{:}\ & Q \times P \rightarrow \mathbb{N}_0^+ \\ & (Q_{(t)}, p) \mapsto \sum_{q \in Q_{(t)}} result(q, p) \end{aligned} \tag{3}$$

2.2 Cost Model

The choice of the point in time at which a partition is replicated influences the network cost in terms of transferred data volume. At any moment t, the transfer cost is the sum of the cost of replicating partitions and the cost of shipping query results:

$$\begin{aligned} cost\text{:}\ & Q \times P \rightarrow \mathbb{N}_0^+ \\ & (Q_t, P_t^R) \mapsto \sum_{p \in P_t^R} size(p) + \sum_{p \notin P_t^R} \sum_{q \in Q_t} result(q, p) \end{aligned} \tag{4}$$

The total cost is simply a summation of the cost up to that moment:

$$\begin{aligned} total\text{-}cost\text{:}\ & 2^Q \times 2^P \rightarrow \mathbb{N}_0^+ \\ & (\hat{Q}_t, \hat{P}_t^R) \mapsto \sum_{i=1}^{t} cost(Q_i, P_i^R) \end{aligned} \tag{5}$$

2.3 Strategies

The optimal point in time at which a partition should be replicated depends on the number of queries that a partition will receive after this point. This information is usually unknown in advance. Thus, our problem has to be solved online [2].

An online strategy in our model is a method which decides which partition should be replicated at any given moment based on the previously seen queries and its previous decisions.

$$\begin{aligned} strategy\text{:}\ & 2^Q \times P \rightarrow P \\ & (\hat{Q}_t, P_{(t)}^R) \mapsto P_{t+1}^R \end{aligned} \tag{6}$$

Two naïve strategies are to use either only replication or only shipping and disregard any knowledge gained from previous queries.

$$replicate\text{-}only\ (RO)\text{:}\ (Q_t, P_t^R) \mapsto P \tag{7}$$

$$ship\text{-}only\ (SO)\text{:}\ (Q_t, P_t^R) \mapsto \emptyset \tag{8}$$

Our goal is to devise strategies that improve on these basic strategies and come close to the *optimal offline strategy*. The optimal solution is to replicate partitions, whose record accumulates, over the whole course of time, accesses that exceed the cost of replicating the partition—the partition size. For this purpose, we consider finite query sequences of length n. The optimal strategy is then:

$$\begin{aligned} optimal\ (OO)\text{:}\ & (Q_t, P_t^R) \mapsto \{p | p \in P \\ & \wedge\ record(Q_{(n)}, p) > size(p)\} \end{aligned} \tag{9}$$

For some use cases, storing the full access history of queries may prove to be prohibitively expensive. In these cases, only the aggregate partition transfer volume can be stored. The strategies presented in this paper would still work although with possibly reduced accuracy.

3 ONLINE STRATEGIES

Given that we deal with an online problem where an optimal strategy has to be selected at any point in time without knowledge of the future, we turn to the area of online algorithms [19]. Our replication problem resembles the ski rental problem [19, 26]. In this problem, a skier faces the choice between buying a ski-set and renting a ski-set on every day of his skiing career. Buying the ski set has the advantage that no future cost for renting will accrue. However, in the worst case, the skier stops skiing on the same day (for whatever reason). Then, the sum of money spent on buying was almost "wasted". Renting the ski-set in this situation would have been cheaper. But, if she keeps on skiing, the accumulated cost of renting can easily exceed the cost of buying them in the long run. The third option is to switch from renting to buying after a number of skiing days or after a certain sum has been spent on renting. Yet, choosing this threshold to minimize the absolute cost in advance, without knowledge of the future, is impossible. This ski rental problem is similar to our problem, whereby, the small renting cost is analogous to the cost of shipping query results, the cost of buying ski

is analogous to replicating a partition and the workload is not known in advance.

