# A Self-Organized, Fault-Tolerant and Scalable Replication Scheme for Cloud Storage

Author: Nicolas Bonvin, Thanasis G. Papaioannou and Karl Aberer

Presenter: Haiyang Xu

4, Jan, 2011 - Cloud Computing Seminar

# Agenda

# Introduction

- Background
  - Cloud storage is becoming a popular business paradigm
    - Amazon S3
    - ElephantDrive
    - Gigaspaces
  - Small companies rent distributed storage and pay per use

# Introduction

- Data availability can be affected in many ways
  - Hardware failures
  - Geographic proximity
  - Natural disasters
  - Highly irregular query rates
  - An application may become temporarily unavailable*

* http://en.wikipedia.org/wiki/Slashdot_effect

# Introduction

- Therefore
  - the support of service level agreements (SLAs) with data availability guarantees in cloud storage is very important
  - In reality, different applications may have different availability requirements
  - Fault-tolerance is commonly dealt with by replication

# Introduction

- Distributed key-value store
  - Widely employed
    amazon.com LinkedIn last.fm
  - Widely researched (by research communities)
    - Peer-to-peer
    - Scalable distributed data structures
    - Databases

# Introduction

- In this paper, the authors propose a scattered key-value store (Skute),
  - Skutes provides high and differentiated data availability statistical guarantees to multiple applications in a cost-efficient way in terms of rent price and query response times

# Introduction

- Skute combines the following innovative characteristics:
  - Computational
  - Differentiated availability statistical guarantees
  - Distributed economic model
  - Efficiently and fairly utilizing cloud resources
- A game-theoretic model is employed

# Skute: Scattered Key-Value Store

- Skute is designed to
  - provide low response time on read and write operations
  - ensure replicas' geographical dispersion in a cost- efficient way
  - offer differentiated availability guarantees per data item to multiple applications

# Skute: Scattered Key-Value Store

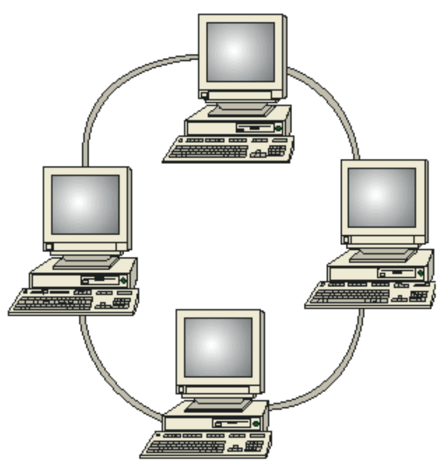
- Skute is divided into these parts
  - Physical node
  - Virtual node
  - Virtual ring
  - Routing

# Skute: Scattered Key-Value Store

- Physical node
  - A physical node (i.e. a server) belongs to a rack, a room, a data center, a country and a continent.
  - A label of the form "continent-country-datacenter-room- rack-server"
  - A server located in a data center in Berlin could be "EU-DE-BE1-C12-R07-S34"

# Skute: Scattered Key-Value Store

- Virtual node
  - ring topology
  - consistent hashing
  - Data is identified by a key
    - A one-way cryptographic hash function, e.g. MD5
  - The key space is split into partitions



# Skute: Scattered Key-Value Store

- Virtual node (cont'd)
  - A physical node (i.e. a server) gets assigned to multiple points in the ring, called tokens
  - A virtual node (alternatively a partition) holds data for the range of keys in (*previous token*, *token*]
  - A virtual node may replicate or migrate its data to another server, or suicide (i.e. delete its data replica)
  - A physical node hosts a varying amount of virtual nodes depending on the query load

Agenda

I

II

III

IV

V

VI

VII

VIII

# Skute: Scattered Key-Value Store

- Virtual ring
  - Skute allows multiple applications to share the same cloud infrastructure
  - Each application uses its own virtual rings, while one ring per availability level is needed, as depicted in Figure 1

# Skute: Scattered Key-Value Store

- Virtual ring

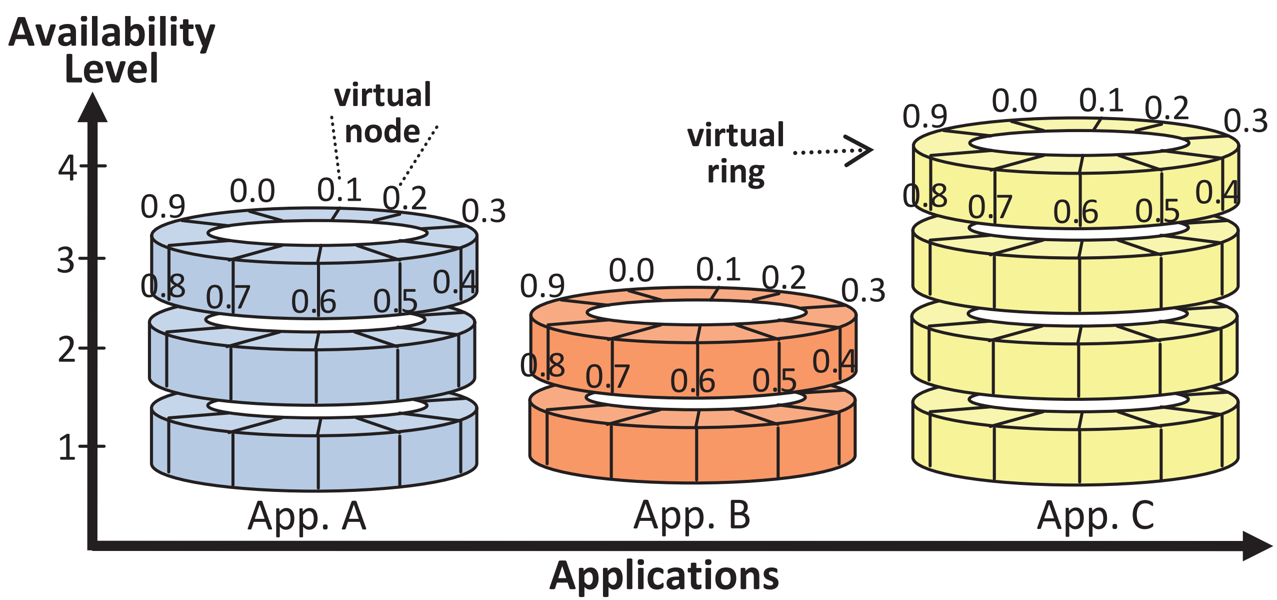



Figure 1: Three applications with different availability levels.

