Workload-Driven Adaptive Data Partitioning and Distribution – The Cumulus Approach

Ilir Fetai Damian Murezzan Heiko Schuldt
Department of Mathematics and Computer Science
University of Basel, Switzerland
firstname.lastname@unibas.ch

***Abstract*—Cloud environments usually feature several geographically distributed data centers. In order to increase the scalability of applications, many Cloud providers partition data and distribute these partitions across data centers to balance the load. However, if the partitions are not carefully chosen, it might lead to distributed transactions. This is particularly expensive when applications require strong consistency guarantees. The additional synchronization needed for atomic commitment would strongly impact transaction throughput and could even completely undo the gain that can be achieved by load balancing. Hence, it is beneficial to avoid distributed transactions as much as possible by partitioning the data in such a way that transactions can be executed locally. As access patterns of characteristic transaction workloads may change over time, the partitioning also needs to be dynamically updated. In this paper we introduce *Cumulus*, an adaptive data partitioning approach which is able to identify characteristic access patterns of transaction mixes, to determine data partitions based on these patterns, and to dynamically re-partition data if the access patterns change. In the evaluation based on the TPC-C benchmark, we show that *Cumulus* significantly increases the overall system performance in an OLTP setting compared to static data partitioning approaches. Moreover, we show that *Cumulus* is able to adapt to workload shifts at runtime by generating partitions that match the actual workload and to re-configure the system on the fly.**

I. Introduction

Cloud providers usually operate several data centers across which data can be replicated and/or partitioned. Data replication can increase the performance of transactions since the load can be distributed among the available replica sites. But this only holds for read transactions. Update transactions in the presence of replication, however, require additional coordination between sites [1]–[3], if applications demand strong consistency guarantees. This is the case for various types of OLTP applications, such as financial applications, that cannot sacrifice consistency. Thus, they need guarantees like One-Copy-Serializability (1SR) [4] which, in turn, require expensive distributed commit protocols like 2PC or Paxos. These protocols are expensive since they require several rounds of network communication [5]–[9]. This additional overhead is considerable if geo-replication is used [10]. Moreover, it has also a negative impact on the system throughput due to the increased duration in which locks have to be held [11]. Protocols like 2PC are potentially blocking, thus further impact system performance. Non-blocking distributed commit protocols, such as Paxos [5], [12], can be used at even higher costs than 2PC. At the same time, despite of the high latency due to the overhead of transaction coordination [10], [13], typical OLTP applications need highly scalable databases [14].

Shared-nothing architectures [15], in turn, that make use of data partitioning (a.k.a. sharding) are able to manage data in such a way that distributed transactions are avoided – or at least that their number is minimized. This eliminates or minimizes the necessity of expensive distributed commit protocols on one hand, and on the other hand, it avoids performance bottlenecks by distributing the load between sites [15], [16]. However, distributed transactions are not the only source of overhead in a distributed system. Hotspots, i.e., very popular data, may lead to certain partitions or sites hosting these partitions being overload [3]. It is thus crucial to jointly avoid both distributed transactions and hotspots.

Recently, the term *NewSQL* has been coined. It is used for databases that provide the scalability of NoSQL systems (that usually only feature weak consistency guarantees), yet without sacrificing consistency [14]. For this, most NewSQL systems use a distributed shared-nothing database architecture [14], [17], [18].

Most current Cloud providers replicate data across different data-centers [19] and thus rather ignore the important quality-of-service requirement of providing low latency [20] for OLTP applications in the presence of strong consistency. While application servers and web servers can be easily scaled out by adding additional machines, the data management layer becomes the bottleneck [21]. Hence, data partitioning, rather than replication, is essential to provide scalability at the data management layer beyond a single site [20].

In this paper we introduce *Cumulus*, an adaptive workload-driven partitioning protocol tailored to applications that need strong consistency guarantees. *Cumulus* is able to partition the data in a shared-nothing architecture so that the number of distributed transactions is at least minimized or that they are even completely avoided.

The contribution of this paper is threefold. First, we propose the *Cumulus* partitioning algorithm that distinguishes between significant and insignificant (noisy) access patterns of transaction mixes. Partitions are created with the objective of minimizing the number of distributed transactions and, at the same time, of effectively distributing the load across sites. Second, we show how the *Cumulus* partitioning algorithm can be adaptively and dynamically used for re-repartitioning the data at runtime. The adaptive behavior is based on a cost model that trades the partitioning cost for the expected gain resulting from a reduced number of distributed transactions with the goal of avoiding unnecessary and expensive re-partitioning. Third, we present the implementation of *Cumulus* and an evaluation of the system based on the TPC-C benchmark. The evaluation results show that *Cumulus* significantly outperforms static data partitioning approaches and approaches that do not restrict data partitioning to only significant access patterns. Thus, it increases the overall transaction throughput in an OLTP setting. Moreover, *Cumulus* can deal with dynamically changing access patterns and adaptively re-partition the data without stopping the entire system.

The remainder of the paper is structured as follows. Section II provides an overview of different data partitioning properties. Section III introduces *Cumulus* and Section IV presents its implementation. The results of the evaluation of *Cumulus* in an OLTP setting are reported in Section V. Section VI surveys related work and Section VII concludes.

II. Data Partitioning

In what follows, we briefly characterize different requirements and approaches for data partitioning.

Distributed Transactions and Load Distribution: Distributed transactions are costly [2] due to the necessary commit coordination. Thus, a partitioning protocol should minimize the number of –or in best case completely avoid– distributed transactions. There are three basic approaches to partition a database: i.) horizontal, ii.) vertical, and iii.) hybrid [16]. Horizontal partitioning splits data across sites based on key ranges. Vertical partitioning splits the data along the line of attributes, and hybrid partitioning is a combination of horizontal and vertical partitioning.

In addition to distributed transactions, a data partitioning protocol should also avoid so-called *hotspots* which are objects or partitions that are very popular. This can be done by creating partitions in such a way that the load is balanced between partitions. Graph-based algorithms [2], for instance, can be used to target both goals, i.e., the minimization of distributed transactions and load distribution.