Even though future events in the ski rental problem are unknown, the worst-case is known and can be summarized in the form of the competitive ratio. The competitive ratio captures the performance of an online algorithm in comparison to an offline one that has perfect knowledge of the future. It can be shown [26] that buying the ski after spending the same amount on renting, means that the skier pays at most twice the sum an all-knowing actor would have spent. Similarly, we too know, that replicating a partition after a data volume equal its size has been shipped is at most twice the cost an all-knowing actor would have spent. To capture this, we introduce *threshold strategies* with the parameter threshold τ:

$$\begin{aligned} threshold(\tau)\colon 2^Q \times P &\longrightarrow P \\ (Q_{(t)}, P_t^R) &\mapsto \{p | p \in P \\ &\wedge\ record(Q_{(t)}, p) > \tau\} \end{aligned} \tag{10}$$

The threshold strategy replicates a partition once it is responsible for a transferred data volume equal to τ. The *ski rental strategy (SR)* is one example of a threshold strategy where the threshold is the size of the partition. If all partitions have the same size, this is a single value. If they have different sizes, then the threshold corresponds to their individual sizes.

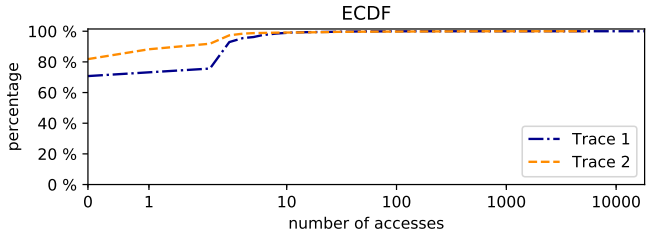
A key difference between our problem and the ski rental problem is that our problem includes multiple partitions. We can either treat them as separate ski rental problems or we can consider them together. In the latter case, we may be able to estimate the distribution of accesses over the partitions. This approach is related to the constrained ski rental problem [20, 25, 27]. Here, the probability for the number of skiing days is known to follow a given probability distribution. This allows the creation of a threshold-based strategy for the average case in terms of the expected cost [20, 27].

Thus, we estimate the distribution of aggregate partition transfer volumes from the history of accesses (the *record* in our model) which results in the *reactive threshold strategy (RT)*. This is another example of the threshold strategy, but one that uses a threshold $\tau(t)$ which changes over time. We call threshold strategies that use different thresholds over time *dynamic* in contrast to *static* strategies that use only a single threshold. For RT, the threshold is estimated from the existing query history up to present time t.

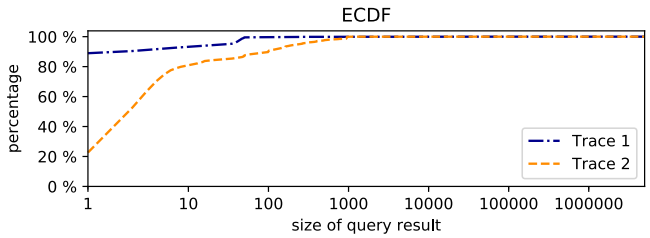
$$\tau_{RT}(t) \mapsto argmin_\tau \sum_{i=1}^{t} cost(Q_i, threshold(\tau)(Q_{i-1}, P_{i-1}^R)) \tag{11}$$

Name	Trace 1	Trace 2
Table size in rows [million]	24	100
Number queries [million]	1.28	2.49
Duration [days]	≈ 3	≈ 3
Accesses in rows [million]	34	137
Avg. rows per query	26	55

Table 2: ERP traces statistics.



(a) ECDF of number of times that individual rows are accessed.



(b) ECDF of result sizes per queries.

Figure 2: Trace characterization.

Given a finite query sequence of length n, we can use the same method to compute the optimal threshold for this query sequence. We refer to the strategy that uses this threshold as the *optimal offline threshold strategy*. Its performance is an *upper bound* for all *static threshold strategies*.