# Skute: Scattered Key-Value Store

- Virtual ring
  - Multiple data availability levels per application
  - Geographical data placement per application

# Skute: Scattered Key-Value Store

- Routing
  - Skute is intended to be used with real-time applications
  - Routing has to be efficient
  - Each virtual ring has its own routing entries, resulting in potentially large routing tables
  - The number of entries in the routing table is:

$$entries = \sum_{i}^{apps} \sum_{j}^{levels_i} partition(i, j) \qquad (1)$$

# Skute: Scattered Key-Value Store

- Routing
  - A physical node is responsible to manage the routing table of all virtual rings hosted in it, in order to minimize the update costs.
  - The routing table is periodically updated using a gossiping protocol

# Problem Definition

- The data belonging to an application is split into $M$ partitions, where each partition $i$ has $r_i$ distributed replicas. We assume that $N$ servers are present in the data cloud

Maximize data availability → Minimize communica tion cost → Maximize net benefit

# Problem Definition

- Maximize data availability
  - Placing replicas of a partition in a set of different servers
  - Data availability generally increases with the geographical diversity of the selected servers.
  - The worst solution
    - Put all replicas at a server with equal or worse probability of failure than others

# Problem Definition

- Maximize data availability
  - Probability a partition *i* to be unavailable:

$$
\begin{aligned}
&P_r(i\ unavailable) = P_r\left(F_1 \cap F_2 \cap \cdots \cap F_{|S_i^d|}\right) \\
&= \prod_{j=1}^{k} P_r(F_j) \cdot P_r\left(F_k \middle| F_{k+1} \cdots \cap F_{|S_i^d|}\right) \\
&\cdot P_r\left(F_{k+1} \middle| F_{k+2} \cap \cdots \cap F_{|S_i^d|}\right) \cdot \cdots \cdot P_r\left(F_{|S_i^d|}\right), \\
&if\ F_{k+1} \cap F_{k+1} \cap \cdots F_{|S_i^d|} \neq \emptyset
\end{aligned}
\tag{2}
$$

# Problem Definition

- Minimize communication cost
  - Save bandwidth during migration and replication
  - Reduce latency

# Problem Definition

- Minimize communication cost
  - $L^d$: $M \times N$ location matrix of application $d$

$$\begin{bmatrix} L_{11} & \cdots & \cdots & L_{1N} \\ \vdots & \ddots & & \vdots \\ \vdots & L_{ij} & \ddots & \vdots \\ L_{M1} & \cdots & \cdots & L_{MN} \end{bmatrix}$$

  - $L_{ij} = 0$ if application $i$ has a replica on server $j$

# Problem Definition

- Minimize communication cost
  - Network cost $c_n$ can be given by:

$$c_n\left(\overrightarrow{L_i^d}\right) = sum(\overrightarrow{L_i^d} \cdot \overrightarrow{\overrightarrow{NC}} \cdot \overrightarrow{L_i^d}^T) \qquad (3)$$

  - $NC$ is a strictly upper triangular $N \times N$ whose element $NC_{jk}$ is the communication cost between servers $j$ and $k$
  - $sum$ denotes the sum of matrix elements

# Problem Definition

- Minimize communication cost
  - Network cost (more clearly)

$$c_n\left(\overrightarrow{L_i^d}\right) = sum\left(\begin{bmatrix} L_{i1}^d & \cdots & L_{iN}^d \end{bmatrix} \cdot \begin{bmatrix} 0 & \cdots & NC_{1,N-1} & NC_{1,N} \\ \vdots & 0 & \ddots & \vdots \\ \vdots & \ddots & 0 & NC_{N-1,N} \\ 0 & \cdots & \cdots & 0 \end{bmatrix} \cdot \begin{bmatrix} L_{i1}^d \\ \vdots \\ \vdots \\ L_{iN}^d \end{bmatrix}\right)$$

# Problem Definition

- Maximize net benefit
  - The data owner wants to

Minimize his expenses by replacing expensive servers with cheaper ones

↔

Maintaining a certain minimum data availability promised by SLAs to his clients

# Problem Definition

- Maximize net benefit
  - Overall, he seeks to maximize his net benefit and the global optimization problem can be formulated as follows:

$$max\left\{u(pop_i, G) - \overrightarrow{L_i^d}\vec{c}^T + c_n\left(\overrightarrow{L_i^d}\right)\right\}, \forall i, \forall d$$

s.t. (4)

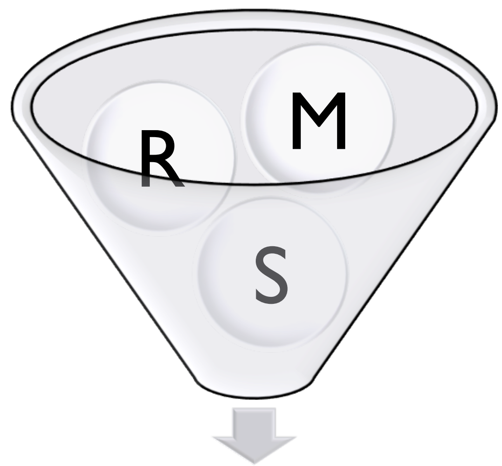
$$1 - P_r\left(F_1^{L_{i1}^d} \cap F_1^{L_{i2}^d} \cap \cdots F_1^{L_{iN}^d}\right) \geq th_d$$

# Individual Optimization

- Keep data availability above a certain minimum level required by the application
- Minimizing the associated costs
- Time is split into epochs
- The virtual rent of each server is announced at a board and is updated at the beginning of a new epoch.

# Individual Optimization

- A virtual node may replicate or migrate its data to another server, or suicide at each epoch and pay a virtual rent
- ***NO*** global coordination and each virtual node behaves ***independently***



# Individual Optimization

- Board
  - At each epoch, the virtual nodes need to know the virtual rent price of the servers
  - One server in the network is elected to store the current virtual rent per epoch of each server
    - i) it assumes trustworthiness of the elected server
    - ii) the elected server may become a bottleneck.

# Individual Optimization

- Board
  - Another approach
    - Each server maintains its own local board
    - Periodically updates the virtual prices of a ran- dom subset (log(N)) of servers by contacting them directly
    - Does not have the aforementioned problems
    - Decision may be based on outdated information
    - Verified with low communication overhead

# Individual Optimization

- Board
  - Confidence value
    - Stored at board(s) after new server added
    - Based on servers' offered availability and performance

# Individual Optimization

- Physical node
  - The virtual rent price $c$ of a physical node for the next epoch can be given by:

$$c = up \cdot (storage_{usage} + query_{load}), \quad (5)$$

- $up$ is the marginal usage price of the server
- The query load and the storage usage at the current epoch are considered to be good approximations of the ones at the next epoch
- an expensive server tends to be also expensive in the virtual economy

Agenda

I

II

III

IV

V

VI

VII

VIII

# Individual Optimization

- Maintaining availability

$$avail_i = \sum_{i=0}^{|S_i|} \sum_{j=i+1}^{|S_i|} conf_i \cdot conf_j \cdot diversity(s_i, s_j) \quad (6)$$