Rigid vs. Elastic Partitioning: Rigid partitioning protocols are not able to provision or de-provision sites as the workload changes. In contrast, elastic protocols are able to deploy new or un-deploy existing sites based on the current workload [22]. Newly deployed sites can be used to create replicas for overloaded partitions and use them for distributing the load, or a re-partitioning can be initiated by incorporating new sites into the distribution process of partitions to sites. Both approaches can be used to address the hotspot challenge, to avoid that a partition is more frequently accessed by transactions than the others. However, by replicating hotspot partitions and distributing the load between sites, expensive distributed transactions are introduced. Even though only the hotspot data objects are involved in these distributed transactions, the performance of this approach may still degrade to the performance of a fully replicated system due to the popularity of the hotspots.

Static vs. Adaptive Partitioning: Static protocols are not able to react at runtime to changes in the access patterns. The system must be taken off-line in order to apply a re-partitioning, and is then again put in operation. However, many Cloud applications demand always-on guarantees with low latency [19]. In addition, such applications typically feature different types of workload skew [3]. This requires adaptive, low-cost partitioning protocols.

Manual vs. Automatic Partitioning: Manual protocols require the involvement of a (human) expert, while automatic approaches are able to re-partition without any expert involvement. Hybrid protocols can propose a partition schema that can be further refined by an expert, or refine a schema proposed by an expert.

Routing Transparency vs. Manual Site Selection: Compared to a fully replicated system, the location of data in a partitioned system is mandatory information needed to route transactions to those sites that contain the required data. Protocols providing routing transparency decouple the client from the data location and give the system the flexibility to re-partition data without affecting client applications.

III. Cumulus

Cumulus is an adaptive data partitioning system that uses a partitioning protocol able to reduce –or if possible completely avoid– both distributed transactions and hotspots.

A. Partitioning Workflow

Cumulus targets three main aspects that considerably impact the quality of the partitioning, the generated overhead, and the correctness of the system behavior. First, *Cumulus* incorporates a workload analysis approach that aims at anticipating future access patterns for the generation of a partition schema to best match the expected workload and thus reduce or completely avoid expensive distributed transactions. Second, *Cumulus* uses a cost model which ensures that new partitions are only applied if they actually lead to a gain. Otherwise, in long-term, the overhead of the re-configuration activities may overweight the actual benefit. And third, *Cumulus*' re-configuration approach exploits two-phase locking (2PL) and two-phase commit (2PC) to ensure correctness even in case of failures.

The partitioning workflow of *Cumulus* is depicted in Figure 1. In step 1.), based on a *triggering strategy*, *Cumulus*

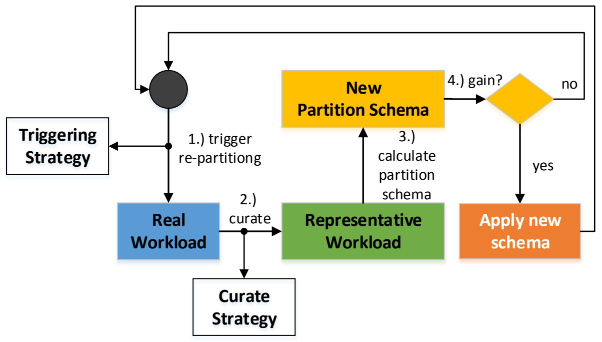


Figure 1: *Cumulus* Workflow

initiates the calculation of a partition schema. In reaction to that, in step 2.), the workload of each site, the so-called *real workload*, is collected. The workload is defined by the objects accessed by transactions. In what follows, we will refer to these objects as *transaction access pattern*. We distinguish between the *local workload*, collected by individual sites, and the *global workload*, which is a collection (aggregation) of all local workloads. In step 3.), the global real workload is curated using a *curate strategy*, leading to the *representative workload*. The goal of step 3.) is to increase the probability that the generated schema based on collected workloads from the past best anticipates the expected workload in the future. Otherwise, a partition schema may be generated that is tailored to a workload that is unlikely to occur. In step 4.), *Cumulus* decides based on a *cost model* if there is enough gain in applying the new schema. If so, the schema is applied and the system is re-configured. *Cumulus* implements an on-the-fly and on-demand re-configuration approach, which is driven by user transactions (i.e., only objects that need to be accessed will be re-distributed on the fly) such that the system remains available during the re-configuration.

B. Workload Collection and Analysis

The goal of the workload collection and analysis step is to collect the workload from each site and perform an analysis in an adaptive manner in order to determine the expected application workload and generate a partition schema that is tailored to that workload. The *Cumulus* approach to workload analysis is based on the idea of separating the expected access patterns that will occur with high-frequency in the future –denoted as *representative patterns*– and access patterns that will occur with low frequency –denoted as *noisy patterns*. The representative patterns are considered as a basis for generating an optimal partition schema, as considering all patterns leads to following drawbacks. First, each access pattern from the workload represents a constraint to the problem of determining an optimal partition schema. The more constraints are available, the more difficult it is to satisfy all of them leading to a sub-optimal schema. Second, optimizing for in-frequent transactions does generate only a low or no benefit at all for future transactions.

The workload collection and analysis approach of *Cumulus* works as follows: The workload of the sites is periodically collected based on a configurable timeout parameter, denoted as $wTimeout$. Once the collected workload has reached a specific size ($wSize$), the workload analysis step is triggered. The analysis step results in a prediction of the expected workload for the future (until the next analysis step), that is used for calculating a new partition schema and for deciding, based on the cost model, if this new schema should actually be applied. In *Cumulus*, the prediction of the expected workload is based on the idea of predicting the frequency of access patterns, using the exponential moving average (EMA) [23]. Let $t.LO$ denote a certain access pattern consisting of actions and the objects actions act on. Further, let $f(t.LO)$ denote the frequency of $t.LO$. The EMA-based frequency prediction is calculated as follows:

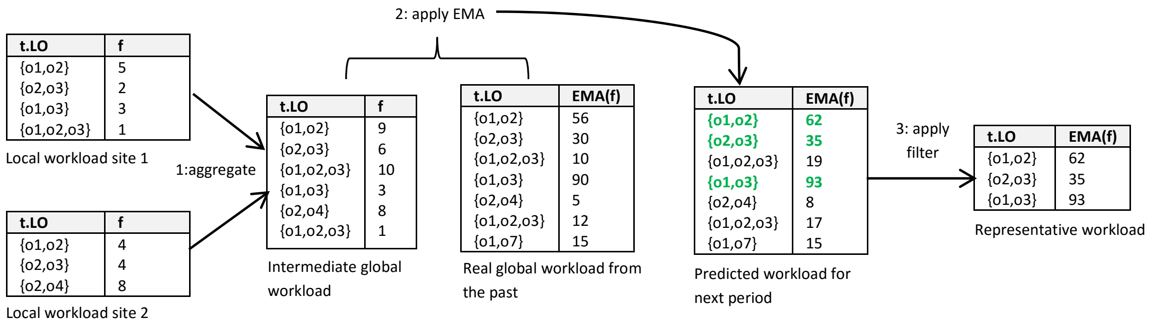
$$\begin{aligned} EMA(f(t.LO_{p+1})) = \\ \alpha \cdot f(t.LO_p) + (1-\alpha) \cdot EMA(f(t.LO_p)) \end{aligned} \tag{1}$$

where α is a smoothing factor with $0 < \alpha < 1$. Each time a prediction is made, a new prediction interval is started and the old one is closed. In Eq. (1), p denotes the recently closed interval and $p+1$ the next interval for which the frequencies are predicted. In EMA, predictions are recursively based on the past; hence, it requires only the predicted frequency for the last period. A prediction for a period p thus includes and is based on the prediction of all previous periods $p-1$, $p-2, \ldots$ (see Figure 2). Experiments show that EMA is able to accurately predict certain workload types [24].

The choice of the smoothing factor α is essential for accurately predicting the frequency of access patterns. In order to determine the most suitable value for α, we apply Eq. (1) using $\alpha \in \{0.1, 0.2, \ldots, 0.9\}$ and then use that value of α that has the overall lowest mean absolute deviation (MAD). Note that we use an incremental approach for calculating the MAD. Other approaches, such as expectation maximization and maximum likelihood [25] for determining the best α are considered part of a future work.

The smoothing property of EMA is crucial, as high frequency access patterns that were added as part of previously triggered re-partitioning activities would remain indefinitely in the workload if no smoothing was applied. Over time, they might superimpose and dominate the current and expected access patterns and would behave similar to noisy transactions, preventing the system to adapt to shifting access patterns. Clearly, a naïve approach would be to construct the workload from scratch each time a partitioning is triggered. However, this memory-less approach would disallow the system to recognize long-term access patterns, and would lead to a highly unstable system that will steadily re-partition in reaction to short-term access pattern changes.

After the frequency prediction, a high-pass filter is applied to the predicted frequencies with the goal of removing in-



t.LO	f
{o1,o2}	5
{o2,o3}	2
{o1,o3}	3
{o1,o2,o3}	1

t.LO	f
{o1,o2}	4
{o2,o3}	4
{o2,o4}	8

t.LO	f
{o1,o2}	9
{o2,o3}	6
{o1,o2,o3}	10
{o1,o3}	3
{o2,o4}	8
{o1,o2,o3}	1

t.LO	EMA(f)
{o1,o2}	56
{o2,o3}	30
{o1,o2,o3}	10
{o1,o3}	90
{o2,o4}	5
{o1,o2,o3}	12
{o1,o7}	15

t.LO	EMA(f)
{o1,o2}	**62**
{o2,o3}	**35**
{o1,o2,o3}	19
{o1,o3}	**93**
{o2,o4}	8
{o1,o2,o3}	17
{o1,o7}	15

t.LO	EMA(f)
{o1,o2}	62
{o2,o3}	35
{o1,o3}	93

Figure 2: Workload Analysis in *Cumulus*

frequent (noisy) patterns from the workload. The filter only preserves access patterns that have a predicted frequency above a certain threshold (see Figure 4). The choice of the applied filter can alter the partitioning behavior significantly. The more aggressive the high pass filter is tuned, the more stable already established patterns will be. On the other hand, new patterns will need a significant frequency before they will be recognized. It is thus advised to tune the filter according to the expected access pattern, i.e., a high threshold for stable patterns and a lower threshold for volatile access patterns.

The resulting representative workload is maintained as a graph, similar to the approach presented in [2]. The nodes of the graph denote the data objects, and there is an edge from o_i to o_j if both objects are accessed by the same transaction (see Figure 3). Edge weights improve the partition step by giving a higher weight to re-occurring transaction edges. The problem of finding an optimal partition schema is thus equivalent to the problem of finding optimal graph partitions and this can be done by graph partitioning libraries such as Metis [26]. Since *Cumulus* takes into account only objects from the representative workload when determining the partitions, objects not available in the workload can be striped to the available sites based on a round-robin approach.

Scalability issues can arise by using graph partitioning libraries, such as Metis. The bigger the graph gets, the longer the partitioning will take. This can be fixed by setting a maximum number of transaction patterns which will be added to the graph. With such an approach, the time to partition should remain constant. However, reducing the number of transactions will also reduce the quality of the partition scheme. This will put an even increased emphasis of the curation step and will only work if a limited number of transactions is responsible for the majority of the load on the system.