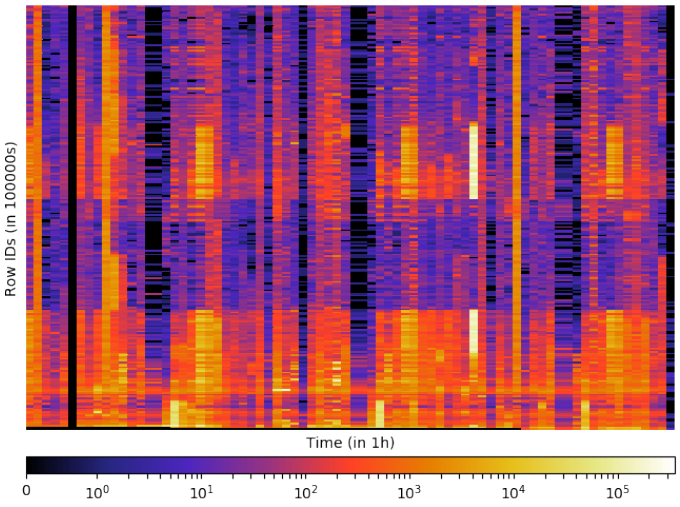
$$\tau_{OT}(t) \mapsto \tau_{RT}(n) \tag{12}$$

Table 3 summarizes all strategies and their parameters.

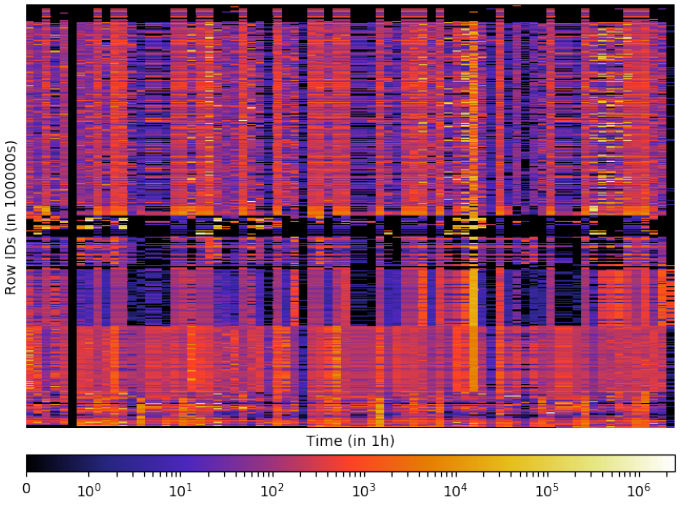
4 DATASETS

Ideally, we would like to deploy and evaluate our strategies in the real world. However, this is impossible as IoT deployments at this scale are still emerging and the currently existing often do not require a distributed setting. Thus, we do a query trace-based evaluation. For this, the query trace should (a) come from a setting with a partitioned data store and (b) include the size of query results by partition.

Unfortunately, traces with this degree of detail are rare for a variety of reasons. First, current database systems, typically, only include statistics on how often a specific database partition was accessed. Second, even when they can record



(a) Trace 1+.



(b) Trace 2+.

Figure 3: Heatmaps of the number of accesses to each partition across time periods.

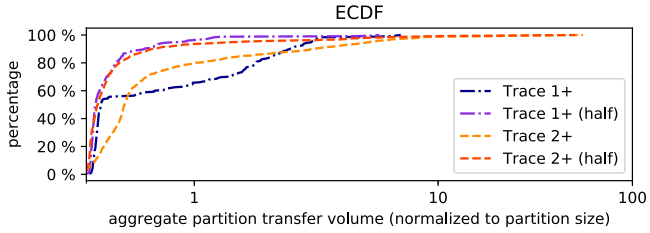


Figure 4: ECDF of aggregate partition transfer volumes.

per query accesses this feature is typically disabled for performance reasons. Third, even if such a trace exists it often contains private, business-critical information and, hence, their owners rarely share them, even for research purposes. Yet, the alternative of relying on synthetic datasets also has the major disadvantage that the data is often much more regular than in the real world [24].

Against all these odds, we were able to get access to two large database query traces with all required details about the queries and the results. Both traces were gathered by Martin Boissier et al. [8] by instrumenting a live production SAP-based Enterprise Resource Planning (ERP) system of a Global 2000 company. Each trace was collected using a two-step process. In the first step, all queries to one table within a live ERP database were recorded for a three-day time period. To reduce the overhead on the production system this trace was sub-sampled to only contain queries that appeared within the first two minutes of every ten-minute time window. The result of this step is a sequence of queries with their timestamps. In a second step, the queries were re-run against a copy of the live system to determine the size of their result set, including all rows that are part of the result set. Note, these tables, in contrast to our assumed setting, are not immutable per se. However, since the result rows are recorded at the level of row ids and since these row ids are not reused, the data is a decent approximation for our use case. Each trace contains more than 1 million queries and accesses more than 20 million rows, which corresponds on average to 26 resp. 55 row accesses per query for Trace 1/2. See Table 2 for a summary.