  - Where $S_i=(s_1,\ldots,s_n)$ is the set of servers hosting replicas of the virtual node $i$ and $conf_i$, $conf_j$ are the confidence levels of servers $i$, $j$. The diversity function returns a number calculated based on the geographical distance among each server pairs

Agenda I II III IV V VI VII VIII

# Individual Optimization

- Maintaining availability
  - Distance representation

| Cont | Coun | Data | Room | Rack | serv |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 |

- If the location parts are equivalent, the corresponding bit is set to 1, otherwise 0
- Diversity value (binary "NOT"):

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

  - Diversity values of server pairs are sum up
  - More replicas in distinct servers located in the same location ➔ increased availability

# Individual Optimization

- Maintaining availability
  - When the availability of a virtual node falls below *th*, it replicates its data to a new server
  - Specifically, a virtual node *i* with current replica locations in *Si* maximizes:

$$\max_j \sum_{k=1}^{|S_i|} g_j \cdot conf_j \cdot diversity(s_k, s_j) - c_j \qquad (7)$$

- $c_j$: virtual rent price of candidate of server j
- $g_j$: a weight related to the proximity of the server location to the geographical distribution of query clients for the partition of a virtual node

# Individual Optimization

- Maintaining availability
  - $g_j$ is given by:

$$g_j = \frac{\sum_l q_l}{1 + \sum_l q_l \cdot diversity(l, s_j)} \quad (8)$$

  - Where $q_l$ is the number of queries for the partition of the virtual node per client location $l$.

# Individual Optimization

- Virtual node decision tree
  - balance (i.e. net benefit) $b$ for a virtual node is defined as follows:

$$b = u(pop, g) - c, \qquad (9)$$

  - balance $b'$ for consecutive $f$ epochs:

$$b' = u(pop, g) - c_n - 1.2 \cdot c'$$

    - where $c_n$ is a term representing the consistency (i.e. network) cost
    - $c'$ is the current virtual rent of the candidate server for replication

# Individual Optimization

- Virtual node decision tree
  - Average bandwidth consumption

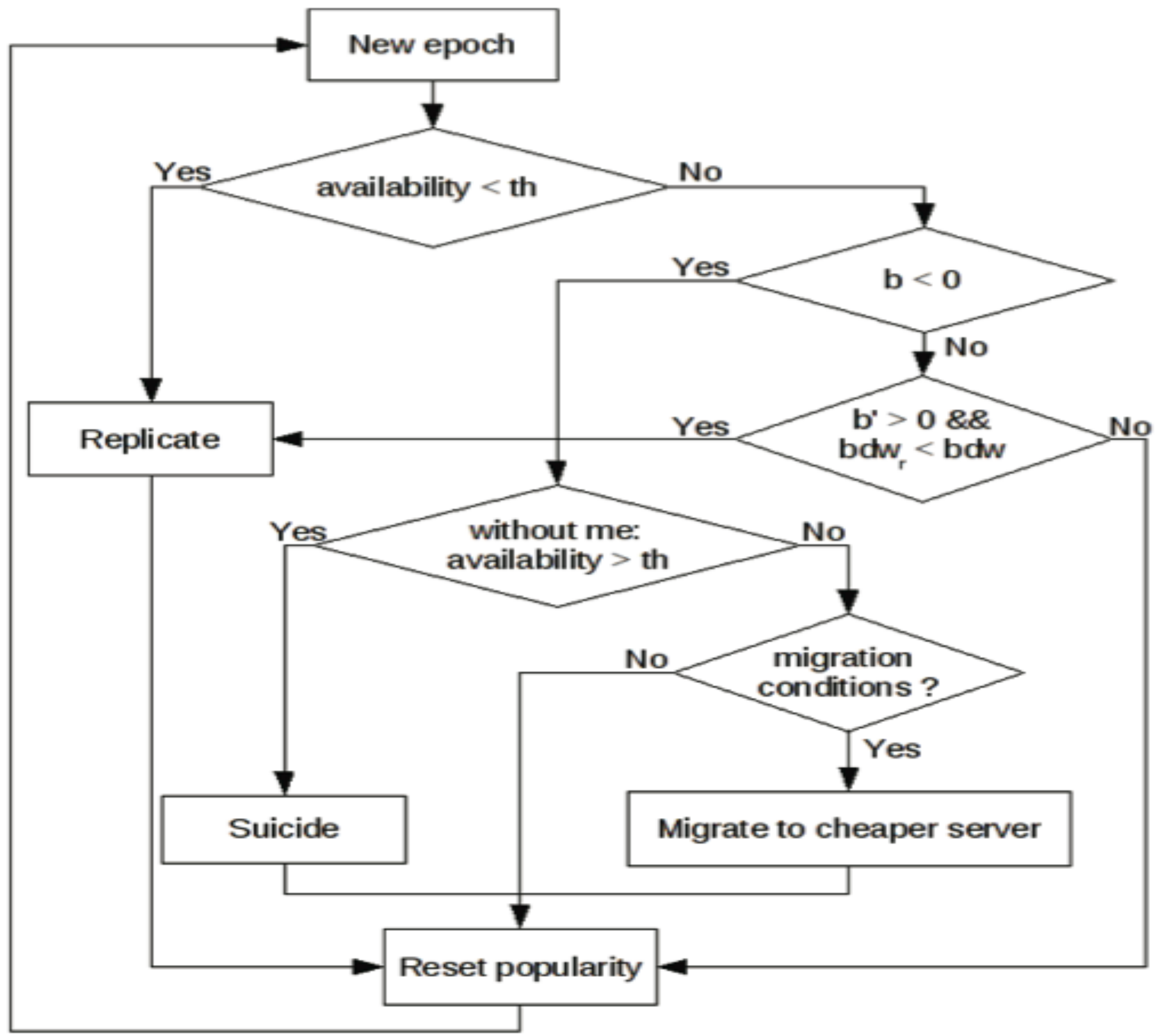
$$bdw_r = \frac{win * q * q_s}{|S_i| + 1} + p_s$$

  - Respective bandwidth per replica

$$bdw = \frac{win * q * q_s}{|S_i|}$$

  - 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.

