

# Is Elasticity of Scalable Databases a Myth?

Daniel Seybold\*, Nicolas Wagner\*, Benjamin Erb† and Jörg Domaschka\*

\*Institute of Information Resource Management

†Institute of Distributed Systems

Ulm University, Ulm, Germany

Email: {daniel.seybold, nicolas.wagner, benjamin.erb, joerg.domaschka}@uni-ulm.de

**Abstract**—The age of cloud computing has introduced all the mechanisms needed to elastically scale distributed, cloud-enabled applications. At roughly the same time, NoSQL databases have been proclaimed as the scalable alternative to relational databases. Since then, NoSQL databases are a core component of many large-scale distributed applications.

This paper evaluates the scalability and elasticity features of the three widely used NoSQL database systems Couchbase, Cassandra and MongoDB under various workloads and settings using throughput and latency as metrics. The numbers show that the three database systems have dramatically different baselines with respect to both metrics and also behave unexpected when scaling out. For instance, while Couchbase’s throughput increases by 17% when scaled out from 1 to 4 nodes, MongoDB’s throughput decreases by more than 50%. These surprising results show that not all tested NoSQL databases do scale as expected and even worse, in some cases scaling harms performances.

## I. INTRODUCTION

The evolvement of cloud computing has gained tremendous focus in industry and academia, especially for web-based applications. With the typical benefits of cloud computing such as on-demand self-service, resource pooling or rapid elasticity [1] also traditional web service architectures experienced the rethinking from monolithic structures to distributed services. Whereas the distribution of the mostly stateless business logic services fits well for distribution in the cloud, the distribution of stateful database services is more challenging. Hence, in parallel to cloud computing, a distributable database class, the NoSQL databases, evolved and proclaim as alternative for traditional relational databases. NoSQL databases store data in non-relational way and promise to be scalable and to run on commodity hardware as offered by the cloud.

These developments in the database area also have influenced the service models of the initial cloud service models [1] Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) by extending them with more fine grained service models such as Database as a Service (DBaaS) [2]. Yet, the selection of the distributed database still remains as challenge.

Within the last years, a high number the existing of NoSQL databases reached a mature state and nearly all of them promise up to “unlimited” scalability by utilising cloud resources. In this context we provide an up-to-date evaluation of the common NoSQL databases, Apache Cassandra, Couchbase and MongoDB, with the focus on their scalability and elasticity capabilities for a DBaaS system.

Our contribution is twofold, (1) we define a thorough methodology to evaluate the scalability and elasticity of distributed databases; (2) we evaluate the mentioned NoSQL databases in a distributed setup with respect to their specific scalability and elasticity capabilities and discuss the results with respect to an expected behaviour.

The remainder of the paper is organised as follows. Section II describes the challenges of scalability and elasticity with respect to DBaaS. In Section III, we introduce the cloud architecture, distributed databases and distributed storage models. Section IV defines the methodology that we used for our evaluation. Section V describes the cloud environment of our evaluation. Section VI presents our evaluation results, followed by a discussion in Section VII. Section VIII discusses the related work. Finally, Section IX concludes and outlines future work.

## II. DATABASE CHALLENGES FOR DBAAS

In the following we define the terms *scalability* and *elasticity* for our considerations in the context of a DBaaS, with respect to their definitions in cloud computing and for distributed databases.

### A. Scalability

The term scalability is a common term in IT in general. As we focus in our work on the scalability of distributed databases in the cloud, we narrow down scalability with respect to cloud computing and database systems. In the cloud context a well-known definition is provided by Herbst et. al. [3] with “*Scalability is the ability of the system to sustain increasing workloads by making use of additional resources*”.

Database system workloads can be classified in *memory-bound* (i.e. the data fits into memory) and *storage-bound* (i.e. fractions of the data is kept in memory and the rest in persistent storage) [4]. In this paper, we only focus on the scaling of memory-bound workloads. A definition of scalability for distributed systems in general and with respect to distributed databases is provided by Agrawal et. al. [5], defining the terms scale-up, scale-out and scale-in in order to manage growing workloads. Scale up or vertically scaling applies by adding more computing resources to a single node. This paper only focuses on the two horizontal scaling actions, namely scale out (i.e. adding nodes to a cluster) and scale in (i.e. removing nodes from a cluster).

