Dynamic Cost-Efficient Replication in Data Clouds

Nicolas Bonvin, Thanasis G. Papaioannou and Karl Aberer
School of Computer and Communication Sciences
École Polytechnique Fédérale de Lausanne (EPFL)
1015 Lausanne, Switzerland
Email: firstname.lastname@epfl.ch

ABSTRACT

Hardware failures in current data centers are common partly due to the higher data scales supported. Data replication is the common approach for improving availability. However, mostly static replication approaches have been proposed, i.e. the number of replicas and their locations are fixed. Moreover, the geographical diversity of data locations has not explicitly been considered. In this paper, we propose a cost-efficient replication scheme across data centers that dynamically adapts the number of replicas employed per partition to the query load, while maintaining availability guarantees in case of failures. Our approach employs a virtual economy that is experimentally proved in a simulated environment to achieve load balancing among data servers at the minimum cost.

Categories and Subject Descriptors

H.3.2 [**Information storage and retrieval**]: Information Storage; H.3.4 [**Information storage and retrieval**]: Systems and Software—*Distributed systems*; H.2.4 [**Database Management**]: Systems—*Distributed databases*; E.1 [**Data Structures**]: Distributed data structures; E.2 [**Data Storage Representations**]: Hash-table representations

General Terms

Reliability, Economics, Design

Keywords

key-value store, geographical diversity, fault tolerance, economic model

1. INTRODUCTION

Recent large Web applications make heavy use of distributed storage solutions in order to be able to scale up. As data scales up its availability becomes more crucial and more important, as even reliable hardware may fail. As reported in [13], failures of any type in current data centers are frequent, e.g. overheating, power (PDU) failures, rack failures, network failures, hard drive failures, network rewiring and maintenance, etc. Moreover, natural disasters, e.g. a tornado destroying a complete data center, or various attacks (DDoS, terrorism, etc.) may happen. Also, as [3] suggests, Internet availability varies from 95% to 99.6%. Geographic proximity significantly affects data availability; e.g., in case of a PDU failure ∼500-1000 machines suddenly disappear, or in case of a rack failure ∼40-80 machines instantly go down. However, to the best of our knowledge, no existing solution ensures that data replicas are located in different racks, rooms or even data centers. Current systems usually rely on randomness to diversify the physical server that hosts the data; e.g. in [14], [9] node IDs are randomly chosen, so that peers that are adjacent in the node ID space are geographically diverse with a good probability.

In this paper, we present a reliable key-value storage system with a cost-efficient fully distributed replication scheme that maintains high availability guarantees for the data owner despite failures leveraging the geographical diversity of the data servers. The number of data replicas also dynamically adapts to the query load and the capabilities of the data servers, so as the total load to be efficiently balanced. To this end, we define a fine-grained virtual economy where each data partition chooses to replicate, migrate or delete itself based on individual optimization objectives. Simulations experiments show that the proposed approach maximizes the efficient utilization of the system. The rest of the paper is organized as follows: In Section 2, we present the global optimization problem that we address. In Section 3, the key-value data store is presented. In Section 4, we define our virtual economy to decentralize the problem. In Section 5, we present experimental results on the effectiveness of the approach, and in in Section 6, we conclude our work.

2. PROBLEM DEFINITION

The data is split into M partitions, where each partition i has r_i distributed replicas. Take that N servers are present in the data cloud.

2.1 Maximize data availability

The first objective of the data owner is to provide the highest availability for a partition i, by placing all of its replicas in a set R_i of different servers. Data availability generally increases with the geographical diversity of the selected servers. Obviously, the worst solution in terms of data availability would be to put all replicas at a server with equal or worse probability of failure than others.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ACDC'09, June 19, 2009, Barcelona, Spain.
Copyright 2009 ACM 978-1-60558-585-7/09/06 ...$5.00.

We denote as F_j a failure event at server $j \in R_i$. These events may be independent to each other or correlated ones. If we assume without loss of generality that events $F_1 \dots F_k$ are independent and that events $F_{k+1} \dots F_{|R_i|}$ are correlated, then the probability a partition i to be unavailable is defined as follows:

$$Pr(i \text{ unavailable}) = Pr(F_1 \cap F_2 \cap \ldots \cap F_{|R_i|}) = \prod_{j=1}^{k} Pr(F_j) \cdot Pr(F_k|F_{k+1} \ldots \cap F_{|R_i|}) \cdot Pr(F_{k+1}|F_{k+2} \cap \ldots \cap F_{|R_i|}) \cdot \ldots \cdot Pr(F_{|R_i|}) \,, \quad (1)$$

if $F_{k+1} \cap F_{k+2} \cap \ldots F_{|R_i|} \neq \oslash$.