# Individual Optimization



# Equilibrium Analysis

- Single round strategy payoffs at round $t+1$ are given by:

*Migrate:* $EV_M = \frac{u_i^{(t)}}{r_i^{(t)}} - f_c^i - f_d^i \cdot r_i^{(t)} - C_c^{(t+1)}$

*Replicate:*

$$EV_R = \frac{u_i^{(t)} + a_i^{(t)}}{r_i^{(t)} + 1} - f_c^i - f_d^i(r_i^{(t)} + 1) - \frac{1}{2}(C_c^{(t+1)} + C_e^{(t+1)})$$

*Suicide:* $EV_D = 0$

*Stay:* $EV_S = \frac{u_i^{(t)}}{r_i^{(t)}} - f_d^i r_i^{(t)} - C_e^{(t+1)}$

$C_c^{(t+1)}$: *price at round $t+1$ of the cheapest server at round $t$*
$C_e^{(t+1)}$: *price at round $t+1$ of the current hosting server at round $t$*

Agenda

I

II

III

IV

V

VI

VII

VIII

# Equilibrium Analysis

- If we assume probability of
  - Migrate: $x$; Replicate: $y$; Suicide: $z$; Stay: $1 - x - y - z$
  - then we calculate $c_c^{(t+1)}$, $c_e^{(t+1)}$ as follows:

$$C_c^{(t+1)} = C_c^{(t)}[1 + (x + y)\sum_{i=1}^{M} r_i^{(t)}] \quad (11)$$

$$C_e^{(t+1)} = C_e^{(t)}[1 - (x + z + \phi y)] \quad (12)$$

    - $0 < \emptyset \ll 1$ : : Recall that the total number of queries for a partition is divided by the total number of replicas of that partition and thus replication also reduces the rent price of the current server.

# Equilibrium Analysis

- The expected payoffs of these strategies should be equal at equilibrium, as the virtual node should be indifferent between them:

$$
\begin{aligned}
&EV_M = EV_S \Leftrightarrow \\
&\frac{u}{r} - f_c - f_d\, r - C_c(1 + x\, N_r) = \frac{u}{r} - f_d\, r - C_e(1 - x) \Leftrightarrow \\
&x = \frac{C_e - C_c - f_c}{C_e + C_c\, N_r}
\end{aligned}
\tag{13}
$$

# Equilibrium Analysis

- At equilibrium,
  - Rent of the current server used by a virtual node > rent of the cheapest server + cost of migration for this virtual node
  - The probability to migrate ***decrease*** with the total number of replicas in the system
  - The # of migration at equilibrium will be almost 0

# Rational Strategies

- The rational strategies that could be employed by servers in an untrustworthy environment
  - Eg. a server may overutilize its bandwidth resources by advertising a lower virtual price
- The aforementioned rational strategies could be tackled as:
  - The confidence value of a server could also reflect its trustworthiness for reporting its utilization correctly
  - Application providers should divide $c_j$ by the confidence $conf_j$ of the server $j$ in the maximization formula (7)

# Test Results in Simulated and Real Testbed

- Simulation Results
  - Test environment

| Parameter | Small scale | Larse scale |
|---|---|---|
| Servers | 5 | 200 |
| Server storate | 10 GB | 10 GB |
| Server price | 100$ | 100$ (70%), 125$ (30%) |
| Total data | 10 GB | 100 GB |
| Averate 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) |
| Storate soft limit | 0.7 | 0.7 |
| Win | 20 | 100 |
| Replication bandwidth | 300 MB/epoch | 300 MB/epoch |
| Mitration bandwidth | 100 MB/epoch | 100 MB/epoch |

