Analyzing the Performance of Data Replication and Data Partitioning in the Cloud: the BEOWULF Approach

Alexander Stiemer* Ilir Fetai§ Heiko Schuldt*

*Department of Mathematics and Computer Science
University of Basel, Switzerland
{firstname.lastname}@unibas.ch

§Swiss Distance University of Applied Sciences
ilir.fetai@ffhs.ch

***Abstract*—Applications deployed in the Cloud usually come with dedicated performance and availability requirements. This can be achieved by replicating data across several sites and / or by partitioning data. Data replication allows to parallelize read requests and thus to decrease data access latency, but induces significant overhead for the synchronization of updates. Partitioning, in contrast, is highly beneficial if all the data accessed by an application is located at the same site, but again necessitates coordination if distributed transactions are needed to serve applications. In this paper, we analyze three protocols for distributed data management in the Cloud, namely Read-One-Write-All-Available (ROWAA), Majority Quorum (MQ) and Data Partitioning (DP) – all in a configuration that guarantees strong consistency. We introduce BEOWULF, a meta protocol based on a comprehensive cost model that integrates the three protocols and that dynamically selects the protocol with the lowest latency for a given workload. In the evaluation, we compare the prediction of the BEOWULF cost model with a baseline evaluation. The results nicely show the effectiveness of the analytical model and the precision in selecting the best suited protocol for a given workload.**

***Keywords*-cloud data management; data replication; data partitioning.**

I. INTRODUCTION

Applications deployed in the Cloud usually have concrete requirements towards the services they consume. If we consider data management services, which are the main focus of this work, application requirements typically include the desired levels of data availability and consistency, but also aspects related to performance. However, the requirements are not independent, but influence each other. According to the CAP theorem (consistency, availability, partition tolerance) [1], [2], only two of these three properties can be provided at the same time. As tolerance to network partition cannot be sacrificed, applications can either choose to provide strong consistency or availability. Moreover, even in the failure-free case, there is an inherent trade-off between consistency and latency, which is captured by PACELC [3].

There are basically two approaches to provide a high degree of scalability. First, to replicate data and use the additional processing capabilities of the replica sites to distribute the overall load. This perfectly works for read-only or read-mostly workloads. In case of high update rates, however, a considerable coordination overhead between the replica sites is necessary if strong consistency, such as One-Copy Serializability (1SR) [4], is required. Different replication protocols exist at different costs, such as Read-One Write-All-(Available) (ROWA(A)) [5] and quorum-based protocols. Second, data can be partitioned so that the load is evenly distributed across sites; in the ideal case, costly distributed transactions can be completely avoided.

The decision on the optimal data distribution (replication or partition) and the optimal protocol to use is heavily dependent on the application workload and the applications' performance requirements. However, as the workload is usually very dynamic, this decision must continuously be assessed and, if necessary, be adapted.

In this paper, we introduce BEOWULF, a meta-protocol that allows to switch between data replication and data partitioning on the basis of the costs that incur for data management. BEOWULF is based on three protocols, namely ROWAA [5]–[7] and a Majority Quorum (MQ) [8] for the management of replicas, and Data Partitioning (DP) [9] for the (non-replicated) placement and distribution of data across sites. All these protocols have different characteristics; some are more suited for high read loads while others are optimized for a higher percentage of write operations in the workload. BEOWULF uses a comprehensive cost model based on which it is able to dynamically select the protocol that provides the lowest response time given an estimation of the applications' workload.

The contribution of this paper is twofold: First, we introduce the cost model of BEOWULF, which takes into account the characteristics of ROWAA, MQ, and DP. Essentially, it analytically assesses the costs that incur for the different protocols for a given workload. Second, we asses the prediction quality of the cost model by comparing it with the costs of a real deployment. We have evaluated the BEOWULF cost model by comparing its recommendation of the best suited protocol with a real implementation using the TPC-C benchmark. The results confirm the accuracy of the cost

model and with that the ability of BEOWULF to choose the optimal protocol.

The remainder of the paper is structured as follows: Section II summarizes the foundations of data replication and partitioning. In Section III, we introduce the cost model of BEOWULF and present details of the implementation of BEOWULF in Section IV. Section V discusses the results of the evaluation of BEOWULF using TPC-C. Section VI surveys related work and Section VII concludes.

II. FOUNDATIONS

In what follows we will briefly summarize the main concepts related to data replication and partitioning.

1) Data Consistency: 1SR consistency requires that the effects of an interleaved transaction execution on a replicated database is equivalent to a serial execution on a single-copy database. 1SR is therefore a strong consistency model whereas Eventual Consistency (EC) is a weak consistency model that was made popular in the context of CAP. In [10] the authors provide a formal definition of EC which is based on the visibility and arbitration order. The visibility order defines the updates that are visible to a transaction, whereas the arbitration order defines the relative order of the updates. It is well known that eventual consistency can tolerate network partitions, as the arbitration order can remain partially determined for some time after transaction commit. There is a number of different consistency levels that lie in the range defined by the 1SR and EC consistency.

2) Application Workload: A workload is a set of transactions and their occurrence (frequency) at time period p_i:

$$\begin{aligned} W^{p_i} &:= \Omega^{p_i} \times \mathbb{N}^+ \\ &= \{(\omega_1, occ(\omega_1)), \ldots, (\omega_n, occ(\omega_n))\} \\ occ &: \Omega^{p_i} \rightarrow \mathbb{N}^+ \end{aligned} \tag{1}$$

Ω defines the set of all possible transactions ω, whereas $\Omega^{p_i} \subseteq \Omega$ is a set of all transactions occurring in the time period p_i.

A transaction $\omega \in \Omega$ can be described by its access pattern ap being a set of operations that act on specific data objects:

$$\begin{aligned} ap &= \{op_1(do_1), op_2(do_2), \ldots, op_n(do_n)\} \\ op &\in \{r, w\} \end{aligned} \tag{2}$$

Therefore, a workload can also be described as a set of tuples consisting of access patterns and their occurrence:

$$\begin{aligned} W^{p_i} &:= \mathcal{AP} \times \mathbb{N}^+ \\ &= \{(ap_1, occ(ap_1)), \ldots, (ap_n, occ(ap_n))\} \\ ap &\in \mathcal{AP} \\ occ &: \mathcal{AP} \rightarrow \mathbb{N}^+ \end{aligned} \tag{3}$$

$\mathcal{AP}$ defines the set of all possible access patterns and occ the number of occurrences of an access pattern in the workload W^{p_i}.