C. Cost Model

Data partitioning comes with costs that are generated by the necessity of calculating the new partition schema, migrating the data, and updating the routing information. All this generates considerable overhead and if not done properly, might have an adverse effect on the system performance. However, the new schema generates a benefit to the system, which can be measured in terms of the reduction in distributed transactions. The benefit is considerably impacted by the quality of the generated partition schema, i.e., how well the schema matches the workload, and the lifetime of the schema (stability). By including a workload analysis based on exponential smoothing and high-pass filtering, *Cumulus* considerably improves both the quality and the stability of the schema. The cost of partitioning must be taken into account when applying a partition schema. Let $pCost(nSc, oSc)$ denote the cost of partitioning, i.e., switching from an old schema oSc to a new schema nSc, and $wCost(sc, w)$ denotes the cost of executing the workload w using a certain partition schema sc. In *Cumulus* each object o to be migrated has certain cost $mCost(o)$, and each distributed transaction t generates a cost $dCost(t)$. Partitioning costs $pCost(nSc, oSc)$ can be calculated as follows:

$$pCost(nSc, oSc) = \sum_{o \in OtM} mCost(o) \tag{2}$$

where OtM is the set of objects to migrate. The cost of executing a certain workload w with the partition schema sc is calculated as follows:

$$wCost(sc, w) = \sum_{t \in Txns} dCost(t) \tag{3}$$

The cost of a transaction $dCost(t)$ increases in non-linear manner with the distribution degree $dDegree(t)$ of the transaction. The cost is maximal ($costOfDistrTxn$) if a transaction has a maximal distribution degree, which is the case if a transaction accesses all available partitions:

$$dCost(t) = \left(\frac{dDegree(t)}{maxDDegree}\right)^2 \cdot costOfDistrTxn$$
$$dDegree(t) = nrOfAccessedPartitions(t) - 1$$
$$maxDDegree = totalNrOfPartitions - 1$$

The benefit of applying a nSc can now be calculated by:

$$b(nSc, oSc, w) = wCost(oSc, w) - wCost(nSc, w) \tag{4}$$

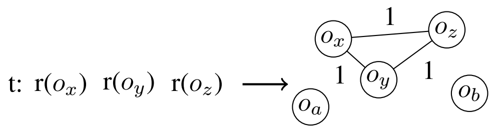


Figure 3: Workload graph with data objects o_x,o_y,o_z,o_a,o_b

Eq. (4) calculates the benefit b as the difference of executing a certain workload (the predicted workload for the next period) using the old and the new schema. There is a benefit if $b > 0$. A gain g is achieved if the benefit is higher than the partitioning costs:

$$g(nSc, oSc, w) = b(nSc, oSc, w) - pCost(nSc, w) \quad (5)$$

The partition schema nSc is applied if the gain g is above a certain threshold th, i.e., if $g(nSc, oSc, w) > th$. In here, the choice of the threshold parameter th is crucial. A high threshold increases the life-cycle of a partition schema and may lead to a high loss as there is a high tolerance to distributed transactions. A low threshold leads to frequent schema applications and might lead to partitioning costs that outweigh the gain. In future work, we plan to incorporate a machine learning-based approach into *Cumulus* to determine the optimal value of th.

D. Re-Configuration

At the end of the partitioning workflow a new partition schema is proposed (c.f. Figure 1). If the new schema is beneficial ($g > th$ in Eq. (5)), it is applied. As a consequence, the system must be re-configured to match the new schema. This process is known as live migration [27], [28]. The re-configuration consists of the migration of data objects to the new locations (sites) and the update of the routing configuration. As the entire partitioning is done dynamically at runtime, the re-configuration process will interfere with the execution of user transactions. Additionally, the re-configuration involves communication between sites and is thus a distributed activity. Two aspects need to be treated with care in order to guarantee proper behavior. First, the re-configuration should be synchronized with the execution of user transactions. Second, the correctness (consistency) of the distributed re-configuration should be guaranteed even in presence of failures.

Cumulus implements an *on-the-fly and on-demand re-configuration* approach, which in contrast to a stop and copy approach [28], has a considerable advantage w.r.t. the system availability. Although it incurs an overhead for single transactions, the on-the-fly and on-demand approach avoids situations in which high arrival rate of transactions fill-in the queues and lead to an explosion in response time and possible system instability [29]. In what follows, we will describe the re-configuration process of *Cumulus*.

Once a new partition schema has been generated, it is sent to all sites. Based on the received schema, the sites locally calculate a *change-set* consisting of a set of triples

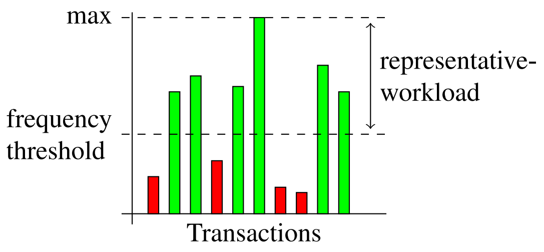


Figure 4: Frequency-based Workload Analysis

$\langle k_{o_i}, currLoc, newLoc\rangle$ which encompasses the id k_{o_i} of the object to be migrated, $currLoc$ denotes the current location, and $newLoc$ the site that will be responsible after the re-configuration. The re-configuration process is initiated by *Cumulus*' *PartitionManager*. The process is coordinated by the 2PC protocol with the *PartitionManager* acting as a coordinator. The prefix `rc-` is used to describe the 2PC-messages used in the context of the re-configuration process.

1) A `rc-prepare` message containing the new schema is sent to all sites. The message denotes at the same time an incentive for re-partitioning.
2) The sites that receive the `rc-prepare` message locally set an `rc-lock` and calculate the change-set based on the received schema. Once the change-set has been created and the execution of active transactions has finished, the sites reply with an `rc-prepare-ack`. Incoming transactions will be added to a wait queue after the `rc-lock` has been set. Hence, no execution is started once the lock is set. This avoids interferences with running user transactions.
3) Once the coordinator has received an `rc-prepare-ack` from all sites, it returns a `rc-commit` message.
4) In reaction to the `rc-commit` message, the sites release their `rc-lock`, return a `rc-commit-ack`, and resume transaction execution.

The re-configuration is driven by user transactions. Each transaction accessing not yet migrated data objects will migrate those objects by transforming its write operations to distributed write operations and, in case the object to be migrated is accessed only in read mode, by appending a write operation for each read operation. Again, the distributed writes are coordinated by 2PC.