2.2 Minimize communication cost

While geographical diversity increases availability, it is also important to have servers geographically close to each other, in order to save bandwidth during replication and to reduce latency in write operations and during conflict resolution for maintaining data consistency. Let $\vec{\vec{L}}$ be a $M \times N$ *location* matrix with its element $L_{ij} = 1$ if a replica of partition i is stored at server j and $L_{ij} = 0$ otherwise. Then, we maximize data proximity by minimizing network costs for each partition i, e.g. the total communication cost for conflict resolution of the mesh network of servers where the replicas of the partition i are stored. In this case, the network cost c_n for conflict resolution of the replicas of a partition i can be given by

$$c_n(\vec{L_i}) = sum(\vec{L_i} \cdot \vec{\vec{NC}} \cdot \vec{L_i}^T) \,, \quad (2)$$

where $\vec{\vec{NC}}$ is a strictly upper triangular $N \times N$ matrix of the mean communication cost between any two servers and *sum* denotes the sum of matrix elements.

2.3 Minimize real costs

The data owner has to periodically pay the operational costs of each server where he stores replicas of his data partitions. The operational cost of a server is mainly influenced by the quality of the hardware, the physical hosting price, the access bandwidth allocated to the server and its storage, CPU and bandwidth overhead. The data owner wants to minimize his expenses by replacing expensive servers with cheaper ones, while maintaining a certain minimum data availability promised by SLAs to his clients. At the same time, he wants to minimize latency. The overall optimization problem can be formulated as follows:

$$\begin{aligned} &\min \vec{L_i} \ \vec{c}^T + c_n(\vec{L_i}), \forall i \\ &\text{s.t.} \\ &1 - Pr(F_1^{L_{i1}} \cap F_2^{L_{i2}} \cap \ldots F_N^{L_{iN}}) \geq th \,, \end{aligned} \quad (3)$$

where $\vec{c}$ is the vector of operational costs of servers with its element c_j being an increasing function of the replicas located at server j. F_j^0 means that a failure event at server j is excluded from the intersection and th is a certain minimum availability threshold promised by the data owner. This constrained optimization problem takes $2^{M \cdot N}$ possible solutions and it is feasible only for small sets of servers and partitions.

3. SKUTE: KEY-VALUE STORE

Skute is designed to provide low and guaranteed response time on read and write operations, to ensure replicas dispersion in an economically efficient way and to minimize bandwidth consumption as well as latency. The number of data replicas and their placement are handled by a distributed cost-aware algorithm. To this end, data replication is dynamically adapted to the distribution of query load and to failures of any kind so as to maintain high data availability.

3.1 Physical node

We take that a physical node (i.e. a server) belongs to a rack, a room, a data center, a country and a continent. Note, that even more fine-grained geographical attributes could be employed. In order to precisely identify its geographical location each physical node has a label of the form "continent-country-datacenter-room-rack-server". Continents are labeled with two letters as follows: Africa (AF), Asia (AS), Europe (EU), North America (NA), South America (SA), Oceania (OC), Antarctica (AN). Countries are labeled using two characters according to ISO 3166-1 alpha-2 (http://www.iso.org/iso/country_codes) recommendations. Labels for data centers, rooms, racks and servers should not contain more than 3 characters. Thus, a possible label for a server located in a data center in Berlin may be "EU-DE-BE1-C12-R07-S34". Labeling is independent from servers' hostname or IP addresses so that it can be possible to easily make scripts for configuring and starting new physical nodes. This is particularly useful when running nodes using cloud computing provider's architectures. Potential failures or the addition of new physical nodes are determined by a distributed membership protocol, such as those in [11].

3.2 Virtual node

Based on the conclusions of [5], Skute is build using a ring topology and a variant of consistent hashing [8]. Data is identified by a key (produced by one-way cryptographic hash function, e.g. MD5) and its location is given by the hash function of this key. The key space is split into partitions. A physical node (i.e. a server) gets assigned to multiple points in the ring, called tokens, in order data and load uniformity to be improved. A virtual node holds data for the range of keys in (*previous token*, *token*], as in [4]. A physical node hosts a varying amount of virtual nodes depending on the query load, the size of the data managed by the virtual nodes and its own capacity (i.e. CPU, RAM, disk space, etc.).