A. ROWA and ROWAA

ROWA replication protocols ensure that an update transaction commits at all sites before the response is delivered to a client (eager replication). Thus, a read-only transaction can be executed at any single site without any additional coordination. ROWAA also provides eager replication semantics. However, in contrast to ROWA, it provides a higher degree of availability as only the available sites are considered and thus failures of sites can be tolerated. As a consequence, ROWAA protocols need to incorporate complex reconciliation algorithms in order to properly manage recovered sites.

B. Quorum Protocols

In quorum protocols, only a subset of sites is eagerly updated. In order to ensure strong consistency, subsets must be chosen so that two writes or a write and a read on the same data object intersect [11]. Compared to ROWA and ROWAA, quorum protocols reduce the overhead for update transactions. However, this is done at a higher cost for read-only transactions, as reads must access a subset of sites that form a read quorum, and typically this subset includes more than one site. Quorum-based replication protocols differ in the size of read and write quorums, and thus in the overhead they generate for read-only and update transactions. Certain protocols organize the sites in a logical structure and exploit that structure to determine the quorums. Thus, they also differ in the generated overhead for creating and maintaining that structure, for example in case of site failures.

MQ is a simple quorum protocol, in which each site has a non-negative number of votes. The quorums are then chosen in such a way so that they exceed half of total votes [8]:

$$|\text{RQ}| \geq \left\lceil \frac{\#votes}{2} \right\rceil \tag{4}$$

$$|\text{WQ}| > \left\lfloor \frac{\#votes}{2} \right\rfloor + 1 \tag{5}$$

In the basic MQ, each site has the same amount of votes. Hence, a read quorum (rq) consists of half of the sites, if the number of sites is even, or else the majority of the sites. A write quorum (wq) consists of the majority of the sites.

C. Data Partitioning

The goal of data partitioning protocols is twofold. First, they avoid expensive distributed transactions by collocating objects that are accessed inside the same transaction [12]. Second, they avoid performance bottlenecks by evenly distributing the load between the partitions. There are three basic approaches to partition a database: horizontal, vertical, and hybrid partitioning [13]. Horizontal partitioning splits data across partitions based on key ranges. Vertical partitioning splits the data along the line of attributes, and hybrid partitioning is a combination of the other two approaches.

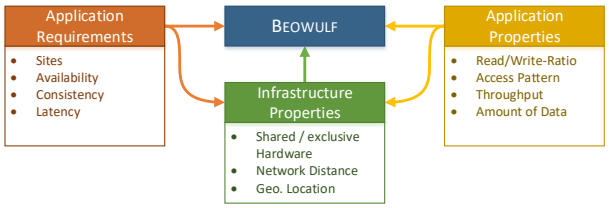


Figure 1. Overview of the discussed parameter space and its classification into the three categories: *application requirements*, *application properties* and *infrastructure properties*

D. Replication vs. Data Partitioning Protocols

In what follows we will analytically compare ROWAA, MQ, and DP protocols in terms of performance.

ROWAA is best suited for read-heavy workloads, as the load can be fully distributed between the different sites in the system. However, as the ratio of updates in the workload increases, the coordination overhead between the replica sites becomes the limiting factor to scalability. This overhead is defined in terms of messages that are exchanged between the sites and their processing overhead. MQ tackles this issue by enforcing coordination only between a subset of sites. However, this comes at a price, as in MQ read-only transactions must also access a subset of sites, which is not the case in ROWAA.

The quality of a partition schema is measured in terms of the number of distributed transactions and the degree of load distribution. In the best case, the schema perfectly matches the access patterns, and by that completely avoids distributed transactions. Moreover, an ideal protocol should evenly distribute the load between the sites. The load may be defined in terms of data size or number of transactions.

III. The Beowulf System & Cost Model

A. Parameter Space

We group the parameter space into three different categories: *Application Requirements*, *Application Properties* and *Infrastructure Properties*. Figure 1 depicts these three categories.

1) Application Requirements (AppReq): contain, for instance, the requirements on the *sites* and / or the *availability*, *consistency* and *latency* of the distributed system. We consider these parameters as time-independent, meaning that changes do not occur frequently.

2) Application Properties (AppProp): are runtime properties, i.e., properties that occur in the running system, for example, *workload*, *throughput* and the *amount of data*. These properties are time-dependent.

3) Infrastructure Properties (InfProp): contain, for instance, the *network distance* between the sites resp. hosts and the network capabilities like *bandwidth* and *latency*, but also other properties like *packet loss* or *data transfer rate*.

Algorithm 1 BEOWULF-Protocol – Client

Require: Address trxExecSiteAddr, Transaction trx

```
Executor exec ← getTrxExecSite(trxExecSiteAddr)

// Figure 2, (1):
// The transaction execution site executes the transaction
// according to the currently applied protocol and its
// configuration.
TransactionResponse trxResp ← exec.execute(trx)
```

Algorithm 2 BEOWULF-Protocol – Beowulf

Require: Properties p

```
List configs ← getAllConfigs(p)

loop
  Config curConfig ← getCurrentConfig()
  Workload w ← getWorkloadFromExecutionSites()
                        // Figure 2, (2)
  Int cost ← MAX_VALUE
  Config newConfig ← null

  for all Config c in configs do
    Int newCost ← cost(p, curConfig, c, w)
    if newCost < cost then
      cost ← newCost
      newConfig ← c
    end if
  end for

  if newConfig ≠ curConfig then
    reconfigureSystem(newConfig)    // Figure 2, (3)
  end if
end loop
```

B. System Model

The BEOWULF system is defined by the set of available protocols, together with their configurations. For example, the configuration of the MQ protocol is determined by the definition of the quorums, whereas the configuration of a DP protocol is determined by its partition schema.