As objects are migrated only when accessed by transactions (on demand), a mechanism has to be in place that adds non-migrated objects to subsequent repartitioning events. For this, the old change-set has to be merged with the new change-set as follows: If an object is still present in the old change-set, the current location of the object will be the location specified there and not the one in the new partition schema. Thus, the change set takes over the current location, but updates the target location with the new set of locations from the new partition schema. This allows to postpone the migration of an object even across different repartition events. The overhead introduced by the migration

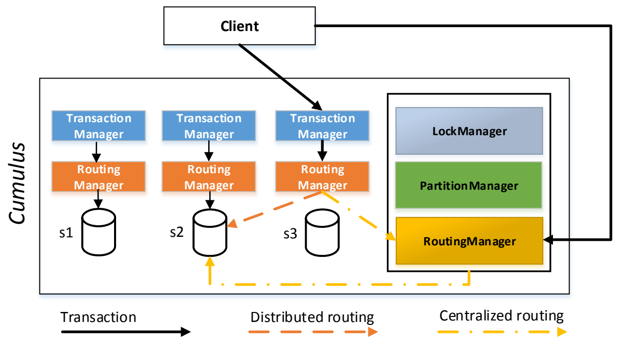


Figure 5: *Cumulus* System Architecture

step in the on-the-fly approach results from distributed write transactions across all sites which either have the up-to-date data or need to update their data. Thus, all transactions which migrate data will be distributed update transactions, regardless of their true nature. As the migration must only take place once, this overhead will decrease over time.

E. Handling of Inserts and Deletes

Insert and delete operations need a special treatment by a partitioning protocol. In *Cumulus*, a delete transaction is split in sub-transactions containing a batch of deletes for each site containing objects to be deleted. The atomicity guarantees are preserved for a transaction by using 2PC for all sub-transactions. If a delete transaction commits, the routing information needs to be updated.

Insert operations require special treatment as no access patterns are known at insert time, yet the goals of partitioning, namely the collocation of objects and load distribution, should nevertheless apply to newly inserted objects. *Cumulus* does not immediately enforce the collocation goal, but is mainly focused on evenly distributing inserted objects across sites. It postpones object collocation to the next partitioning event. Each site can locally decide on where to next insert objects, by choosing a site from all available sites, all with the same probability. In worst-case, an additional sub-transaction might be necessary, when the chosen site is not accessed by the current transaction.

IV. Cumulus Implementation

The architecture of *Cumulus* is depicted in Figure 5 and consists of the following modules: *TransactionManager*, *PartitionManager*, *LockManager*, and *RoutingManager*, which are described in detail in the following.

A. TransactionManager

The *TransactionManager* is the module responsible for the execution of transactions and is available at each site. When a *TransactionManager* receives a transaction for execution, it will first acquire all locks at the *LockManager* and become the coordinator for that transaction. In *Cumulus*, both read and update transactions may be distributed, which is inherent to all partitioned systems. In case of a distributed read transaction, *Cumulus* will collocate all operations that can be served locally into a local sub-transaction at the coordinating site, the rest will be greedily distributed as sub-transactions to the sites at which the accessed objects are located. In here, greedy means that sub-transactions are generated with the maximum number of objects. For distributed update transactions, 2PC is used for atomic commitment. In order to preserve the ordering of operations inside the same transaction, all objects accessed by the transaction will be collected at the coordinating site, the transaction will be executed, and the generated results will be written back to the corresponding sites.

As described in Section III-D, 2PC is used not only for distributed user transactions, but also during the re-configuration process. The 2PC messages are exchanged, depending on the context, between the *PartitionManager* and the *TransactionManager*, or between the *TransactionManagers*. In summary, *TransactionManagers* are involved in the execution of 2PC at three different levels. First, during the distributed commit of user transactions. Second, during the initiation of the re-configuration process coordinated by the *PartitionManager*. And third, in the execution of re-configuration transactions.

B. PartitionManager

The *PartitionManager* is responsible for the generation of the partition schema and it triggers and coordinates the entire re-configuration process. Moreover, it feeds its sub-modules, i.e., the *WorkloadAnalyser*, the *PartitionEngine*, and the *CostAnalyser*, with necessary data.

A daemon, which is responsible for triggering a re-partitioning, will periodically (based on a configured time-out) collect the workload from all sites via the sites' *TransactionManagers*. The *triggering-strategy* is based on the size of the collected workload. If the size of the collected workload is above the specified threshold, then the workflow for calculating a partition schema will be initiated (see Figure 1) by the *WorkloadAnalyser* sub-module. After completion of this workflow, a new schema is proposed. Based on the cost model defined in Section III-C, the *CostAnalyser* sub-module will decide if the new schema is to be applied, and if so the re-configuration activities are initiated. The *PartitionEngine* sub-module is responsible for calculating a partition schema given a certain representative workload. The current *PartitionEngine* of *Cumulus* is based on a graph algorithm similar to [2]; however, other *PartitionEngines* can also be seamlessly integrated into *Cumulus*.

C. RoutingManager

Cumulus supports both a centralized and distributed routing architecture as depicted in Figure 5. It provides full routing transparency to clients by allowing them to connect to any site and managing all routing inside the system. The receiving site will forward a transaction to any of the sites that manage objects accessed by the transaction and that site will become the coordinator of the transaction. If clients send randomly transactions to sites, in worst-case one additional forwarding step is required and the transaction is a remote transaction. The probability of a transaction becoming a remote one is $1 - \frac{s_{rel}}{s}$, with s_{rel} denoting the set of sites containing any object accessed by the transaction. The distribution degree of transactions is not determined by the routing strategy, but by the quality of the partition schema. In case of the additional forward step, the optimal approach w.r.t. the number of objects transferred in the system is to send the transaction to the site containing the largest subset of objects accessed by the transactions. The complexity of such an approach is $\mathcal{O}(s \cdot m)$ with m being the transaction size. The optimal strategy would reduce the bandwidth consumption, as the number of objects transferred would be smaller compared to the naïve approach. If $s \gg m$ then the complexity is reduced to that of naïve routing. When sending transactions to the optimal replica, a tight coupling of server and client is necessary, as the partition scheme needs to be known by the client.