Even though the traces were recorded from an ERP system rather than an IoT system, we find that the access distribution per row is highly skewed. Figure 2a depicts the empirical cumulative distribution function (ECDF) of the accesses per row id for both traces. Amazingly, more than 70%/80% of the rows are never accessed at all. More than 95% of the rows are accessed less than ten times. Only a small fraction (less than 1%) of the rows are very popular with more than 100 accesses. Query result sizes are heavily skewed as well, see Figure 2b which does not even show queries with empty result set sizes. The result sets of more than 80% of the queries include less than 10 rows. Hereby we note that Trace 1 is significantly more skewed than Trace 2. In conclusion, the access patterns and the result sizes of both traces are highly skewed, which supports our earlier assumption. IoT use cases are likely to be even more skewed and, thus, would likely give even better results for our approach.

Recall, our presumption that data is organized in partitions. Yet, the traces do not contain any information about partitions. To nevertheless use the trace for our evaluation, we add partitions to the trace, whereby each partition contains 100K adjacent rows. Similar results (not shown) apply

Short	Strategy	Description
RO	replicate only	Replicate immediately
SO	shipping only	Never replicate
SR	ski rental	Threshold strategy where $\tau = size(p)$
OT	optimal thresh.	Optimal offline threshold
RT	reactive thresh.	Choose threshold that would have worked best for past queries
OO	optimal offline	Lower bound

Table 3: Strategy overview.

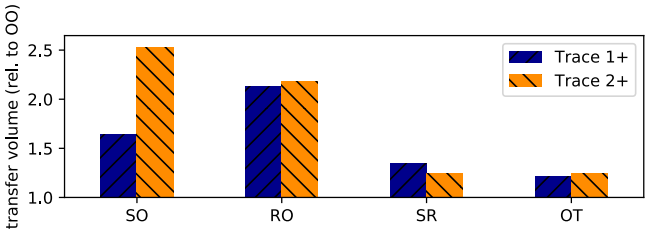
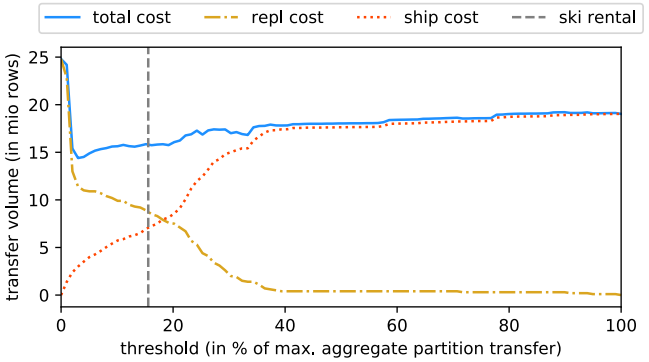


Figure 5: Transfer volume for all static threshold strategies normalized to the OO (optimal offline) strategy.

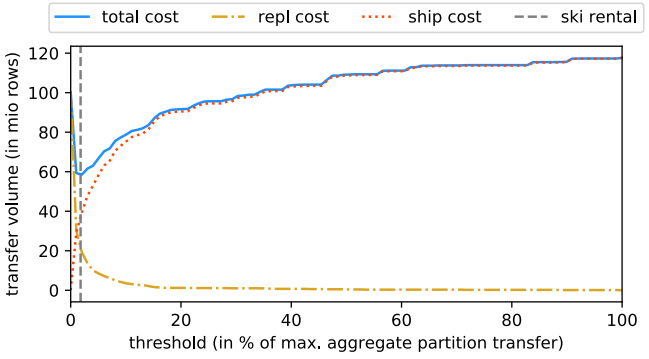
for different partition sizes. When looking at the accesses per partition over the duration of the trace we noticed some periodic access which resulted in repeating daily (1 AM) high volume access. After closer investigation, we concluded that these are likely to be the result of daily maintenance jobs and eliminated them by replacing them with the accesses from the previous hour. We refer to the cleaned traces as "Trace 1+" and "Trace 2+" respectively. Figure 3a and Figure 3b show the resulting access patterns as heatmaps (using logarithmic scale) per partition and hour. Frequent accesses to the same partition result in light color entries while low frequent ones result in dark color entries. Notice, that most entries are dark. However, some rows are much lighter than others. These are partitions that are frequently accessed (heavily used) and, thus, should be replicated. Partitions that are rarely accessed should not be replicated. Rather, the results should be computed at the back-end and shipped to the front-end.