The system consists of sites, which execute transactions according to a protocol and its configuration. The protocol is determined by BEOWULF based on the cost model and the workload. Figure 2 and the Algorithms 1 and 2 describe BEOWULF's meta protocol. (1) Clients execute transactions on the transaction execution sites without any notice about the currently applied protocol / configuration. In the background, BEOWULF monitors transparently to the client the workload occurring at the transaction execution sites by polling the corresponding data (2). The future workload is

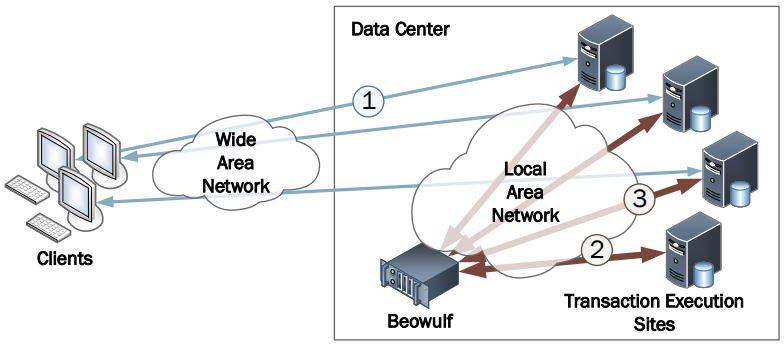


Figure 2. BEOWULF's deployment scenario: BEOWULF interacts with the local transaction execution sites inside a datacenter to gather the required data to schedule reconfigurations if necessary. (1) The client executes the transaction at the specified *Transaction Execution Sites* which executes the transaction according to the currently applied protocol and its configuration. (2) BEOWULF fetches the workload from the *Transaction Execution Sites* and (3) triggers a reconfiguration if needed.

predicted using this data which is then used to calculate the cost for each available configuration. The configuration with the lowest cost is finally applied in the system which may require a reconfiguration at the transaction execution sites (3).

C. Cost Model Parameters

The current version of the cost model considers the following parameters: 1) From the AppReq we use the *number of sites* and the binary parameter *replication*, which are compiled in the configuration parameter. 2) The parameter space of the AppProp is reduced to the *workload* and its properties *read/write-ratio* and *access pattern*. 3) In this work we do not yet consider any parameter from the InfProp; this will be subject to future work.

D. Cost Model

Let $W^{p_i} = \{\omega^1, ..., \omega^n\}$ be a workload of the time period p_i with size n consisting of the transactions ω and $\Delta = \{ROWAA, MQ, DP\}$ be the set of all configurations.

Given a (predicted) workload W^{p_i} (a set of transactions ω) we can calculate the cost for each configuration $\delta \in \Delta$ and we find the new configuration that minimizes the cost

$$\delta_{new} = \arg\min_{\forall\, \delta \in \Delta} \left(\sum_{\omega \in W^{p_i}} \mathrm{cost}(P, \delta_{cur}, \delta, \omega) + c_{ch} \cdot \mathrm{changeCost}(\delta_{cur}, \delta) \right) \tag{6}$$

with the vector $P = (AppReq, AppProp, InfProp)$ containing the properties as mentioned above and δ_{cur} as the currently applied configuration. $\mathrm{cost}(P, \delta_{cur}, \delta, \omega)$ is defined as follows:

$$\mathrm{cost}(P, \delta_{cur}, \delta, \omega) := c_{mon} \cdot \mathrm{trxCost}(\delta, \omega) \tag{7}$$

with the constants c_{mon} and c_{ch} which are used for weighting the cost components. The parameter c_{mon} is used to transform the abstract costs into real monetary costs (e.g., $c_{mon} = 7.6 \cdot 10^{-8}$ USD per distributed transaction[1]). The function $\mathrm{changeCost}(\delta_{cur}, \delta)$ determines or estimates the number of data objects which have to be transferred between the sites to switch the configuration. This number strongly depends on the size of the database. It is scaled by c_{ch} which is a data cost coefficient depending on the system. It abstracts cost values such as the cost for a data transfer which, among others, depends on the site and the network between the sites (e.g., $c_{ch} = 2.28 \cdot 10^{-7}$ USD per data object to be moved/copied[2]).

Using the *Distribution-Degree-Function* introduced in [14], we define:

$$\mathrm{trxCost}(\delta, \omega) := \begin{cases} \left(\dfrac{\mathrm{dD}(\delta, \omega)}{\max \mathrm{dD}(\delta)} \right) & , \text{ if } \max \mathrm{dD}(\delta) > 0 \\ 0 & , \text{ otherwise} \end{cases} \tag{8}$$

with

$$\mathrm{dD}(\delta, \omega) := \#AccessedSites(\delta, \omega) - 1 \tag{9}$$

as the *distribution degree* and

$$\max \mathrm{dD}(\delta) := \#Sites(\delta) - 1 \tag{10}$$

as the *maximum distribution degree*. Please note that we assume each transaction to anyways run on at least one site; hence, the maximum distribution degree specifies the number of additional sites (at most $n-1$, with n being the

[1]Using Amazon Web Services EC2 network pricing listed at https://aws.amazon.com/ec2/pricing/on-demand/ (accessed Nov. 2016) and assuming messages having a size of 1 kB:
$0.02\ {}^{\mathrm{USD}}/_{\mathrm{GB}} \approx 1.9 \cdot 10^{-8}\ {}^{\mathrm{USD}}/_{\mathrm{kB}}$
One distributed transaction ($dTrx$) needs for 2PC four messages:
$4\ {}^{\mathrm{msg}}/_{\mathrm{dTrx}} \cdot 1\ {}^{\mathrm{kB}}/_{\mathrm{msg}} \cdot 1.9 \cdot 10^{-8}\ {}^{\mathrm{USD}}/_{\mathrm{kB}} = 7.6 \cdot 10^{-8}\ {}^{\mathrm{USD}}/_{\mathrm{dTrx}}$

[2]Four sites imply three 2PC-messages; *do* meaning *data object*:
$3\ {}^{\mathrm{dTrx}}/_{\mathrm{do}} \cdot 7.6 \cdot 10^{-8}\ {}^{\mathrm{USD}}/_{\mathrm{dTrx}} = 2.28 \cdot 10^{-7}\ {}^{\mathrm{USD}}/_{\mathrm{do}}$

Table I
$trxCost(\delta, \omega)$ WITH $\Delta = \{ROWAA, MQ, DP\}$

Configuration	Read-Only Transaction	Write Transaction
ROWAA	$= 0$	$= 1$
MQ	≥ 0.5	> 0.5
DP	schema & access pattern dependend	schema & access pattern dependent

overall number of sites in the system) which also need to be considered in case of distributed transactions.