V. Evaluation

The objective of the evaluation of *Cumulus* is twofold. First, we show the impact of workload analysis (to determine the representative workload) on the partition schema quality and its impact on the reduction distributed transactions. For this, we compare *Cumulus* to a partitioning approach that does not consider workload analysis (to an approach that considers the entire workload for determining the partition schema). Second, we compare the *Cumulus*' re-configuration approach to a stop and copy approach, and we provide a comparison of *Cumulus* with a static partitioning approach.

A. Evaluation Set-Up

For the evaluation we have used the TPC-C data model [30], which is generated using the following TPC-C parameters: the number of districts is 10, the number of customers is 3'000, and the number of stock entries is 10'000.

The system under evaluation consists of a set of sites given by $nrOfSites$. A test client running on a dedicated machine generates the desired workload as specified by the following workload parameters: *w/r ratio*, *txnSize*, *nrOfAccessPatterns*, *noiseLevel*, and *nrOfWorkers*. Each worker, corresponding to a thread, creates a transaction, submits it for execution, and waits for its response before creating and submitting a new transaction. The evaluation parameters are summarized in Table I. All experiments are conducted in the AWS EC2 environment[1]. We have used $c1.medium$[2] as machine type deployed in the *eu-west*[3] region.

[1]http://aws.amazon.com/de/ec2/

[2]http://aws.amazon.com/de/ec2/instance-types/

[3]http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html

Parameter	Description
nrOfSites	Number of sites in the system
w/r ratio	The ratio of write/read transactions
nrOfAccessPatterns	The number of access-patterns
noiseLevel	The percentage of noisy transactions
nrWorkers	The number of worker threads that generate transactions

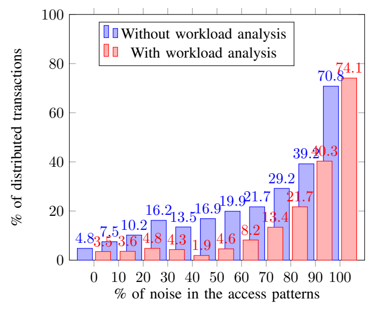
Table I: Evaluation-Setup Parameters

B. Workload Analysis and Data Partitioning

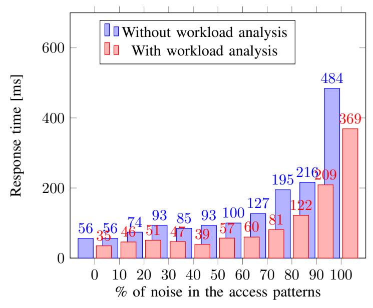
The goal of this experiment is to evaluate the impact of workload analysis and curation (see Section III-B) on the quality of the generated partition schema. As *Cumulus* uses a similar partitioning algorithm as Schism [2] and as Schism does not incorporate a workload analysis, the results of this experiment compare *Cumulus* with Schism by depicting the advantages of workload analysis.

The set-up is as follows: a pool of re-occurring access patterns and a pool of transactions introducing noise in the workload is generated for each of the runs, with a varying ratio of noise in the access patterns. Each run is initiated by a warm-up phase, followed by the actual evaluation. In Figures 6a and 6b, we have depicted the percentage of distributed transactions for varying ratios of noise in the access patterns, and we have compared the curated with the non-curated workload. The charts show that the frequency-based workload analysis of *Cumulus* considerably improves the quality of the generated partition schema which is reflected in the lower percentage of distributed transactions generated compared to the workload that has not been curated. However, if the ratio of noise in the workload is extremely high (especially in the case of 100% noise), then, as no characteristic access patterns exist, workload analysis cannot lead to any improvement.

Another important feature of the *Cumulus* workload analysis is that only the objects accessed within the representative access patterns are used to determine the partition schema – this is in contrast to similar approaches such as Schism that incorporate all existing data objects into the partition schema generation. This avoids hot spots in which all objects frequently accessed together are collocated at one single site. We have used a size-up test as defined in [31] to depict *Cumulus*' performance impact on the improved object (load) distribution to sites. In Figure 7, we have depicted the performance results of three different partition approaches. SUBSET corresponds to the *Cumulus* approach that considers only objects that occur in the representative access patterns to determine the schema, ALL uses all objects, and the third uses a hash-based control mechanism. As it can be seen, the *Cumulus* approach yields best performance compared to the other two approaches, mainly due to the better load balancing. The ALL approach yields a 0% distributed transaction rate, which is expected as it better collocates objects accessed together. This rate

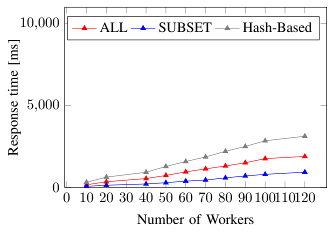


(a) Percentage of Distributed Transactions

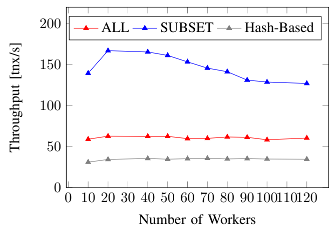


(b) Transaction Response Time

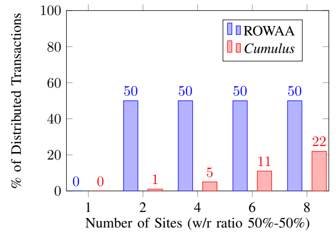
Figure 6: Impact of workload analysis on the quality of the generated partition schema



(a) Transaction Response Time



(b) Transaction Throughput



(c) Increasing number of sites

Figure 7: Performance of different partition approaches

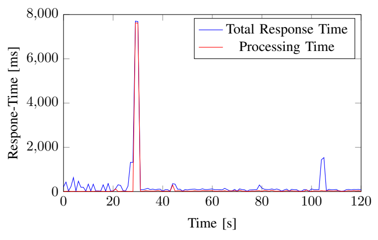
was between 4% and 10% for *Cumulus* and over 90% for the hash-based approach. Hence, the better performance of *Cumulus* (SUBSET) comes from the better load balancing capability compared to the ALL approach, especially from the avoidance of hot spots. This shows that an important criterion for the overall system performance is to find a good trade-off between load balancing and minimization of distributed transactions; concentrating only on one aspect does not automatically lead to better system performance.