# Test Results in Simulated and Real Testbed

- Simulation Results – small scale

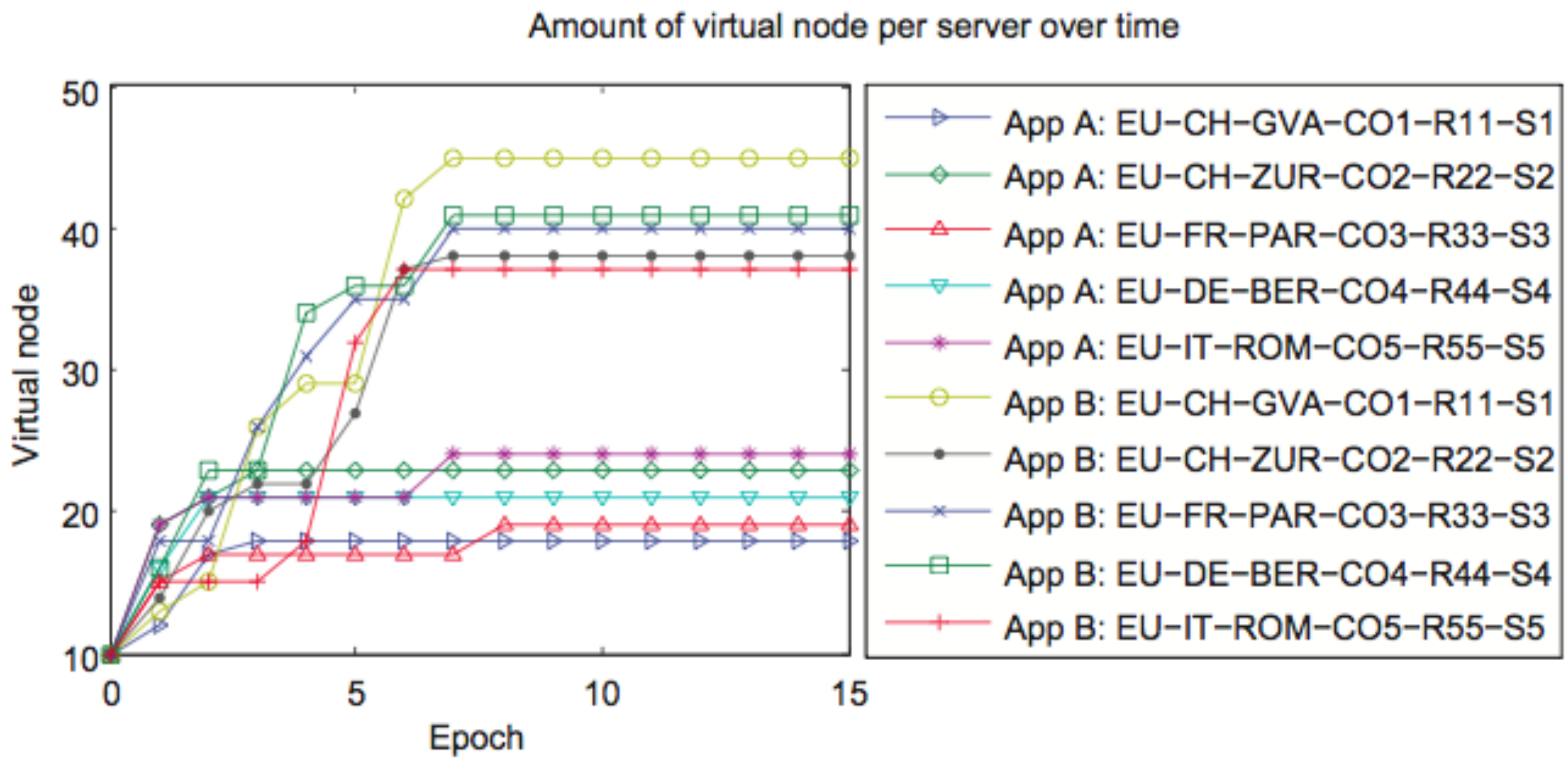



Figure 3: Small-scale scenario: replication process at startup

# Test Results in Simulated and Real Testbed

- Simulation Results – large scale

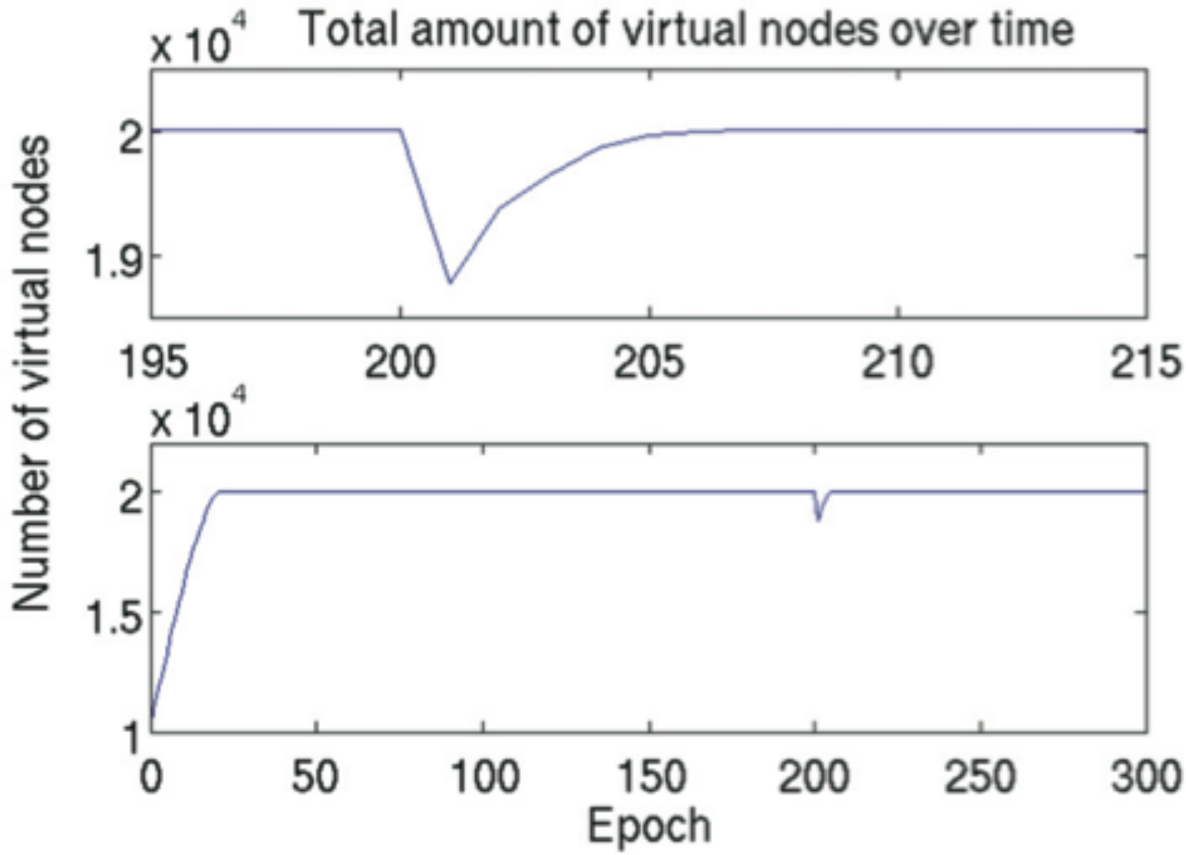



Figure 4: Large-scale scenario: robustness against upgrades and failures

# Test Results in Simulated and Real Testbed

- Adaptation to the query load
  - Simulate a load peak similar to what it would result with the "Slashdot effect": in a short

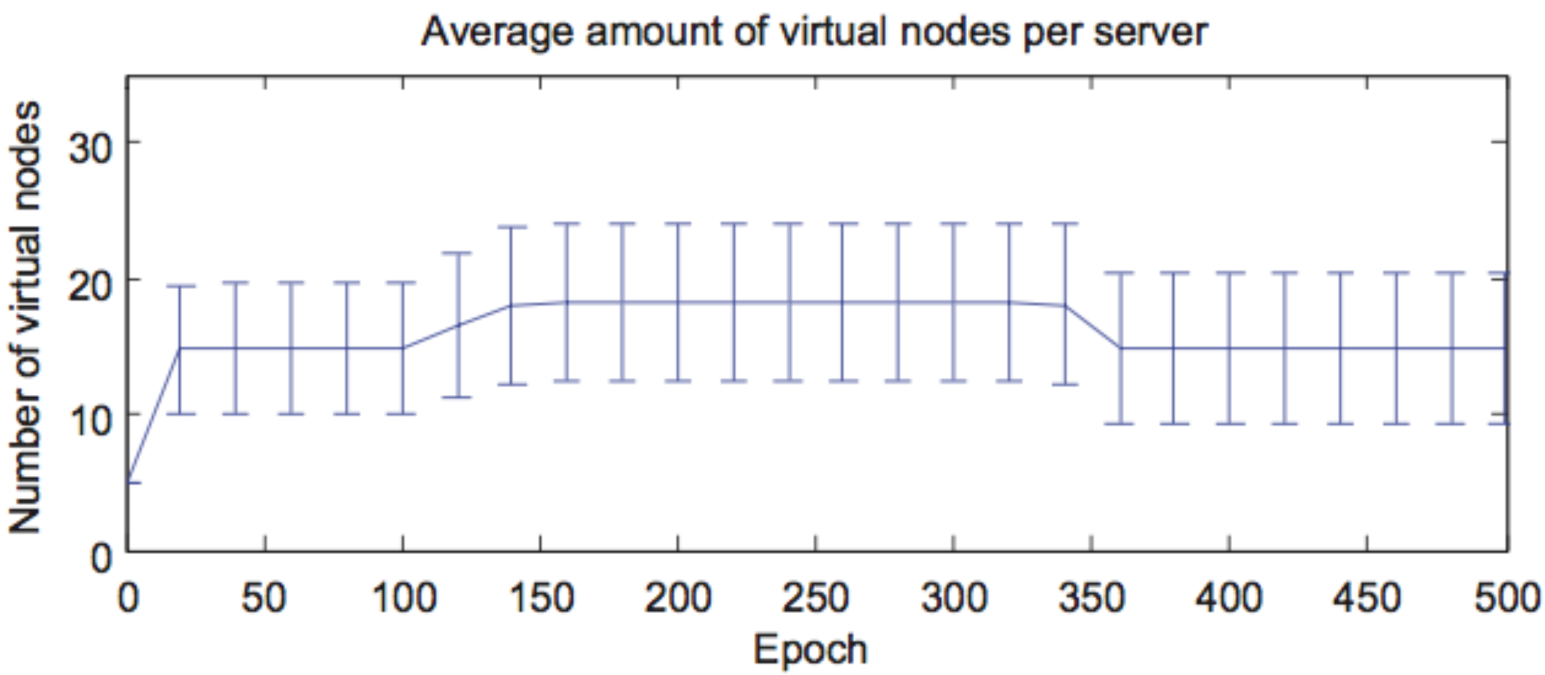



Figure 5: Large-scale scenario: total amount of virtual nodes in the system over time