$\text{changeCost}(\delta_{cur}, \delta)$ estimates the costs that incur if there is a need to change a configuration, e.g., for copying / moving data objects. This can be the case if the configuration switches, for instance, from a partitioned to a fully replicated configuration. Then, every data object needs to be copied on all other sites. We define $\text{changeCost}(\delta_{cur}, \delta)$ in a straightforward way as the sum of *location mismatches* (locMis) of D containing all data objects do:

$$\text{changeCost}(\delta_{cur}, \delta) := \sum_{do \in D} \text{locMis}(\delta_{cur}, \delta, do) \quad (11)$$

We define a location mismatch ($locMis$) as a database object which does not reside on the physical location(s) specified by the configuration or schema. Hence, a location mismatch generates costs by the need of copying / moving it to the new location(s).

E. Analysis

Table I shows a rough analysis of $trxCost(\delta, \omega)$ for $\Delta = \{ROWAA, MQ, DP\}$:

1) In case of $\delta = ROWAA$, $\text{trxCost}(\delta, \omega) = 0$ for read-only transactions (ω_{ro}) and $\text{trxCost}(\delta, \omega) = 1$ for transactions containing write operations (ω_{rw}).
   For ω_{ro}: $\text{dD}(ROWAA, \omega_{\text{ro}}) = 0$ and therefore also $\text{trxCost} = 0$. For ω_{rw}: $\text{dD}(ROWAA, \omega_{\text{rw}}) = n - 1$ since a write transaction needs to be executed on all (remaining) sites and therefore $\text{dD}(ROWAA, \omega_{\text{rw}}) = \max \text{dD}(ROWAA)$.
2) For MQ ($\delta = MQ$), $\text{trxCost}(\delta, \omega)$ can be estimated with 0.5 as the lower bound for both read-only and write transactions. The reason is the quorum construction rule (Equations (4) and (5)).
3) The cost for DP strongly depends on the applied partitioning schema and the current access pattern caused by the current workload. A well partitioned system experiencing nearly matching access patterns can have very little cost since only some "outlier-transactions" need to be distributed in the system.

The costs of $\text{changeCost}(\delta_{cur}, \delta)$ for $\Delta = \{ROWAA, MQ, DP\}$ can be estimated as follows:

1) There is no cost at all if $\delta_{cur} = ROWAA$. The reason is that with the ROWAA configuration the data is fully replicated. Switching to partial replication (MQ) or no replication at all (DP) simply means that some data objects will not be maintained anymore.
2) Changing to ROWAA requires copying data. For partial replication, the amount of data to be copied is not as high as for partitioned data. For switching from DP to ROWAA, every data object of the whole database needs to be replicated on every other site which causes high transition costs.
3) The overhead for changing the partition schema is schema dependent. Also the change from MQ to DP depends on the desired partitioning schema since not all data is located everywhere like it is in ROWAA.

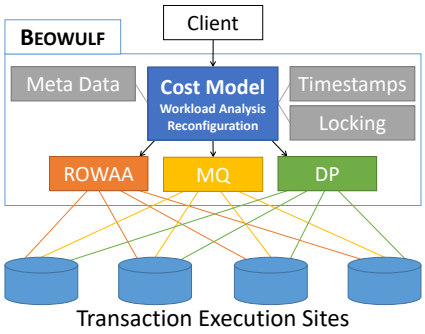


Figure 3. Conceptual architecture of BEOWULF. BEOWULF consists of subsystems which are responsible for collecting meta data, providing a timestamp service, centralized locking and executing transactions using one of the three protocols (ROWAA, MQ, DP).

IV. IMPLEMENTATION

The BEOWULF system as well as the three protocols are implemented as web services. BEOWULF consists of management modules and transaction execution sites as depicted in Figure 3. BEOWULF has four different roles to fulfill: 1) BEOWULF analyses the transactions w.r.t. the resulting workload. This information is needed, for instance, to predict future workloads. 2) It calculates the cost for each protocol using the introduced *Cost Model* and decides whether the actual active protocol is still the one with the lowest cost. If not, it schedules a reconfiguration or protocol switch, resp. 3) BEOWULF steers and monitors the reconfiguration. 4) Additionally, BEOWULF collects empirical data for statistical analysis like the latency of a transaction, current throughput, average workload within a time period, etc. This information and additional meta data can be used for further improvements of BEOWULF and its deployment.

A. Management Modules

BEOWULF uses several management modules which provide services for the transaction execution sites. For instance, the *Meta Data* module as shown in Figure 3 collects meta data of the whole system. The *Timestamp* module provides a central timestamp service and the *Locking* module provides Two-Phase Locking (2PL) [7].

B. Transaction Execution Sites

The transaction execution sites are able to provide 1SR consistency. To achieve this, BEOWULF uses 2PL as concurrency control protocol and Two-Phase Commit (2PC) [15] for the consistent propagation of updates [7].

1) ROWAA: According to the protocol, read-only transactions are executed locally. Write transactions are executed on all sites.

2) Majority Quorum: Our implementation of the MQ protocol is similar to ROWAA. Both share a common code base. However, in contrast to ROWAA, MQ commits only on a subset of all available sites according to the MQ protocol. Also, the execution of read operations differs since for MQ more than one site is needed to read the value of a data object. We construct the quorums at site level, not at data object level. This means that each site has its quorum partners that are inquired for read resp. write requests.

3) Data Partitioning: For the DP part we use Cumulus, which is an implementation of a dynamic data partitioning protocol [14]. Cumulus collects the workload of the last period from all available sites. The workload is then used for workload prediction. With this information Cumulus calculates the partition schema using graph partitioning. This schema is then applied on its transaction execution sites.