As depicted in Figure 7c, collocation and load distribution are two competing goals. The more sites are added, the less collocation can be achieved in *Cumulus* and thus the number of distributed transactions increases with increasing number of sites. In a fully replicated system such as ROWAA [1], the number of sites does not have any effect on the number of distributed transactions which is only determined by the percentage of update transactions in the transaction mix.

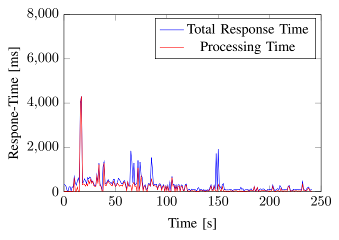
C. Adaptive Partitioning

Cumulus implements an on-the-fly and on-demand adaptive re-configuration approach that dynamically reacts to access pattern changes. We have conducted experiments to compare the runtime behavior of the *Cumulus* approach to the stop and copy approach [28]. For this experiment, we have used following parameters: $nrOfWorkers = 50$, $nrOfSites = 4$, and $w/r\ ratio = 50\%/50\%$. We have used one pool of access patterns. There is no warm-up phase as the goal is to show the adaptive behavior. *Cumulus* runs for a certain period of time, extracts the access patterns and then triggers a re-partitioning. In Figure 8a, we have depicted the response time behavior of transactions using the stop and copy approach, and in Figure 8b the response time using the on-the-fly approach. As it can be seen, the stop and copy approach generates a huge spike in the response time which then stabilizes once the re-configuration is completed. During re-configuration, all incoming transactions are added to a wait-queue, thus they will be deferred. In case of the on-the-fly approach, the immediate impact of re-configuration is lower. However, there is an overhead incurring for individual transactions (as the effort for re-partitioning for the objects they access will be added to these transactions) which is visible in the higher response time compared to the interruptible approach after the re-configuration. The overhead for individual transactions will decrease, as after some time the re-configuration will be completed.

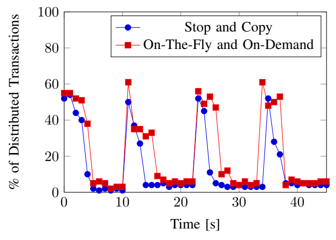
In Figure 8c we have analyzed *Cumulus*' ability to timely react to evolving access patterns. For this, we have used different pools of access patterns. The four workload switches lead to a considerable increase of the percentage of distributed transactions. *Cumulus* is however able to quickly adapt to the workload change which is reflected in the significant reduction of distributed transactions, whereas the spike of distributed transactions remains in a static partitioning approach. Hence, in the presence of dynamically changing access patterns, *Cumulus* outperforms static approaches that can only be optimized for an initial access pattern but become suboptimal when the workload evolves.



(a) Stop and Copy



(b) On-The-Fly and On-Demand



(c) Shifting access patterns

Figure 8: Adaptive behavior of *Cumulus*

D. Discussion

The experiments show that providing the partitioning algorithm with a curated workload is crucial for the reducing the overall cost of data partitioning and for generating a high quality schema. Moreover, *Cumulus* jointly minimizes the number of distributed transactions and avoids hotspots, as both is needed to increase the overall system performance.

Access patterns of applications deployed in the Cloud are dynamic and may change over time. Thus, a partitioning protocol has be adaptive and, at the same time, should incur minimal overhead. Frequent partitioning events that do not provide sufficient gain should not take place as the cost of adaptiveness may overweight their advantage. By curating the workload and by assessing the potential gain of re-partitioning before actually initiating data re-distribution, *Cumulus* allows to avoid unnecessary and costly re-configurations.

VI. Related Work

Database partitioning with the objective of optimal placement to minimize the number of distributed transactions and to evenly distribute the load has been addressed in several approaches, such as [2], [32]–[35]. An optimal configuration of a partitioned database should result in a shared-nothing scenario. In this case, the sites of the database are independent of each other. Shared-nothing databases are a beneficial as they can be subject to virtually unlimited scale-out [15].

Schism [2] introduces a graph-based partition algorithm, which is also used in *Cumulus*. Compared to Schism, *Cumulus* is an adaptive partitioning protocol that supports an on on-the-fly re-partitioning approach.

In *Zephyr* [27], live migration in shared nothing databases is considered. In contrast to *Zephyr*, *Cumulus* never aborts any transaction as a consequence of the re-configuration.

The re-configuration approach of *Cumulus* has some common concepts with *Squall*, which implements an online re-configuration protocol for partitioned databases [28]. As both systems have evolved in parallel, we have not yet compared them through evaluations – this will be part of our future work. Squall, in contrast to *Cumulus*, initiates refresh transactions at the new destination that pull the objects from the old destinations. *Cumulus* follows the push approach and adds migrate operations to user transactions. This allows to lock only those objects that need to be migrated and supposedly leads to a higher degree of concurrency compared to Squall, as Squall locks entire partitions.

AutoStore [35] uses a similar process as *Cumulus* for analyzing an access pattern, for creating a partition scheme from this access pattern, and for comparing it by means of a cost function with the previous schema. However, in contrast to *Cumulus*, AutoStore does not curate the workload; rather, it uses a sliding window to determine the access pattern.

ElasTras [36] supports both a manual mode (based on a domain expert) and an autonomous mode for range partitioning. In the autonomous mode, ElasTras is able to create partitions using a hash-based algorithm. ElasTras supports only so called mini transactions which have a very rigid structure [37], whereas *Cumulus* provides full ACID support without limiting the size and structure of transactions.

Accordion [22] provides an elastic partitioning protocol which is particularly beneficial for the Cloud. It aims at finding the optimal mapping of partitions to servers by taking server capacity into account. Moreover, it is able to scale-out and scale-in as needed. The Accordion approach is complementary to *Cumulus* and the highly modular architecture of *Cumulus* would allow for an easy integration of Accordion.