Given that the distribution of row accesses is skewed (see Figure 2a), we expect that the distribution of data transfer over partitions is skewed as well. This is indeed the case, see Figure 4. Note, we normalized the x-axis with respect to the partition size. For example, for Trace 1+ about 60% of all partitions have data transfers less than the partition size, i.e., 100k rows. For these, shipping the query results is the "right" strategy while 40% have a data transfer of more than the partition size. For those, replicating the partitions is the "right" strategy.

To highlight that this does not only apply to the full trace but also for any sub-sequence of the query trace, we also include the distribution for the first half of the two traces (labeled Trace X+ (half)). All traces show a significant skew. We also see a distribution shift, in the sense that the skew decreases slightly for Trace 2+ and a bit more for Trace 1+ as the trace progresses. Possible explanations are that these are traces from an ERP system and not an IoT application and, thus, still may contain some regular access patterns.



(a) Trace 1+.

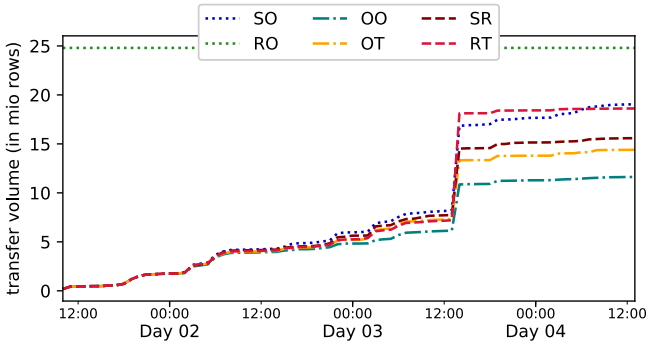


(b) Trace 2+.

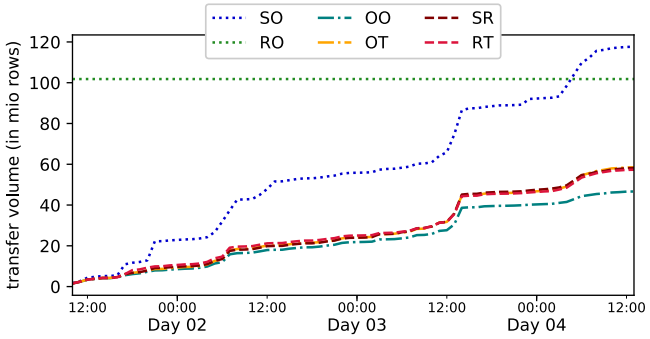
Figure 6: Transfer volume [in rows] for the threshold strategy for all possible thresholds (normalized by maximum aggregate partition transfer volume). In addition, we highlight the threshold of the SR strategy by a dashed line. Note, a threshold of 0 corresponds to RO while a threshold of 100 corresponds to SO.

5 EVALUATION

Next, we use the traces to do a *what-if* evaluation of the proposed strategies for our scenario with one back-end (incl. data store) and one front-end. For every strategy, we compute the transfer volume that would have been generated between



(a) Trace 1+.



(b) Trace 2+.

Figure 7: Transfer volume [in rows] over time for all strategies.

the front-end and the back-end, if this strategy had been used. For each trace and each strategy, we compute the total cost for the whole trace duration. Recall, Table 3 summarizes the strategies.