C. BEOWULF's Cost Model

The implementation of the *Cost Model* uses the workload provided by the workload analyzer to compute the cost given a certain system configuration. The cost model needs the DP schema with the lowest cost. For the time being, the workload analyzer is trained with the actual workload. The cost calculation (calculating the the distribution degree) of ROWAA and MQ is straightforward. However, to calculate the values for DP, BEOWULF also uses the graph partition engine of our baseline implementation. Running the partition engine requires an initial schema. In the baseline implementation, this initial schema is obtained by querying the whole underlying database. For BEOWULF, the initialization is simply replayed in an in-memory database. Using this initial schema and the provided workload, the partition engine computes the new schema. This in turn is used to compute the cost for the DP configuration by comparing the desired locations (provided by the schema) of the data objects.

V. EVALUATION

A. Evaluation Environment & Parameters

Our evaluation environment consists of one active client executing the benchmark, the management nodes and the transaction manager sites. The specification of the nodes used in our evaluation is summarized in Table II. The servers are running in a cluster built of commodity hardware, interconnected via a 1 GBit/s Ethernet network.

The parameters of our tests are summarized in Table III. We use the TPC-C benchmark (with a scale factor of 1) to emulate our workload [16]. Moreover, we use the parameters $c_{mon} = 1$ and $c_{ch} = 0$ as the main objective in our evaluation is to assess the quality of BEOWULF's prediction, not to actually apply changes to the current configuration.

Table II
SPECIFICATION OF THE NODES USED IN THE EVALUATION

	Sites & Management	**Client**
CPU	Intel Pentium D 945 (2 C / 2 T @ 3.4 GHz)	Intel Core i5-2520M (2 C / 4 T @ 2.5 GHz)
RAM	2 GB	8 GB
Network	1 GBit/s Ethernet	1 GBit/s Ethernet
OS	Ubuntu Server 14.04 LTS	Ubuntu Server 14.04 LTS
Arch.	amd64	amd64
Software	Oracle Java 1.8.0_77 Apache Derby 10.11.1.1 Apache Tomcat 8.0.33 JAX-WS RI 2.2.10	OpenJDK 1.8.0_91

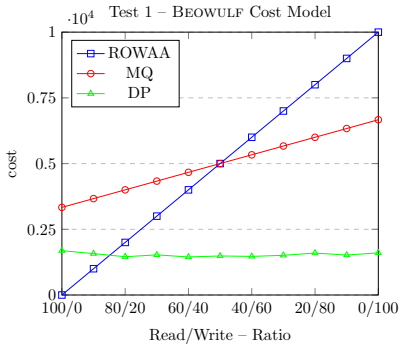
Table III
TEST PARAMETER DESCRIPTION

Workload Mix	Read/Write-Ratio (r/w-ratio), lies within [100 / 0, 0 / 100].
Workload-Incr.	The number by which the read proportion is the decreased resp. write increased.
# of Threads	The number of threads sending transactions.
Threads-Incr.	The number by which the quantity of threads is increased.
# Trx per Thread	The number of transactions a thread sends to the system.
Transaction Size	The number of queries/statements in a transaction.
Storage Nodes	Number of sites/hosts/replicas.
Replication	binary – If required, DP is excluded.

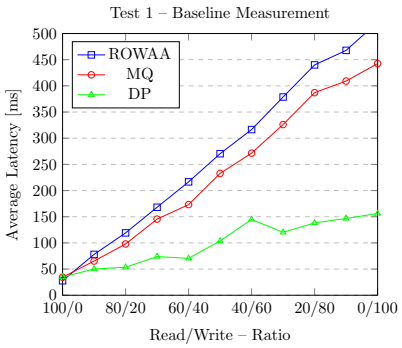
B. Evaluation Results

1) Test 1 – Variable Read/Write-Ratio: Our first test uses the following parameters: 10 threads execute 1,000 transactions on four sites, each transaction containing three operations. In this test, the r/w-ratio is varied from read-only (r / w = 100 / 0) to write-only (r / w = 0 / 100) in steps of 10. The results of this test are illustrated in Figure 4.

Figure 4a depicts the outcome of BEOWULF's Cost Model. For ROWAA, the number increases linearly since all read-only transactions are executed locally and every write transaction on all available sites. As expected, MQ has higher costs for read-heavy transactions since two of the four sites are used for transaction execution. However, for write-heavy transactions, the costs of MQ are lower than the ones of ROWAA since only three of the four sites are needed for execution. The cost estimation of DP shows that there are few distributed transactions and these may require



(a) BEOWULF results of Test 1. The lowest cost determines the suggested protocol/configuration.



(b) Baseline results of Test 1. The lower the latency the better.

Figure 4. Results of Test 1: 10 threads, 1,000 transactions per thread, transaction size of 3, 4 transaction execution sites

only one to two additional sites for execution. However, most transactions can be executed locally.

Figure 4b shows the measured average latency of our baseline system. As expected, for the read-only (r / w = 100 / 0) scenario ROWAA is the best of the three protocols (although the differences between the protocols are minimal). However, for higher write ratios, the average latency of ROWAA increases the most. The reason is that all four nodes have to execute write transactions and participate in the commit phase. In contrast to ROWAA, only three nodes are needed for write transactions in MQ. But to do so, MQ needs at least two nodes to execute read transactions. This results in only a slight improvement compared to ROWAA. The average latency of DP has the lowest increase because most transactions, regardless if read-only or write transactions, can be executed locally.

A comparison of the results of BEOWULF and the baseline regarding minimum costs and minimum latency, we see that from a transction mix of r / w = 80 / 20 onward, the minimum for both is achieved by DP. BEOWULF and the baseline actually differ for r / w = 90 / 10. Here, BEOWULF's minimum is ROWAA whereas the actual minimum regarding latency is DP. For the read-only workload, the minimum for both is ROWAA. Comparing both results under the assumption of required replication, which invalidates DP as a valid protocol, results in the following observation: the minimum cost for read-heavy workloads (r / w = 90 / 10 to 50 / 50) calculated by BEOWULF has the ROWAA protocol whereas the minimum latency for the baseline is achieved by MQ. For write-heavy workloads (r / w = 50 / 50 to 0 / 100) the minimum for both is MQ.