3.3 Routing

As Skute is intented to be used for real-time applications, a query should not be routed through multiple servers before reaching its destination. Routing has to be efficient, therefore every server has enough information in its routing table to route a query directly to its final destination. Skute could be seen as a zero-hop DHT, similarly to [4]. The routing table is actively updated using a gossiping protocol when a virtual node moves to a new physical node, replicates or commits suicide. This mechanism works well for practical systems containing a limited number (a few hundreds) of servers. Moreover, as experimentally proved in 5, almost no routing table updates are expected in equilibrium with stable system conditions, i.e if the mean query load and the number of servers remain fixed.

Only virtual node migrations and suicides may have an negative impact on routing. First, after a migration the old physical node updates its routing table, such that requests for this particular virtual node are directly forwarded to the new physical node. Second, only one virtual node of the same partition is allowed to suicide at the same epoch. This can be achieved employing Paxos distributed consensus algorithm [10] among virtual nodes of the same partition. The virtual node collectively agreed to commit suicide notifies its physical node to update its routing table, so as to point to the locations of the surviving virtual nodes of the partition.

Stale routing tables only increase the number of hops needed to route a request. A physical node not able to process a request will forward it to the corresponding node by looking at its routing table. Note that a query can be initiated at any physical node. If there is no matching entry for the requested virtual node, the request is forwarded to a random physical node. Eventually, the request will be forwarded to a physical node with a routing entry for the requested virtual node and it will reach its final destination. Upon locating the virtual node, all stale routing entries, which are tracked in the request message in the discovery phase, are restored to the current location of the node. When the system is in a transient phase, there is an obvious trade-off between the freshness of routing table information and the communication cost associated to the update messages. Also, when new physical nodes join the network they copy the routing table of another random one. Upon server failure, the routing table entries associated to that server are deleted by all nodes. The evaluation of the effectiveness of this approach for route maintenance is left for future work.

Table 1: Possible routing table for 3 servers and 5 partitions.

Token range	Label of physical node
(0.0-0.2]	NA-US-NY2-C01-R02-S12
(0.2-0.4]	EU-DE-BE1-C12-R07-S34
(0.4-0.6]	EU-CH-GVA-C06-R12-S01
(0.6-0.8]	NA-US-NY2-C01-R02-S12
(0.8-0.0]	EU-CH-GVA-C06-R12-S01

4. ECONOMIC MODEL

The data owner rents storage space in data servers in several data centers around the world and pays a monthly usage-based *real rent*. Each virtual node is responsible for the data in its key range and should always try to keep data availability above a certain minimum level of confidence while minimizing the associated costs (i.e. latency and cost). To this end, a virtual node can be assumed to act as an autonomous agent on behalf of the data owner to achieve these goals. Time is assumed to be split in epochs. At each epoch, a virtual node may replicate or migrate its data to another server, or suicide (i.e. delete its data replica). It also pays a *virtual rent*, which is defined later in this section and it is a proxy of the possible real rent, to the server where its data is stored. These decisions are made based on the query rate for the virtual node, the renting costs and the maintenance of high availability upon failures. There is no global coordination and each virtual node behaves independently. The virtual rent of each server is announced at a board and is updated at the beginning of a new epoch.

4.1 Board

At each epoch, the virtual nodes need to know the virtual rent price of the servers. Therefore a server in the network is elected (i.e. by a leader election distributed protocol) to store the current virtual rent per epoch of each server. This centralized approach has a low communication overhead per epoch, but it requires the elected server to be trusted. An alternative approach in a non-cooperative setting would be each virtual node to contact directly each server to get the updated price. Thus, there is an obvious trade-off between network efficiency and trust. When a new server is added to the network, the data owner estimates its *confidence* essentially based on its hardware components and its location. This estimation depends on technical factors (e.g. redundancy, security, etc.) as well as non-technicals factors (e.g. political and economical stability of the country, etc.) and it is rather subjective. Confidence values of servers are stored at the board in a trustworthy setting, while they can be stored at each virtual node in case that trustworthiness is an issue.

4.2 Physical node

The virtual rent price of a physical node for the next epoch is an increasing function of its query load and its storage usage at the current epoch and it can be calculated by the following simple formula:

$$virtual\_rent = up \cdot (storage\_usage + query\_load)\,, \quad (4)$$

where up is the unit price for marginal usage of the server, which can be calculated dividing the real rent of the previous month to the mean usage of the server in the previous month. The real rent price per server also takes into account the network cost for communicating with the server. We can safely assume that the communication cost increases with the distance of communicating servers. Thus, multiplying up with the query load satisfies the network proximity objective. The query load and the storage usage at the current epoch are considered to be good approximations of the ones at the next epoch, as they are not usually expected to significantly change at very small time scales, such as a time epoch. Note that the virtual rent reflects usage in small time scales, while the real rent reflects usage on a monthly basis. They are both dynamic, when the system is at a transient phase, but they are constant at steady state, as explained in Section 5.2. The monthly real rent paid by the network owner influences the virtual rent price such that an expensive server tends to be also expensive in the virtual economy. A server agent residing at the server calculates per epoch its virtual rent price.