VII. Conclusion and Outlook

In this paper we have presented *Cumulus*, an adaptive and autonomous partitioning protocol. *Cumulus* implements an on the on-the-fly and on demand re-configuration approach driven by user transactions. Moreover, *Cumulus* is able to curate the workload by filtering out noise before determining how to re-partition data. This increases the partitioning quality and decreases the re-partitioning overhead. We have evaluated *Cumulus* in an OLTP setting based on the TPC-C benchmark. These evaluation results show the overall gain in performance compared to systems in which distributed transactions occur. In particular, the evaluation results show that *Cumulus* outperforms static partitioning approaches and also approaches which do not support the curation of workload before initiating a re-partitioning step.

As part of our future work, we plan to introduce machine learning approaches to learn the optimal threshold configuration for our cost model. Additionally, elastic behavior, a crucial property in Cloud environments, will be added to *Cu-*

mulus. Due to its modular architecture, existing approaches (e.g., as presented in [22]) can be integrated with low effort.

VIII. Acknowledgment

This work has been partly supported by the Swiss National Science Foundation and an AWS research grant.

References

[1] B. Kemme, R. Jiménez-Peris, and M. Patiño-Martínez, *Database Replication*. Morgan & Claypool Publishers, 2010.

[2] C. Curino *et al.*, "Schism: a workload-driven approach to database replication and partitioning," *Proc. VLDB*, 2010.

[3] R. Taft *et al.*, "E-store: Fine-grained elastic partitioning for distributed transaction processing systems," *VLDB*, 2014.

[4] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems*. Addison-Wesley, 1987.

[5] J. Gray and L. Lamport, "Consensus on transaction commit," *ACM TODS*, vol. 31, no. 1, pp. 133–160, March 2006.

[6] J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The dangers of replication and a solution," in *Proc. SIGMOD*, 1996.

[7] E. A. Brewer, "Towards robust distributed systems," in *Proc. PODC*, 2000.

[8] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *ACM SIGACT News*, vol. 33, no. 2, pp. 51–59, June 2002.

[9] D. Abadi, "Consistency tradeoffs in modern distributed database system design: CAP is only part of the story," *IEEE Computer*, 2012.

[10] P. Bailis *et al.*, "Highly available transactions: Virtues and limitations," *Proc. VLDB*, 2013.

[11] P. A. Bernstein and E. Newcomer, *Principles of transaction processing*. Morgan Kaufmann, 2009.

[12] D. Skeen, "Nonblocking commit protocols," in *Proc. SIGMOD*, 1981.

[13] J. Dean, "Designs, lessons and advice from building large distributed systems," *Keynote from LADIS*, 2009.

[14] M. Aslett, "How will the database incumbents respond to NoSQL and NewSQL?" http://www.cs.cmu.edu/∼pavlo/courses/fall2013/static/papers/aslett-newsql.pdf, 2011.

[15] M. Stonebraker, "The case for shared nothing," *IEEE Data Engineering Bulletin*, vol. 9, no. 1, March 1986.

[16] T. M. Özsu and P. Valduriez, *Principles of distributed database systems*, 3rd ed. Springer-Verlag, 2011.

[17] J. C. Corbett *et al.*, "Spanner: Googles globally distributed database," *ACM Transactions on Computer Systems (TOCS)*, vol. 31, no. 3, p. 8, August 2013.

[18] R. Cattell, "Scalable SQL and NoSQL data stores," in *Proc. SIGMOD*, 2011.

[19] D. Agrawal, S. Das, and A. E. Abbadi, *Data management in the Cloud: Challenges and opportunities*. Morgan & Claypool Publishers, 2012.

[20] C. Curino *et al.*, "Relational Cloud: A database service for the Cloud," in *Proc. CIDR*, 2011.

[21] S. Das, D. Agrawal, and A. El Abbadi, "G-store: a scalable data store for transactional multi key access in the Cloud," in *Proc. SoCC*, June 2010, pp. 163–174.

[22] M. Serafini *et al.*, "Accordion: Elastic scalability for database systems supporting distributed transactions," *VLDB*, 2014.

[23] B. G. Robert, *Smoothing, Forecasting and Prediction of Discrete Time Serie*. Prentice-Hall, 1963.

[24] M. Andreolini and S. Casolari, "Load prediction models in web-based systems," in *Proc. VALUETOOLS*, 2006.

[25] J. Harris and H. Stocker, "Maximum likelihood method," *Handbook of Mathematics and Computational Science*, vol. 1, p. 824, 1998.

[26] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, Dec. 1998.

[27] A. J. Elmore *et al.*, "Zephyr: Live migration in shared nothing databases for elastic cloud platforms," in *Proc. SIGMOD*, 2011.

[28] ——, "Squall: Fine-grained live reconfiguration for partitioned main memory databases," in *Proc. SIGMOD*, 2015.

[29] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., 1984.

[30] TPPC Benchmark, http://www.tpc.org/tpcc/, accessed January-2015.

[31] D. J. DeWitt, "The wisconsin benchmark: Past, present, and future," in *The Benchmark Handbook*, 1991, pp. 119–165.

[32] S. B. Navathe and M. Ra, "Vertical partitioning for database design: a graphical algorithm," in *Proc. SIGMOD*, 1989.

[33] W. W. Chu and I. T. Ieong, "A transaction-based approach to vertical partitioning for relational database systems," *IEEE Trans. Software Eng.*, vol. 19, no. 8, pp. 804–812, 1993.

[34] C.-H. Cheng, W.-K. Lee, and K.-F. Wong, "A genetic algorithm-based clustering approach for database partitioning," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 32, no. 3, pp. 215–230, August 2002.

[35] A. Jindal and J. Dittrich, "Relax and let the database do the partitioning online," in *Enabling Real-Time Business Intelligence*. Springer-Verlag, 2012, pp. 65–80.

[36] S. Das, D. Agrawal, and A. El Abbadi, "Elastras: An elastic transactional data store in the cloud," in *HotCloud*, 2009.

[37] M. K. Aguilera *et al.*, "Sinfonia: a new paradigm for building scalable distributed systems," in *Proc. SIGOPS*, 2007.