# Test Results in Simulated and Real Testbed

- Adaptation to the query load

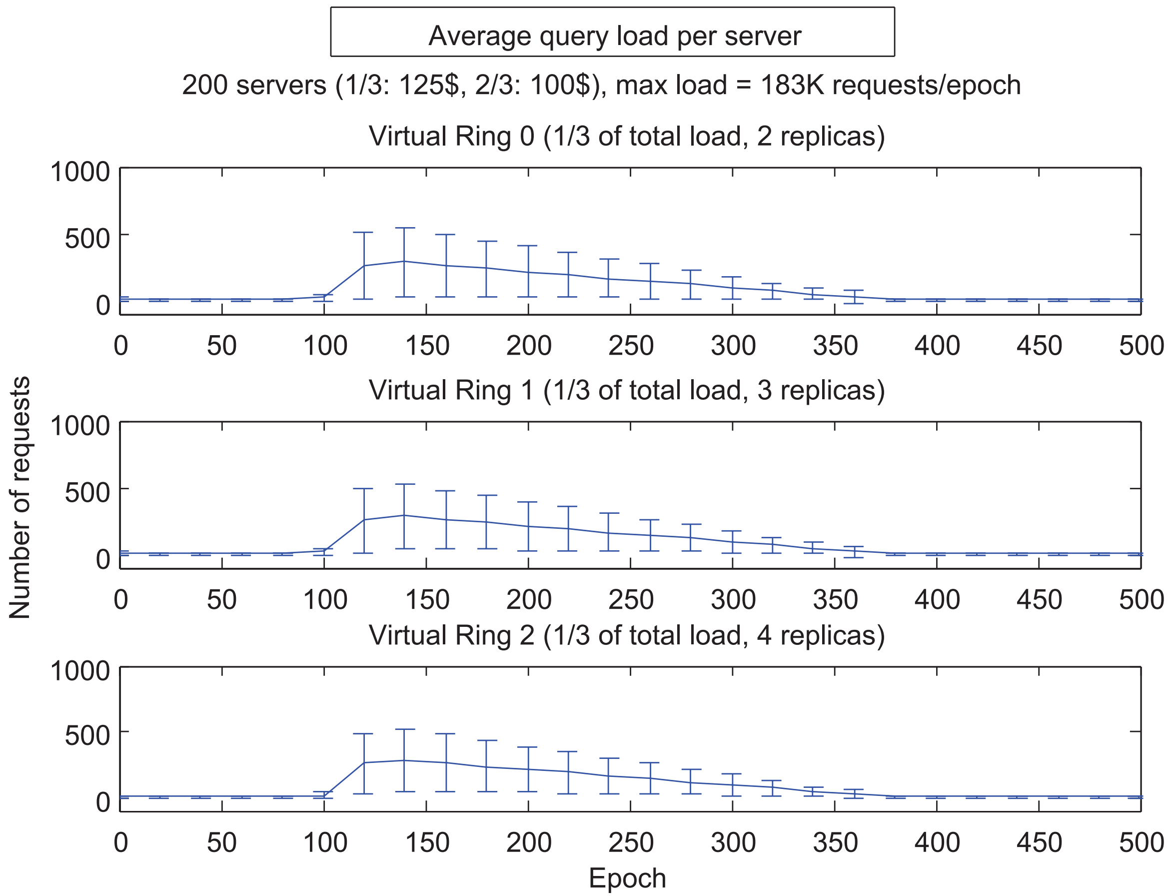



Figure 6: Large scale scenario: average query load per virtual ring per server over time with queries evenly distributed

# Test Results in Simulated and Real Testbed

- Scalability of the approach
  - The insert queries are distributed according to Pareto(1, 50)
  - Max. partition capacity of 256MB after which the data of the partition is split into two new ones ➔each virtual node is always responsible for up to 256MB of data
  - Fixed insert query rate = 2000 queries/epoch,
  - Each query inserts 500KB of data
  - Large-scale scenario parameters, but with 100 servers and 2 racks/room.
  - Initial number of partitions is M=200