4.3 Maintaining availability

A virtual node always tries to keep the data availability above a minimum level th (e.g. 99%) specified by the data owner, as specified in Section 2. As estimating the probabilities of each server to fail necessitates access to an enormous set of historical data and private information of the server, we approximate the potential availability of a virtual node by means of the geographical diversity of the servers that host it. Therefore, the availability of a virtual node i is de-

fined as the sum of *diversity* of each distinct pair of servers, i.e.:

$$avail_i = \sum_{i=0}^{|S_i|} \sum_{j=i+1}^{|S_i|} conf_i \cdot conf_j \cdot diversity(s_i, s_j) \qquad (5)$$

where $S_i = (s_1, s_2, \ldots, s_n)$ is the set of servers hosting an instance of the virtual node i and $conf_i$, $conf_j \in [0,\ 1]$ are the confidence levels of servers i, j. The diversity function returns a number computed using the geographical distance among server pairs. This distance is represented as a 6 bits number, each bit corresponding to the location parts of a server, namely continent, country, data center, room rack and server. The most significant bit (leftmost) represents the continent while the least significant bit (rightmost) represents the server. Note that more bits should be used to describe geographical distance in case of more geographical attributes. Starting with the most significant bit, each location part of both servers are compared one by one to compute their *similarity*: if the location parts are equivalent, the corresponding bit is set to 1, otherwise 0. Once a bit has been set to 0, all less significant bits are also set to 0. For example, two servers belonging to the same data center but located in different rooms cannot be in the same rack, thereby all bits after the third bit (data center) have to be 0. The similarity number would then look like this:

cont	coun	data	room	rack	serv
1	1	1	0	0	0

A binary "NOT" operation is then applied to the similarity to get the diversity value:

$$\overline{111000} = 000111 = 7(decimal)$$

The diversity between servers are summed because having $n + 1$ servers in the same location part is always better in terms of availability than having only n, as individual servers may fail as well.

When the availability is below th, a virtual node will replicate its data to a new server. The best candidate server is selected so as to maximize the net benefit between diversity of the resulting set of replica locations for the virtual node and the virtual rent of the new server. Specifically, a virtual node i with current replica locations in S_i maximizes the formula below:

$$\arg\max_j \sum_{k=1}^{|S_i|} conf_j \cdot diversity(s_k, s_j) - pr_j\ , \qquad (6)$$

where pr_j is the virtual rent price of candidate server j. The constant th allows a fine-grained control over the replication process. A low value means that a partition will be replicated on few servers potentially geographically close, whereas a higher value enforces many replicas to be located at dispersed locations. However, setting a high value for the minimum confidence level in a network with a few servers can result in an undesirable situation, where all partitions are replicated everywhere. To circumvent this, a maximum number of replicas per virtual node are allowed.

4.4 Virtual node decision tree

As already mentioned, a virtual node may decide to replicate, migrate, suicide or do nothing with its data at the end of an epoch. Upon replication, a new virtual node is associated with the replicated data. The decision tree of a virtual node is depicted in Figure 1. First, it verifies that the current availability of its partition is greater than th. If the minimum acceptable availability is not reached, the agent replicates its data to the server that maximizes *net benefit*, as described in Subsection 4.3.

If the availability is satisfactory, the agent tries to minimize costs. During an epoch, virtual nodes receive queries, process them and send the replies back to the client. Each query creates a *value* for the virtual node, which can be assumed to proportional to the size of the query reply. For this purpose, it employs a balance b defined as follows:

$$b = popularity - virtual\_rent\ , \qquad (7)$$

where *popularity* is the query load normalized in monetary units. To this end, a virtual node decides to:

- *migrate or suicide:* if it has negative balance for the last f epochs. First, the agent calculates the availability of its partition without its own replica. If satisfied, it suicides, i.e. deletes its replica. Otherwise, the agent tries to find a cheaper server that could host the replica of the partition. To avoid that a data replica oscillates between servers the migration is only allowed if the following *migration conditions* apply:
  - the minimum availability is still satisfied using the new server,
  - the absolute price difference between the current and the new server is greater than a threshold,
  - the current server storage usage is above a storage soft limit, typically 70% of the hard drive capacity, and the new server is below this limit.
- *replicate:* if it has positive balance for the last f epochs, it may replicate. For replication, a virtual node agent also has to verify that:
  - It can afford the replication by having a positive balance b' for consecutive f epochs:

    $$b' = popularity - c_n - 1.2 \cdot virtual\_rent'$$

    where c_n is a term representing the consistency (i.e. network) cost, which can be approximated as the number of replicas of the partition times a fixed average communication cost parameter for conflict resolution and routing table maintenance. $virtual\_rent'$ is the current virtual rent of the candidate server for replication, while the factor 1.2 accounts for the upper bound 20% increase at this rent price at the next epoch due to the potentially increased used storage of the candidate server.
  - the average bandwidth consumption bdw_r for the answering queries per replica after replication (left term of left side of inequality (8)) plus the size of the partition ps is less than the respective bandwidth bdw per replica without replication (right side of inequality (8)) for a fixed number win of epochs to compensate for steep changes of the query rate. A large win value should be used

for bursty query load. That is, replicate if:

$$\frac{win * q * q_s}{|S_i| + 1} + p_s < \frac{win * q * q_s}{|S_i|}\ , \qquad (8)$$

where q is the average number of queries for the last win epochs, q_s is the average size of the replies, $|S_i|$ is the number of servers currently hosting replicas of partition i and p_s is the size of the partition.

The virtual node agent finally sets lowest *popularity* to the current lowest virtual rent price to prevent unpopular nodes from migrating indefinitely.

A virtual node getting a large number of queries or having to process large queries will become wealthier. At the same time, the load of the corresponding server will increase, as well as the virtual rent price for the next epoch. Popular virtual nodes on the server will have enough "money" to pay the growing rent price, as opposed to unpopular ones that will have to move to a cheaper server. The transfer of unpopular virtual nodes will in turn decrease the virtual rent price, hence stabilizing the rent price of the server. This approach is self-adaptive and matches the query load by replicating popular virtual nodes.

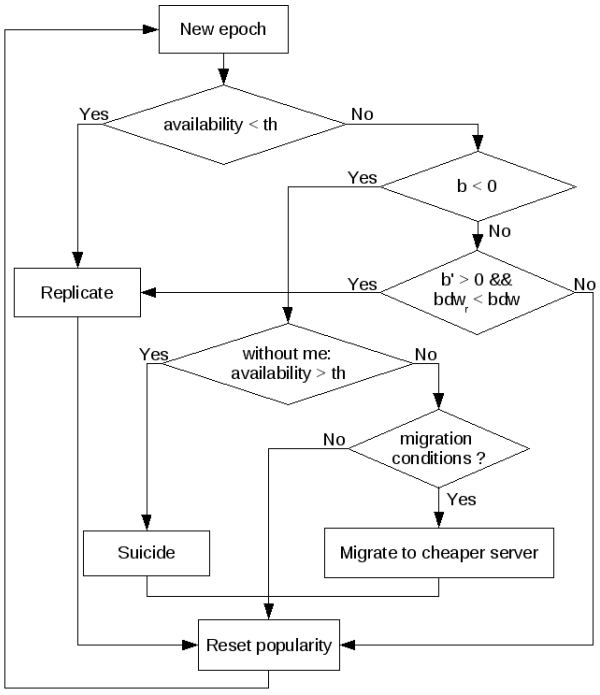

Figure 1: Decision tree of the virtual node agent.

5. EXPERIMENTAL RESULTS

5.1 The simulation model

We assume a simulated cloud storage environment consisting of N servers geographically distributed according to different scenarios that are explained on a per case basis. Data per application is assumed to be split into M partitions each managed by a virtual node. Each server reserves fixed bandwidth capacities for replication and migration per epoch. It also has a bandwidth capacity for serving queries and a fixed storage capacity. The popularity of the virtual nodes (i.e. the query rate) is distributed according to Pareto(1, 50). The number of queries per epoch is distributed according to Poisson with a mean rate λ, which is different per experimental setting. For simplicity, the size of every data partition is assumed to be fixed and equal to 256MB. Time is assumed to be slotted into epochs. At each epoch, virtual nodes employ the decision making algorithm of Subsection 4.4. Each server keeps track of its available bandwidth for migration, replication or answering queries, and its available storage after any data transfer that is decided to happen within one epoch. All data migrations and replications are considered to be completed when decided, as long as the necessary resources suffice. The virtual price per server is determined according to formula (4) at the beginning of each epoch. The average real rent of a server is considered fixed but with different values per experimental setting. Also, we assume that the environment is trustworthy and therefore the confidence assigned to all servers is 1. The simulation is written in Java.