To start, we consider static strategies, i.e., those that use a fixed parameter. These include the naïve strategies ship-only (SO) and replicate-only (RO) as well as the ski-rental (SR) and the optimal threshold strategy (OT). Figure 5 shows the results normalized by the cost of the optimal offline strategy (OO). A lower bar corresponds to a smaller transfer volume. Note, all strategies are within a factor of 2.5 of optimal. However, the strategies that combine shipping and replication–SR (ski-rental) and the OT (optimal threshold) strategy–perform much better for both traces. Their overhead vs. the optimal offline algorithm is less than 34%. Indeed, the SR strategy already saves 22% (Trace 1+) and 100% (Trace 2+) compared to SO and saves 59% (Trace 1+) and 75% (Trace 2+) compared to RO. This confirms our intuition regarding exploring ski rental-based strategies. OT further improves upon the SR strategy about 10% (Trace 1+) and 0.1% (Trace 2+).

To further explore threshold strategies, we next look at all possible threshold choices in Figure 6[1]. Here, the x-axis is the threshold normalized by the maximum aggregate partition transfer volume for each trace, as depicted in Figure 4. The transfer volume ("total-cost") is the sum of the cost of shipping the query results ("ship-cost") and the cost of replication ("repl-cost"). Note, as the threshold increases the shpping cost increases monotonically while the replication cost decreases monotonically. However, since they are not convex (concave) their sum, the replication cost can have multiple local minima. This plot again confirms our intuition that combining replication and shipping is beneficial (local minima exist) and that the naïve strategies RO or SO are sub-optimal.

Next, we evaluate how the performance of the strategies changes as the trace length increases, see Figure 7. For both traces, the performance of strategies except the naïve ones (RO and SO) are very similar for the first half of the traces. Indeed, up to this point in time, the performance penalty of not knowing the future (compared to the optimum offline) is less than 15%. During the second half of the trace, the access patterns change which cause the RT strategy to send 60% (Trace 1+) and 30% (Trace 2+) more than the optimum. We suspect that the difference is caused by the distribution shift, as described in Section 4.

We propose the RT strategy to improve the OT/SR strategies. However, RT assumes that the access distribution can be approximated from the past. If a distribution shift occurs, this may no longer be the case. Thus, given the shift, it is not surprising that RT does not perform as well as OT and may even be worse than SR. However, we believe that better ways of estimating the distribution (e.g., by using a limited time window) and bounding the threshold should yield better performance. We are currently in the process of further evaluating such alternative strategies.

6 RELATED WORK

In this paper, we discuss the adaptive replication of a partitioned data store for a dynamic workload. This work bears similarities with the file allocation problem (FAP) and the database allocation problem (DAP).

In the file allocation problem, a file and its copies must be allocated in a network of computers with the goal of either minimizing the cost or maximizing the performance for a workload of file reads and writes. This problem has been shown to be NP-complete by Eswaren [18]. Solutions of the FAP can be classified by their assumption of the workload. Most of the earlier work on the FAP assumed a static workload [17] or a dynamic workload, which is known in advance [21]. The first solution for a dynamic workload, which

[1]RO/SO correspond to using a threshold of zero/infinity resp.

is unknown in advance, was presented by Wolfson et al. [34] in 1992. The problem was treated from the perspective of competitive analysis in [5]. Bartal et al. split the problem into separate allocation (placement of a fixed set of file copies), migration (movement of existing file copies) and replication (addition and deletion of additional file copies) problems [5]. Most research on the FAP assumed that each file could be allocated, migrated and replicated independently. In our work, we instead use the correlation between access of different partitions to inform the replication decision.

The database allocation problem (DAP) defined the same problem for database partitions (or "table fragments"). Different to files, dependencies between database partitions have to be considered due to integrity and consistency constraints and for performance reasons (e.g., joins) [3, 30]. Therefore, solutions to the DAP often include mechanisms to re-partition the data [23] or to estimate the template of future queries [22]. In our work, we assume that partitions function as the smallest unit in the storage organization that can be replicated without incurring a prohibitive overhead. Further, we believe that partitions, which are co-accessed, have similar access statistics and are therefore replicated soon after each other.

Our solution is based on earlier work on the ski rental problem [19, 20, 25–27], i.e., the problem of deciding at what point in time one should switch from renting to buying a ski-set. This and similar work on the FAP [4, 7, 10] are based on competitive analysis [33], a worst-case analysis for online algorithms. We have transferred the problem from independent objects, ski-sets, to partitions of a data store and from the worst-case to an average-case analysis.