# Test Results in Simulated and Real Testbed

- Scalability of the approach

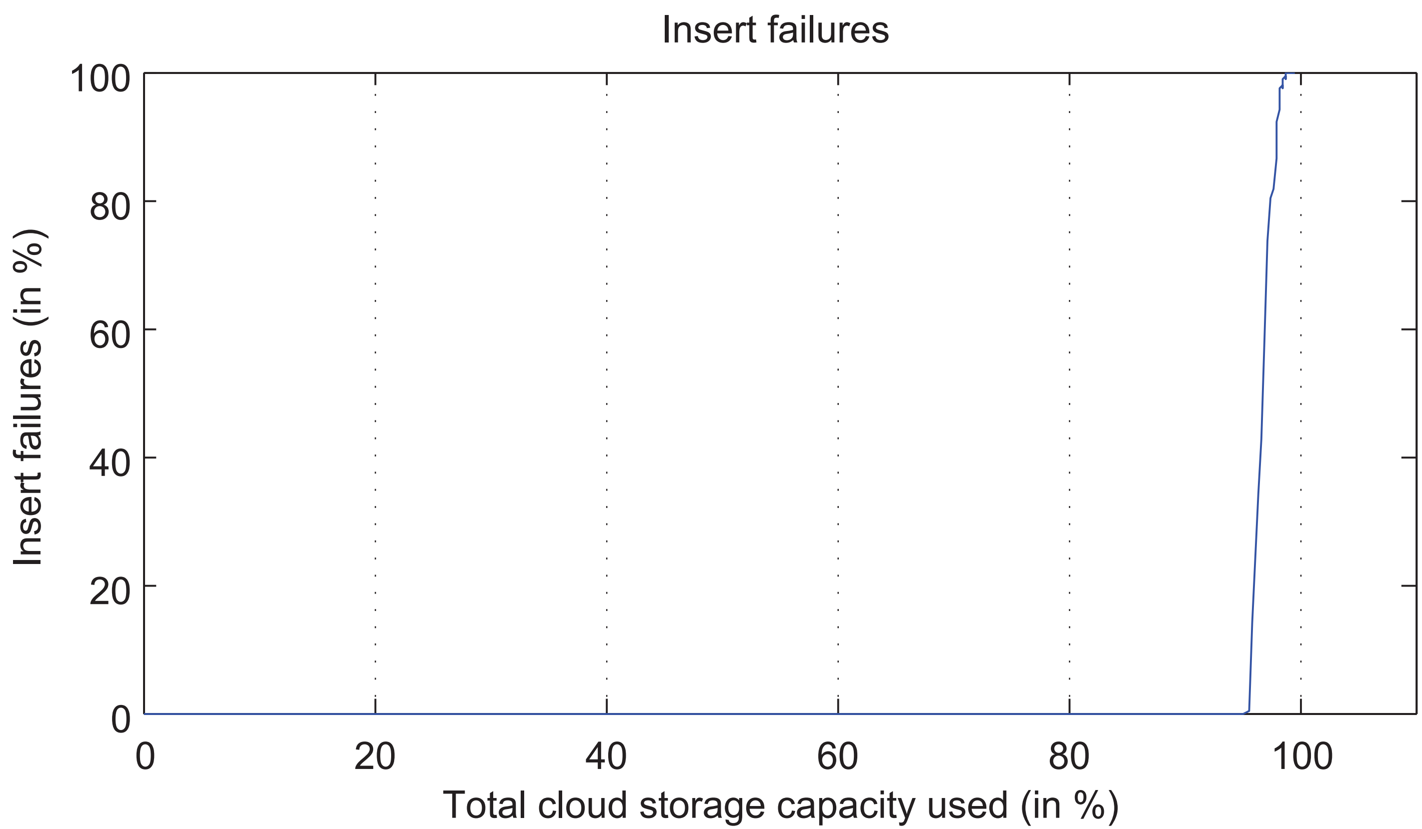



Figure 8: Storage saturation: insert failures

# Test Results in Simulated and Real Testbed

- Scalability of the approach
  - Now consider that the query rate is not distributed according to Poisson
  - It increases with the rate of 200 queries/epoch until reaching the total bandwidth capacity
  - In this experiment, real rents of servers are uniformly distributed in [1, 100]$
  - Our approach (referred as *Economic*) compared with other basic approaches: *Random* and *Greedy*

# Test Results in Simulated and Real Testbed

- Scalability of the approach

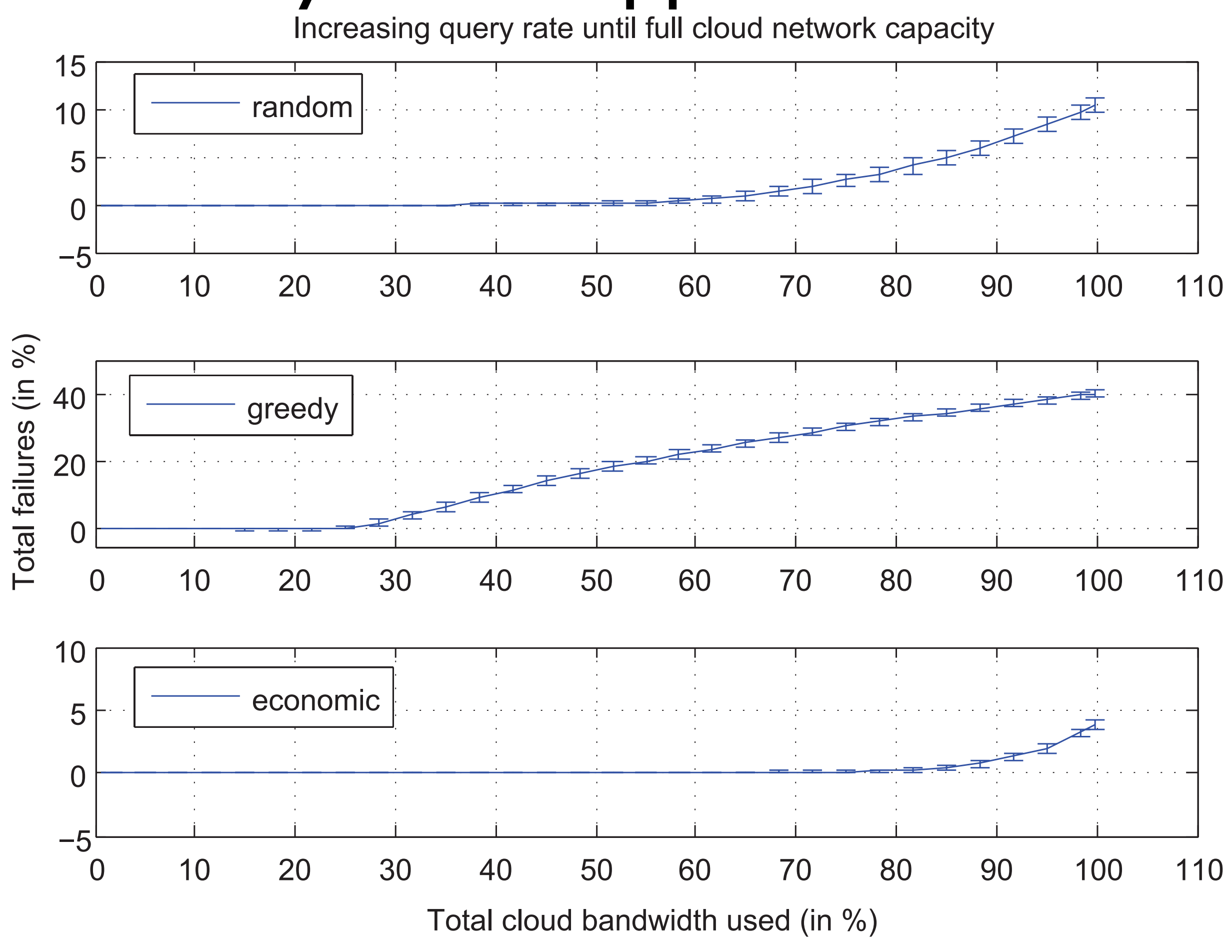



Figure 9: Network saturation: query failures

# Test Results in Simulated and Real Testbed

- Real Testbed
  - Servers are not synchronized
  - No centralized component is required
  - The epoch equals to 30 seconds
  - Fully decentralized board
  - Routing tables maintained using a gossiping protocol for routing entries
  - In case of migration, replication or suicide of a virtual node, the hosting server broadcasts the routing table update

Agenda

I

II

III

IV

V

VI

VII

VIII

# Test Results in Simulated and Real Testbed

- Real Testbed
  - 40 Skute servers
  - hosted by 8 machines
    - OS: Debian 5.0.3, Kernel: 2.6.26-2-amd64
    - CPU: 8 core Intel Xeon CPU E5430 @ 2.66GHz
    - RAM: 16GB
  - Sun Java 64-Bit VMs (build 1.6.0_12-b04)
  - 100 Mbps LAN