5.2 Convergence to optimal solution

First, we consider a *small scale* setting to be able to validate our results solving numerically the optimization problem of Section 2. Specifically, we consider a data cloud consisting of $N = 5$ servers dispersed in Europe: two servers are hosted in Switzerland in separate data centers, one in France and two servers are hosted in Germany in the same rack of the same data center. Data is split into $M = 50$ partitions that are shared among servers at startup. In the optimization problem, we assume that each server has probability to fail 0.3 and that the failure probabilities of the first three servers are independent, while those of the Germanian data centers that are located at the same rack are correlated, so that $Pr[F4|F5] = Pr[F5|F4] = 0.5$. The minimum availability level th is chosen so as to ensure that a partition is hosted by at least 2 servers located at different data centers, i.e.: $\overline{001000}$=7 for the simulation model and 90% for the optimization problem. Network-related operational costs are considered dominant factor for the communication cost and thus distance of servers is not taken into account in decision making, i.e. $c_n = 0$ in both the simulation model and the optimization problem. Also, for equivalence to decision process of the simulation model, the operational cost c of each server in formula (3) of the optimization problem is divided by the popularity of the virtual node. The detailed parameters of the small scale setting of the simulation experiments are shown at the left column of Table 2.

As depicted in Figure 2, the virtual nodes replicate and migrate to other servers according to the approach of Subsection 4.4 and the system soon reaches equilibrium. At equilibrium, almost no migrations, replications or suicides of virtual nodes happen and therefore the virtual and real rents remain constant for fixed mean query rate and fixed number of servers. The convergence process actually takes only about 8 epochs, which is very close to the communication bound for replication (i.e. total data size / replication bandwidth = 10GB / 1.5GB per epoch$\approx$ 6.6 epochs). Also, as revealed by comparing the numerical solution of the optimization problem of Section 2 with the one that is given by simulation experiments, the proposed distributed eco-

Table 2: Parameters of small-scale and large-scale experiments.

Parameter	Small scale	Large scale
Servers	5	200
Server storage	5 GB	15 GB
Server price	100$	100$ (70%), 150$ (30%)
Total data	10 GB	1 TB
Average size of an item	500 KB	500 KB
Partitions	50	10000
Queries per epoch	Poisson ($\lambda = 300$)	Poisson ($\lambda = 3000$)
Query key distribution	Pareto (1,50)	Pareto (1,50)
Storage soft limit	0.7	0.7
Win	20	20
Replication bandwidth	300 MB/epoch	300 MB/epoch
Migration bandwidth	100 MB/epoch	100 MB/epoch

nomic approach solves rather accurately the optimization problem. Specifically, the total numbers of virtual nodes were the same and the distributions of virtual nodes per server were similar.

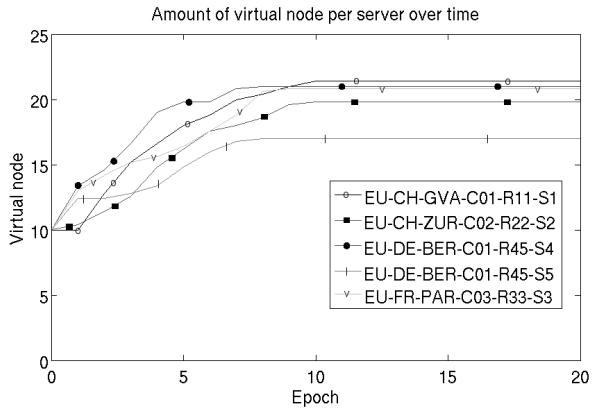

Figure 2: Small-scale setup : replication process at startup.

5.3 Server arrival and departure

Henceforth, we consider a more realistic *large-scale* setting of $M = 100000$ partitions and $N = 200$ servers with differentiated unit prices that belong to two categories: cheap and expensive. The detailed parameters of this experimental setting are shown at the right column of Table 2. At epoch 100, we assume that 30 new cheap servers are added to the data cloud, while 30 cheap servers are removed at epoch 200. As depicted in Figure 3, our approach is very robust to resource upgrading or failures: the total number of virtual nodes remains constant after adding resources to the data cloud and increases upon failures to maintain availability guarantees above the desired level.