To our knowledge, no commercial database includes an adaptive mechanism for data replication today. Even though modern commercial databases manage far more data than their predecessors, they use replication mostly to guarantee availability rather than improving performance and mostly rely on manual or static replication schemes. Often, database administrators must either manually specify the replication factor [16] or trigger the creation of new replicas [1, 15]. One exception is BigTable [11], which leaves the decision to the application. The strategies that we have proposed in this paper can help to build a self-tuning database [12, 13, 28] that uses replication to improve its performance. We see a strong need for such databases for future Internet of Things deployments.

7 CONCLUSION

In large-scale IoT deployments, the data is produced at rates that require storage and processing in distributed data stores. In this work, we look at data replication as a mechanism to minimize the transferred data volume in these distributed data stores. We study the cost trade-off between shipping query results versus replicating data partitions. To this end, we introduce two online replication strategies that decide when to replicate partitions. Both strategies use a threshold similar to the solution of the classical ski rental problem. One strategy uses a static threshold, i.e., a single threshold over time, and the second a dynamic threshold, i.e., a varying threshold over time. By applying our proposed strategies to two real-life datasets we show that they can yield significant transfer cost reduction compared to the baseline strategies (exclusively relying on shipping or replication). The static strategy resulted in a reduction of transfer cost between 22% to 100%.

As part of our future agenda, we plan to evaluate our strategies over a more extensive parameter space (different partition sizes, time granularities, etc.). We are also interested in assessing the performance of our strategies when the characteristics of the access patterns change. Finally, we are planning to investigate the integration of replication with caching and the inclusion of additional performance metrics such as response time.

ACKNOWLEDGMENTS

We want to thank Hannes Rauhe, Christian Krause, and Daniel Johannsen from the SAP team in Potsdam for their feedback and Martin Boissier from the Hasso Plattner Institute for giving us access to the dataset used in this analysis. Georgios Smaragdakis and the dissemination efforts of this work were supported in part by the European Research Council (ERC) grant ResolutioNet (ERC-StG-679158).

REFERENCES

[1] Hazelcast IMDG Reference Manual. published: 2019-04-25, accessed: 2019-05-05.

[2] Susanne Albers. Online Algorithms: A Survey. *Mathematical Programming*, 97(1-2):3–26, 2003.

[3] Peter MG Apers. Data Allocation in Distributed Database Systems. *ACM Transactions on Database Systems (TODS)*, 13(3):263–304, 1988.

[4] Baruch Awerbuchy, Yair Bartalz, and Amos Fiatx. Competitive Distributed File Allocation. *Theory of Computing*, pages 164–174, 1993.

[5] Yair Bartal, Amos Fiat, and Yuval Rabani. Competitive Algorithms for Distributed Data Management. *Journal of Computer and System Sciences*, 51(3):341–358, 1995.

[6] Brian Baumley. Deutsche Telekom Completes World's First Public Mobile Edge Network Powered By MobiledgeX Edge-Cloud R1.0. published: 2019-02-19, accessed: 2019-04-17.

[7] David L. Black and Daniel D. Sleator. *Competitive Algorithms for Replication and Migration Problems.* Carnegie-Mellon University. Department of Computer Science, 1989.

[8] Martin Boissier, Carsten Alexander Meyer, Timo Djürken, Jan Lindemann, Kathrin Mao, Pascal Reinhardt, Tim Specht, Tim Zimmermann, and Matthias Uflacker. Analyzing Data Relevance and Access Patterns of Live Production Database Systems. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 2473–2475, 2016.

[9] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012. 3.

[10] Kaushik Chakrabarti, Minos Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Approximate query processing using wavelets. *The VLDB Journal—The International Journal on Very Large Data Bases*, 10(2-3):199–223, 2001.

[11] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A Distributed Storage System for Structured Data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[12] Surajit Chaudhuri and Vivek Narasayya. Self-Tuning Database Systems: A Decade of Progress. In *Proceedings of the 33rd international conference on Very large data bases*, pages 3–14. VLDB Endowment, 2007.