# Test Results in Simulated and Real Testbed

- Real Testbed Results

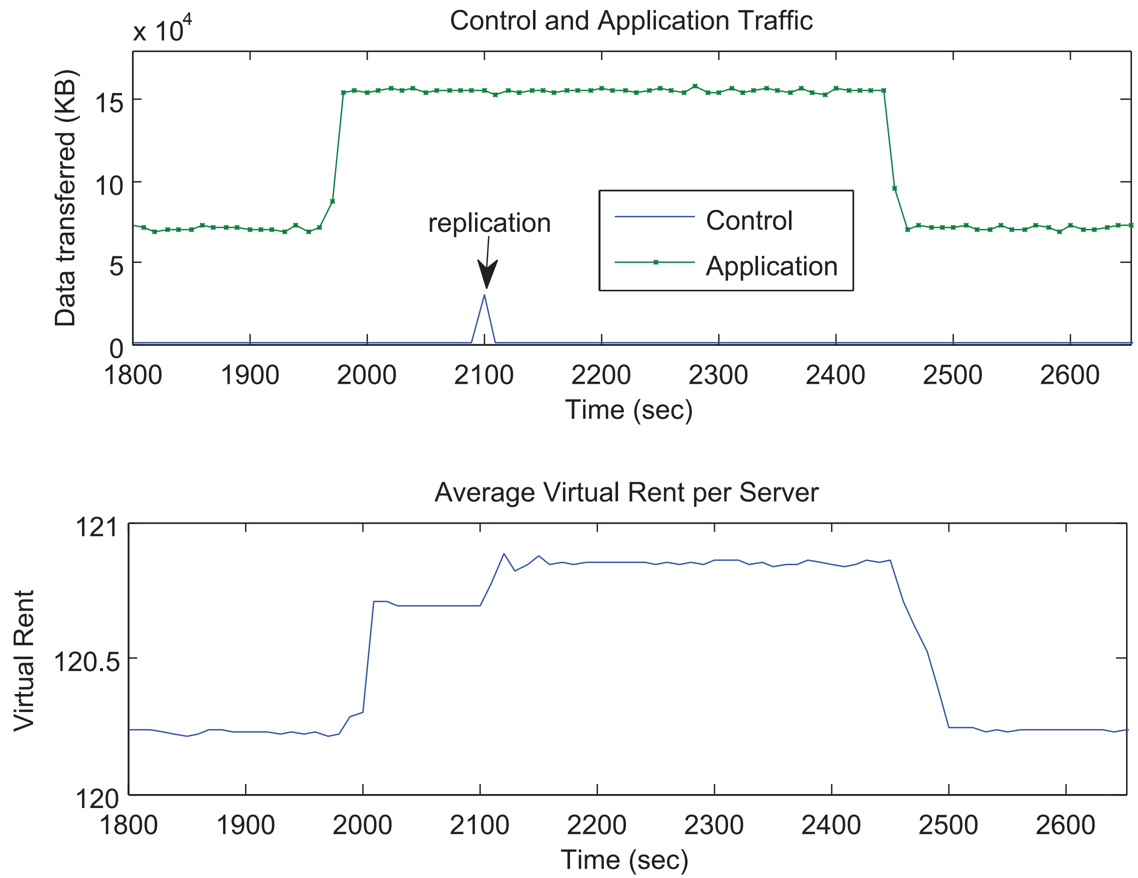



Figure 10: Top: Application and control traffic in case of a load peak. Bottom: Average virtual rent in case of a load peak.

# Test Results in Simulated and Real Testbed

- Real Testbed Results

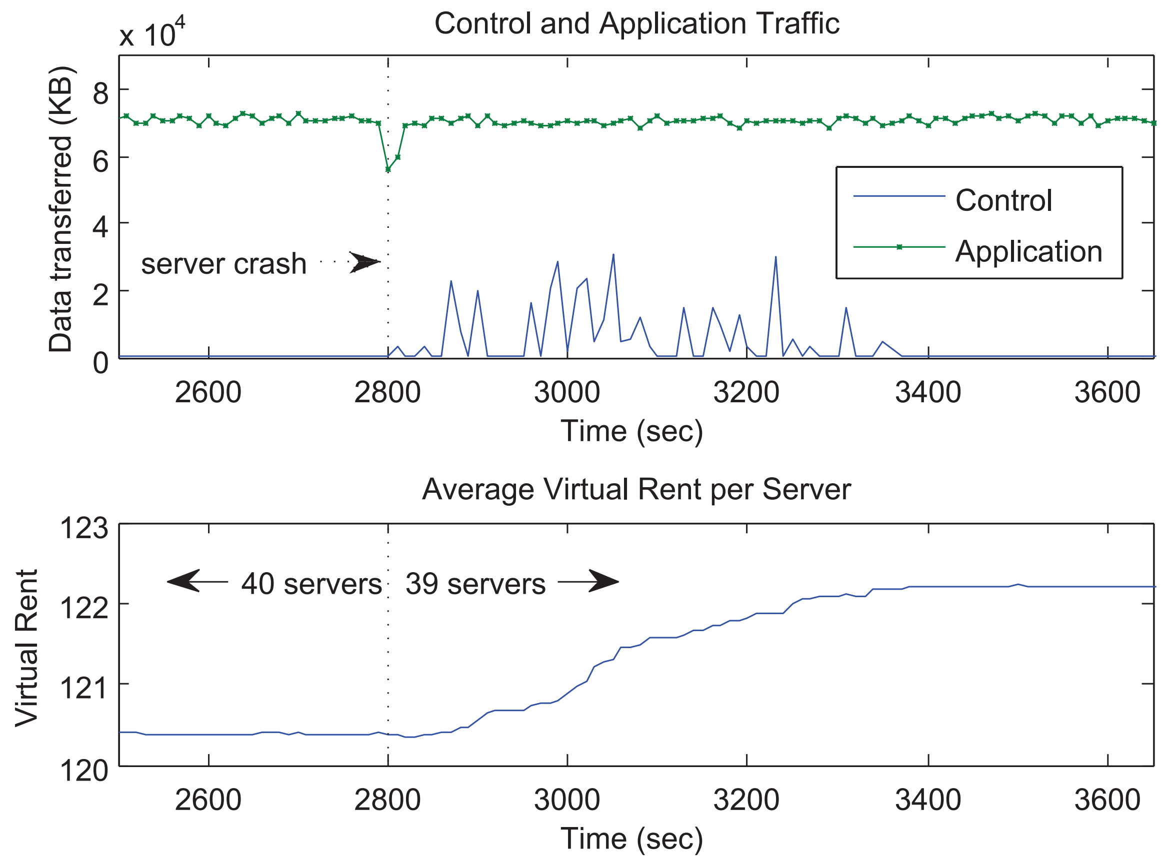



Figure 11: Top: Application and control traffic in case of a server crash. Bottom: Average virtual rent in case of a server crash.

# Conclusion and Future Work

- Conclusion
  - Skute - a robust, scalable and highly- available key-value store
    - dynamically adapts to varying query load or disasters
    - determining the most cost-efficient locations of data replicas with respect to their popularity and their client locations
  - Experimentally proved that our Skute converges fast to equilibrium
  - As predicted by a game-theoretical model no migrations happen for steady system conditions

# Conclusion and Future Work

- Future Work
  - Investigate the employment of Skute for more complex data models

# Question