5.4 Variations in the query load

Next, to show the adaptability of the store to the query load, a load peak similar to what would result with the slashdot effect (*http://en.wikipedia.org/wiki/Slashdot_effect*) is simulated: in a short period the mean query rate λ gets multi-

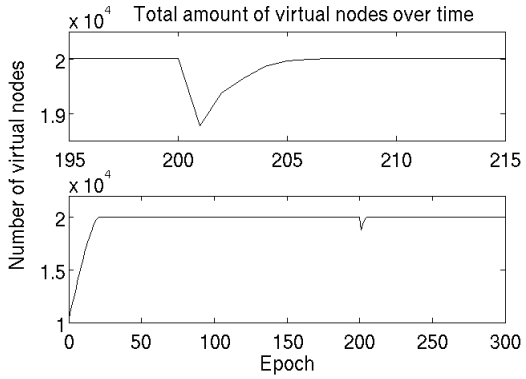

Figure 3: Large-scale setup : robustness against upgrades and failures.

plied by 30. Hence, the mean number of queries per epoch increases from 3000 to 90000 within 25 epochs and then slowly decreases during 250 epochs to the normal rate of 3000 queries per epoch. Then, at epoch 375 the same query load pattern is repeated: the mean query rate rapidly increases for 25 epochs and then it gradually decreases for 250 epochs. According to Pareto distribution, a small amount of virtual nodes attract a large amount of queries. These virtual nodes become "wealthier" thanks to their high popularity, and they are able to replicate to one or several servers in order to handle the increasing load. Therefore, the total amount of virtual nodes dynamically adjusts to the query load, as depicted in Figure 4. More importantly, the query load per server always remains quite balanced despite the variations in the total query load, as depicted in Figure 5. Specifically, the variance of the query load per server is minimal when the mean total query rate is steady and it has a decreasing trend when the system query load changes smoothly (e.g. between epochs 125 and 375). Also, as depicted in Figure 6, the average virtual rent price per server per epoch has a very small variance. The query load is well balanced among servers, as shown by the minimal impact of the load peaks to the average server virtual rent price in Figure 6. Therefore, our approach performs load balancing and utilizes the resources of the cloud in a cost-efficient way.

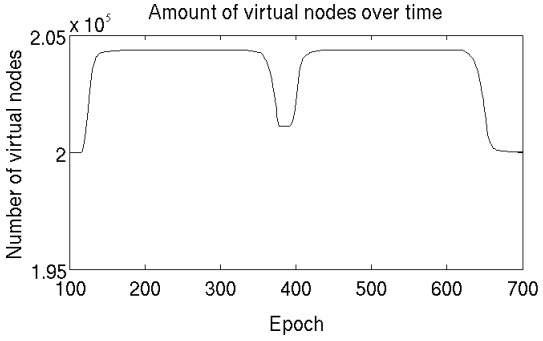

Figure 4: Large-scale setup : total amount of virtual nodes in the system over time.

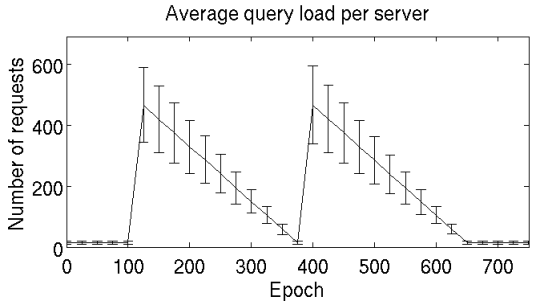

Figure 5: Large scale setup : query load per server over time.

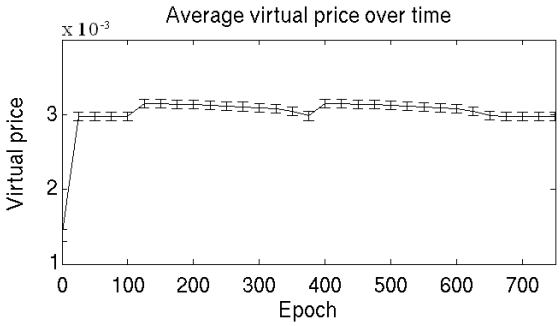

Figure 6: Large scale setup : average virtual rent per server per epoch (1 minute).

6. RELATED WORK

Dealing with network failure, strong consistency and high data availability can not be achieved at the same time, according to [2]. High data availability by means of replication has been investigated in various contexts, such as P2P systems [14, 9], data clouds, distributed databases [12, 4] and distributed file systems [6, 15, 1]. However, in all these systems, replication is employed in a static way, i.e. the number of replicas and their locations are predetermined. Also, no replication cost considerations are taken into account and no geographical diversity of replicas is employed. We have already mentioned that we share two common characteristics with Dynamo [4], namely zero-hop routing and the notion of virtual nodes. However, Dynamo deals with load balancing by assuming the uniform distribution of popular data items in data partitions. Instead, our approach deals with load balancing for dynamic changes of query load in a cost-effective way.