Hence, the minimum costs predicted by BEOWULF correctly correspond to the minimum latency of the baseline implementation if DP is considered. When comparing only ROWAA and MQ, BEOWULF correctly selects the best protocol for high update rates.

2) Test 2 – Variable Read/Write-Ratio & Variable Number of Threads: Our second test varies the number of threads additionally to the r/w-ratio from 10 to 50. Each thread executes 200 transactions resulting for 50 threads in 10,000 transactions in total (like for Test 1).

The results of BEOWULF are shown in Figures 5a to 5c and summarized in Figures 5g and 5i. Please note that both summaries are depicted as meshes instead of only points for better visibility. The outcome of Figures 5a to 5c is straightforward by evaluating the appropriate cost function. The summary (Figure 5g) shows, as expected, that for read-heavy workloads BEOWULF suggests ROWAA and for write-heavy DP. We did not expect that the transition from ROWAA to DP shifts towards higher write workloads for a higher number of workers. Instead, we expected the transition to be independent of the number of workers. Note that an increase of the workers corresponds to an increase of number of transactions.

Figures 5d to 5f depict the results of the baseline which correspond mostly with our expectations. The results are summarized in Figures 5h and 5j. Increasing the number of workers also increases the latency. For heavy-read or read-only workloads, the latency stays nearly constant, only a slight increase is visible as the shared locks used do not block transactions. However, we did not expect that DP's latency increase is that low. Even for 50 workers and heavy write workloads, the increase is roughly a third compared to ROWAA. A simple explanation is that for ROWAA three more sites have to participate in the atomic commit via 2PC. Having fewer sites participating leads to a lower waiting time, for example when waiting for the votes of the other participants. For read-only workloads, we expected ROWAA to be the best protocol. However, Figure 5h shows that for (r / w = 100 / 0, no. threads = 10) MQ has the lowest latency although both protocols actually show similar values. The reason for MQ being better might simply be environmental causes or other effects.

A comparison of Figure 5g with Figure 5h points out that the differences are again for the r / w = 90 / 10 workload. BEOWULF's result is in general ROWAA whereas the DP

configuration yields the minimum latency. If replication is required, meaning that DP is no valid choice, the result of the comparison between BEOWULF and the baseline is similar to Test 1. Figure 5i shows the same switch from ROWAA to MQ for heavy-write workloads whereas the baseline's minimum is MQ even for read-heavy workloads (Figure 5j).

Again, BEOWULF correctly predicts the best protocol out of ROWAA, MQ, and DP. If replication is needed for reasons of availability, i.e., the without DP option, then BEOWULF correctly predicts the best protocol for high update rates. Even though the prognosis is not in line with the baseline measurement for read-heavy workloads when choosing between ROWAA and MQ, the consequences can rather be neglected as no or few additional coordination efforts are needed in both protocols for these workloads.

VI. RELATED WORK

As extension to the CAP theorem, Partition-Availability-Consistency-Else-Latency-Consistency (PACELC) [3] describes the latency vs. consistency trade-off for failure-free distributed systems. To minimize latency, several Concurrency Control Models (CCMs) [17] exist. 1SR [4] executes transactions in a distributed system as if they were serially executed on one database; Strong 1SR and Strong Session 1SR [18] are even stronger CCMs. In [19], snapshot isolation [20] is extended for distributed databases. Causal consistency (e.g., [21], [22]) and eventual consistency (e.g., [10], [23]) are examples for weaker CCMs.

In [11], ROWAA is compared with other replication protocols such as quorum, grid- and tree-based protocols w.r.t. availability and scalability. [24] compares the latency of ROWAA with MQ and Log-Write Tree Quorum.

Workload-driven systems change their behavior or (internal) configuration in case of changing workloads. They are especially of interest when using data partitioning protocols since the partitioning itself depends on the actual (or future) workload. In contrast, replication protocols usually do not deal with varying user behavior as long as the load in the entire system stays nearly the same. Schism [12] is a graph-based, workload-driven partitioning system for transactional workloads. Is uses a graph with data records as nodes and edges that correspond to the co-occurrence of records in the same transaction. The partitioning then aims at minimizing the number of distributed transactions. AutoStore [25] analyses access patterns (APs) using a sliding window approach to create its partitioning schema. It compares the currently applied schema with a new schema candidate using a cost model that also considers the transition cost. Cumulus [14] is a dynamic data partitioning protocol, also using a graph based approach. To change the currently applied partitioning schema dynamically, Cumulus derives its schema from an analysis of the current workload. If the workload changes, the system performs a dynamic reconfiguration.

The workload is one of the two main parameters of BEOWULF's cost model; hence the quality of the prediction of the expected workload is crucial. The better these predictions are, the better the cost model is in optimizing the system for future workloads. Different models are proposed, e.g., the Exponential Moving Average (EMA) [26] or the Autoregressive Integrated Moving Average (ARIMA) [27].

[28] uses the ARIMA model to predict future application workload in Cloud environments. The proactive model is implemented in a workload Analyzer and evaluated using real web server traces.

Live-migration techniques and that allow to dynamically migrate data to cope with changing workloads are relevant when the current configuration and/or deployment is no longer optimal for the expected workload. Zehpyr [29] combines pulling transactions and asynchronous pushing to migrate data. Zephyr provides ACID guarantees [30] during migration. The consistency is guaranteed by the use of 2PL and 2PC. The Squall migration system [31] efficiently performs a dynamic reconfiguration of partitioned databases with strong consistency. The reconfiguration is done by migrating the data asynchronously, by means of distributed transactions, during execution of the actual transaction. Squall keeps track of the migrated data items. This allows its transaction managers to decide whether the transaction can be executed or not. If not, the data has first to be pulled. The Lazy Snapshot Isolation Rule (LSIR) [32], allows a transparent live-migration under snapshot isolation. During the copy process, LSIR keeps track of the operations changing the state of the source database. These operations need to be re-applied on the target database. Hand-over takes place after these migration operations.