[13] Surajit Chaudhuri and Vivek R. Narasayya. An Efficient, Cost-Driven Index Selection Tool for Microsoft SQL Server. In *VLDB*, volume 97, pages 146–155, 1997.

[14] Louis Columbus. 2018 Roundup Of Internet Of Things Forecasts And Market Estimates. published: 2018-12-13, accessed: 2019-05-04.

[15] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. Spanner: Google's Globally Distributed Database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):8, 2013.

[16] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's Highly Available Key-Value Store. In *ACM SIGOPS operating systems review*, volume 41, pages 205–220, 2007.

[17] Lawrence W. Dowdy and Derrell V. Foster. Comparative Models of the File Assignment Problem. *ACM Computing Surveys (CSUR)*, 14(2):287–313, 1982.

[18] Kapali P. Eswaran. Placement of Records in a File and Fileallocation in a Computer Network. In *Proc. IFIPS Conference*, pages 304–307, 1974.

[19] Amos Fiat. Online Algorithms: The State of the Art (Lecture Notes in Computer Science). 1998.

[20] Hiroshi Fujiwara and Kazuo Iwama. Average-Case Competitive Analyses for Ski-Rental Problems. *Algorithmica*, 42(1):95–107, 2005.

[21] Bezalel Gavish and Olivia R. Liu Sheng. Dynamic File Migration in Distributed Computer Systems. *Communications of the ACM*, 33(2):177–189, 1990.

[22] Tobias Groothuyse, Swaminathan Sivasubramanian, and Guillaume Pierre. Globetp: Template-Based Database Replication for Scalable Web Applications. In *Proceedings of the 16th international conference on World Wide Web*, pages 301–310, 2007.

[23] Jon Olav Hauglid, Norvald H Ryeng, and Kjetil Nørvåg. DYFRAM: Dynamic Fragmentation and Replica Management in Distributed Database Systems. *Distributed and Parallel Databases*, 28(2-3):157–185, 2010.

[24] Windsor W. Hsu, Alan Jay Smith, and Honesty C. Young. I/O Reference Behavior of Production Database Workloads and the TPC Benchmarks—an Analysis at the Logical Level. *ACM Trans. Database Syst.*, 26(1):96–143, March 2001.

[25] Anna R Karlin, Kai Li, Mark S Manasse, and Susan Owicki. Empirical Studies of Competitve Spinning for a Shared-Memory Multiprocessor. In *ACM SIGOPS Operating Systems Review*, volume 25, pages 41–55, 1991.

[26] Anna R Karlin, Mark S Manasse, Larry Rudolph, and Daniel D Sleator. Competitive Snoopy Caching. *Algorithmica*, 3(1-4):79–119, 1988.

[27] Ali Khanafer, Murali Kodialam, and Krishna PN Puttaswamy. The Constrained Ski-Rental Problem and its Application to Online Cloud Cost Optimization. In *Proceedings of IEEE INFOCOM*, pages 1492–1500, 2013.

[28] Tim Kraska, Mohammad Alizadeh, Alex Beutel, Ed H Chi, Jialin Ding, Ani Kristo, Guillaume Leclerc, Samuel Madden, Hongzi Mao, and Vikram Nathan. SageDB: A Learned Database System. In *CIDR*, 2019.

[29] David Mc Hugh. Volkswagen to network factories in the cloud with Amazon. published: 2019-03-27, accessed: 2019-04-17.

[30] M Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Springer Science & Business Media, 2011.

[31] Larry Peterson, Tom Anderson, Sachin Katti, Nick McKeown, Guru Parulkar, Jennifer Rexford, Mahadev Satyanarayanan, Oguz Sunay, and Amin Vahdat. Democratizing the Network Edge. *ACM SIGCOMM Computer Communication Review*, 49(2), May 2019.

[32] Niklas Semmler, Georgios Smaragdakis, and Anja Feldmann. Distributed Mega-Datasets: The Need for Novel Computing Primitives. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019.

[33] Daniel D. Sleator and Robert E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM*, 28(2):202–208, 1985.

[34] Ouri Wolfson and Sushil Jajodia. Distributed Algorithms for Dynamic Replication of Data. In *Proceedings of the eleventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 149–163. ACM, 1992.