Some economic-aware approaches are dealing with the optimal locations of replicas. Mariposa [17] aims at latency minimization in executing complex queries over relational distributed databases, i.e. not primary-key access queries on which we focus. Sites in Mariposa exchange data items (i.e. migrate or replicate them) based on their expected query rate and their processing cost. For the exchanges, combinatorial auctions are employed where winner determination is tricky, as opposed to our straightforward decision making approach. Also, in [16], a cost model is defined for the factors that affect data and application migration for minimizing latency in replying queries. Data is migrated towards the application or the application towards the data based on their respective costs that depends on various aspects, such as query load, replicas placement and network and storage availability. On the other hand, in the Mungi operating system [7], a commodity market of storage space has been proposed. Specifically, storage space is lent by storage servers to users and the rental prices increase as the available storage runs low, forcing users to release unneeded storage. This model is equivalent to that of dynamic pricing per volume in telecommunication networks increasing prices with the level of congestion, i.e. congestion pricing. However, this approach does not take into account the different query rates for the various data items and it does not have any availability objectives.

7. CONCLUSION

In this paper, we described Skute, a robust, scalable and highly-available key-value store that dynamically adapts to varying load or disasters determining the most cost-efficient positions of data replicas. As a future work, we intend to implement a fully working prototype and investigate several other aspects, such as dynamic server prices, untrustworthy servers, competitive virtual nodes belonging to multiple data owners and data integrity and privacy.

8. REFERENCES

[1] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. Farsite: federated, available, and reliable storage for an incompletely trusted environment. *ACM SIGOPS Operating Systems Review*, 36(SI):1–14, 2002.

[2] P. A. Bernstein and N. Goodman. An algorithm for concurrency control and recovery in replicated distributed databases. *ACM Transactions on Database Systems*, 9(4):596–615, 1984.

[3] M. Dahlin, B. B. V. Chandra, L. Gao, and A. Nayate. End-to-end wan service availability. *IEEE/ACM Transactions on Networking*, 11(2):300–313, 2003.

[4] G. Decandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *Proc. of ACM Symposium on Operating Systems Principles*, New York, NY, USA, 2007.

[5] R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *Proc. of ACM SIGCOMM*, Karlsruhe, Germany, August 2003.

[6] R. G. Guy, J. S. Heidemann, and J. T. W. Page. The ficus replicated file system. *ACM SIGOPS Operating Systems Review*, 26(2):26, 1992.

[7] G. Heiser, F. Lam, and S. Russell. Resource management in the mungi single-address-space operating system. In *Proc. of Australasian Computer Science Conference*, Perth, Australia, February 1998.

[8] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proc. of*

ACM Symposium on Theory of Computing, pages 654–663, May 1997.

[9] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: an architecture for global-scale persistent storage. *SIGPLAN Not.*, 35(11):190–201, 2000.

[10] L. Lamport. Processor membership in asynchronous distributed systems. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

[11] L. Moser, P. Melliar-Smith, and V. Agrawala. Processor membership in asynchronous distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 5(5):459–473, 1994.

[12] K. Petersen, M. Spreitzer, D. Terry, and M. Theimer. Bayou: replicated database services for world-wide applications. In *Proc. of the 7th workshop on ACM SIGOPS European workshop*, Connemara, Ireland, 1996.

[13] E. Pinheiro, W.-D. Weber, and L. A. Barroso. Failure trends in a large disk drive population. In *Proc. of 5th USENIX Conference on File and Storage Technologies (FAST '07)*, San Jose, CA, USA, February 2007.

[14] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proc. of ACM Symposium on Operating Systems Principles*, Banff, Alberta, Canada, 2001.

[15] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: a highly available file system for a distributed workstation environment. *Transactions on Computers*, 39(4):447–459, 1990.

[16] H. Stockinger, K. Stockinger, E. Schikuta, and I. Willers. Towards a cost model for distributed and replicated data stores. In *Proc. of Euromicro Workshop on Parallel and Distributed Processing*, Italy, February 2001.

[17] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. An economic paradigm for query processing and data migration in mariposa. In *Proc. of Parallel and Distributed Information Systems*, Austin, TX, USA, September 1994.