VII. CONCLUSION

In this paper, we have introduced BEOWULF, a system tailored to distributed data management which is able to dynamically switch between data replication and data partitioning, based on the actual and expected workload. The heart of BEOWULF is a cost model that is used as a basis to compare three data management protocols: ROWAA and MQ for data replication with strong consistency constraints, and data partitioning. In the evaluation, we have compared the recommendation derived from the analytical analysis of the three protocols with the results of measurements using the TPC-C benchmark. The results show the effectiveness of BEOWULF and its suitability for adaptive data management.

So far, we have considered either replication or partitioning, but no combination of both. In our future work, we aim at modeling also hybrid cases (partial replication, combined with data partitioning). The objective is to extend BEOWULF and to identify, for each workload, the best possible protocol (or protocol combination) for data management. As the quality of the workload estimation is essential for this, we also plan to use machine learning techniques to predict the

future workload of applications and also to assess the effects of the window size in which the access pattern is collected.

REFERENCES

[1] E. Brewer, "Towards robust Distributed Systems (Abstract)," in *Proc. PODC'2000*. Portland, OR, USA: ACM, Jul. 2000.

[2] S. Gilbert and N. A. Lynch, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services," *SIGACT News*, vol. 33, no. 2, pp. 51–59, 2002.

[3] D. Abadi, "Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story," *Computer*, vol. 45, no. 2, pp. 37–42, Feb. 2012.

[4] R. Attar, P. Bernstein, and N. Goodman, "Site Initialization, Recovery, and Backup in a Distributed Database System," *Software Engineering, IEEE Transactions on*, vol. SE-10, no. 6, pp. 645–650, Nov 1984.

[5] J. Rothnie, P. Bernstein, S. Fox, N. Goodman, M. Hammer, T. Landers, C. Reeve, D. Shipman, and E. Wong, "Introduction to a System for Distributed Databases (SDD-1)," *ACM TODS*, vol. 5, no. 1, pp. 1–17, Mar. 1980.

[6] P. Bernstein, D. W. Shipman, and J. Rothnie, "Concurrency Control in a System for Distributed Databases (SDD-1)," *ACM TODS*, vol. 5, no. 1, pp. 18–51, Mar. 1980.

[7] P. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[8] R. H. Thomas, "A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases," *ACM Trans. Database Syst.*, vol. 4, no. 2, pp. 180–209, Jun. 1979.

[9] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou, "Vertical Partitioning Algorithms for Database Design," *ACM Transactions on Database Systems*, vol. 9, pp. 680–710, 1984.

[10] S. Burckhardt, D. Leijen, M. Fähndrich, and M. Sagiv, "Eventually Consistent Transactions," in *Proc. 2012*. Tallinn, Estonia: Springer-Verlag, pp. 67–86.

[11] R. Jiménez-Peris, M. Patiño Martínez, G. Alonso, and B. Kemme, "Are Quorums an Alternative for Data Replication?" *ACM TODS*, vol. 28, no. 3, pp. 257–294, Sep. 2003.

[12] C. Curino, E. Jones, Y. Zhang, and S. Madden, "Schism: A Workload-driven Approach to Database Replication and Partitioning," *Proc. VLDB '10*, vol. 3, no. 1-2, Sep. 2010.

[13] M. T. Özsu and P. Valduriez, *Principles of Distributed Database Systems, Third Edition*. Springer, 2011.

[14] I. Fetai, D. Murezzan, and H. Schuldt, "Workload-Driven Adaptive Data Partitioning and Distribution – The Cumulus Approach," in *Proc. SCDM '15*, Oct. 2015.

[15] J. Gray, "Notes on Data Base Operating Systems," in *Operating Systems, An Advanced Course*. London, UK, UK: Springer-Verlag, 1978, pp. 393–481.

[16] TPC-C Benchmark, http://www.tpc.org/tpcc/, 2010, online; accessed July 2016.

[17] D. Terry, "Replicated Data Consistency Explained Through Baseball," Tech. Rep., October 2011.

[18] V. Zuikevičiūtė and F. Pedone, "Correctness Criteria for Database Replication: Theoretical and Practical Aspects," in *Proc. OTM '08*. Monterrey, Mexico: Springer, 2008.

[19] S. Elnikety, W. Zwaenepoel, and F. Pedone, "Database Replication Using Generalized Snapshot Isolation," in *Proc. SRDS '05*. IEEE, 2005, pp. 73–84.

[20] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil, "A Critique of ANSI SQL Isolation Levels," in *Proc. SIGMOD '95*. San Jose, USA: ACM, 1995, pp. 1–10.

[21] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "Don'T Settle for Eventual: Scalable Causal Consistency for Wide-area Storage with COPS," in *Proc. SOSP '11*. Cascais, Portugal: ACM, 2011, pp. 401–416.

[22] M. Raynal, G. Thia-Kime, and M. Ahamad, "From Serializable to Causal Transactions for Collaborative Applications," in *Proc. EUROMICRO '97*, Sep. 1997, pp. 314–321.

[23] M. Shapiro and B. Kemme, "Eventual Consistency," in *Encyclopedia of Database Systems (online and print)*, M. T. Özsu and L. Liu, Eds. Springer-Verlag, Oct. 2009.

[24] A. Stiemer, I. Fetai, and H. Schuldt, "Comparison of Eager and Quorum-based Replication in a Cloud Environment," in *Proc. SCDM '15*, Oct. 2015.

[25] A. Jindal and J. Dittrich, *Relax and Let the Database Do the Partitioning Online*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 65–80.

[26] A. Lawrance and P.Lewis, "An Exponential Moving-Average Sequence and Point Process (EMA1)," *Journal of Applied Probability*, vol. 14, no. 1, pp. 98–113, 1977.

[27] G. E. P. Box and G. Jenkins, *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

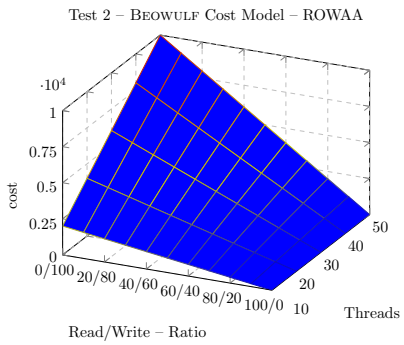
[28] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS," *IEEE Trans. Cloud Comput.*, vol. 3, no. 4, pp. 449–458, Oct. 2015.

[29] A. J. Elmore, S. Das, D. Agrawal, and A. El Abbadi, "Zephyr: Live Migration in Shared Nothing Databases for Elastic Cloud Platforms," in *Proc. SIGMOD'11*. ACM, 2011, pp. 301–312.

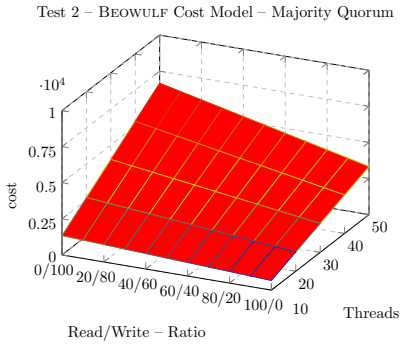
[30] T. Haerder and A. Reuter, "Principles of Transaction-oriented Database Recovery," *ACM Comput. Surv.*, vol. 15, no. 4, pp. 287–317, Dec. 1983.

[31] A. J. Elmore, V. Arora, R. Taft, A. Pavlo, D. Agrawal, and A. El Abbadi, "Squall: Fine-Grained Live Reconfiguration for Partitioned Main Memory Databases," in *Proc. SIGMOD'15*. Melbourne, Australia: ACM, 2015, pp. 299–313.

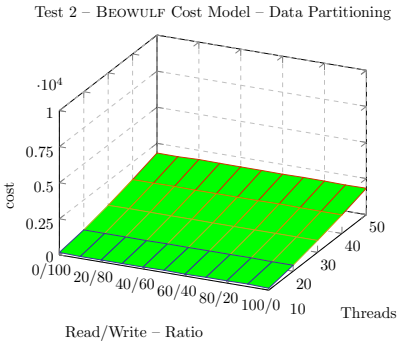
[32] T. Mishima and Y. Fujiwara, "Madeus: Database Live Migration Middleware Under Heavy Workloads for Cloud Environment," in *Proc. SIGMOD '15*. ACM, 2015, pp. 315–329.



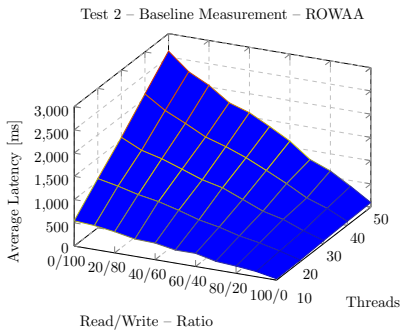
(a) Beowulf ROWA(A) results.



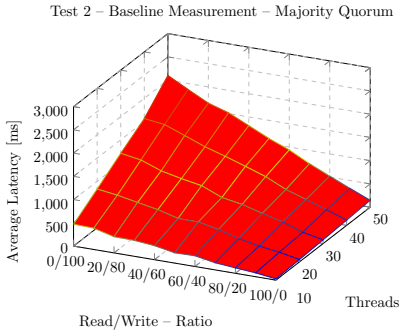
(b) Beowulf Majority Quorum (MQ) results.



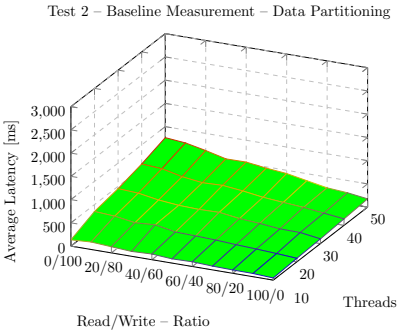
(c) Beowulf Data Partitioning (DP) results.



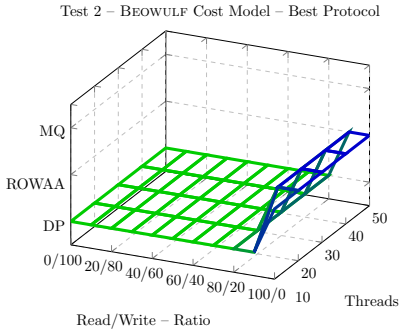
(d) Baseline ROWA(A) results.



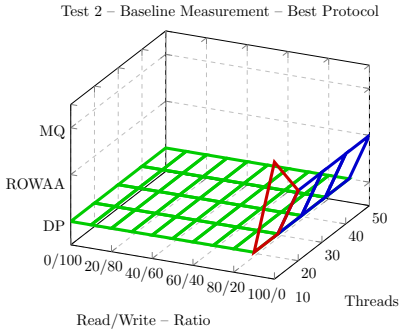
(e) Baseline MQ results.



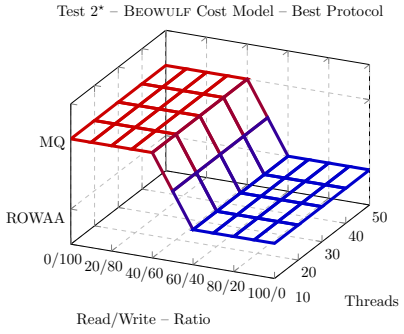
(f) Baseline DP results.



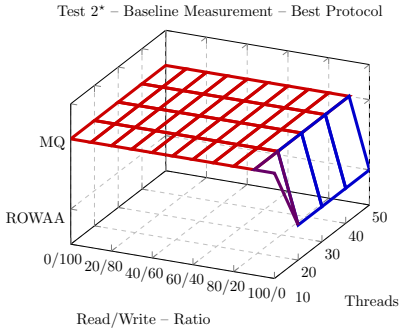
(g) Beowulf's best protocol in terms of cost.



(h) Baseline's best protocol in terms of latency.



(i) Beowulf's best protocol in terms of cost requiring replication.



(j) Baseline's best protocol in terms of latency requiring replication.

Figure 5. Results of Test 2: 10 to 50 threads, 200 transactions per thread, transaction size of 3, 4 transaction execution sites