For applying a distributed database in a DBaaS a high scalability factor is required to process virtually unlimited workload sizes by scaling the database cluster to a sufficient size. A high scalability factor is represented by constant latency and proportionally growing throughput with respect to the number of nodes and the workload size [6].

### B. Elasticity

Scalability focuses on the general ability to process arbitrary workload sizes. Elasticity is tightly coupled to scalability and enables the overcoming of sudden workload fluctuations by scaling the cluster without any downtime. With respect to the cloud, elasticity is defined as “*Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand.*” [1]

A definition with the focus on distributed databases is provided by Agrawal et. al. [5] with “*Elasticity, i.e. the ability to deal with load variations by adding more resources during high load or consolidating the tenants to fewer nodes when the load decreases, all in a live system without service disruption, is therefore critical for these systems.*”

Hence, a DBaaS architecture requires a high elasticity factor, defined by scaling the cluster at run-time without downtime and improving the throughput to resolve the workload fluctuations [6].

## III. DATABASE AS A SERVICE ARCHITECTURE

In this section, we introduce the technical architecture for our following evaluation. This potential architecture of a DBaaS covers the underlying cloud infrastructure, the scalability-enabled architecture of distributed databases and the corresponding storage models.

### A. Cloud Infrastructure

A DBaaS system requires a high degree of flexibility on the resource level, so that it can be built upon a bare metal infrastructure or upon the IaaS cloud service model. Based on cloud features such as resource pooling or rapid elasticity [1], IaaS offers more flexibility than bare metal and is therefore the preferable way to build a DBaaS system.

IaaS provides processing, storage, networks and other fundamental computing resources to run arbitrary software [1]. The processing and storage resources are typically encapsulated in a virtual machine (VM) entity that also includes the operating system (OS). VMs run on the virtualised physical infrastructure of the IaaS provider, which is not accessible to the user. An exemplary IaaS architecture is depicted in Fig. 1.

### B. Distributed Databases Architectures

Distributed databases provide a single database system to the user which is spread over multiple nodes as depicted in Fig. 1. A single database instance as part of the overall distributed database system is termed database node in the following. The overall distributed database system is termed database cluster. Our evaluation only considers distributed, shared-nothing database clusters where each database node resides on its own VM as shown in Fig. 1.

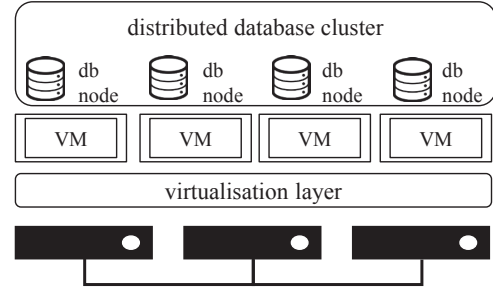


Fig. 1: System Model

The architecture of distributed databases is typically categorised into the following distribution models [7]:

1) *Single Server*: A single server architecture represents the simplest option without any distribution at all. The single node handles all read and write requests. We do not consider single server architectures as they do not offer horizontal scalability.

2) *Master-Slave*: In a master-slave distribution the data is replicated across multiple nodes. One node represents the designated master node, executing all write requests and synchronizing the slave nodes, which only execute read requests. The master-slave distribution can be applied to scale read-heavy workloads.

3) *Multi-Master*: The multi-master or peer-to-peer distribution do not build upon different node types, i.e. all nodes are equal. In a multi-master distribution replication and sharding [7] of the data is applied to spread write and read requests across all nodes of the cluster.

### C. Storage Models

Distributed databases are commonly classified according to the following storage models.

In *relational data stores*, data is stored in tuples, forming an ordered set of attributes. Relations consist of sets of tuples while a tuple is a row, an attribute is a column and a relation forms a table. Tables are defined using a normalised data schema. SQL has established itself as a generic data definition, manipulation and query language for relational data. *Column-oriented data stores* store data by columns rather than by rows. This enables to store large amounts of data in bulk and allows for efficiently querying over very large, structured data sets. A column-oriented data model does not rely on a fixed schema. *Key/value data stores* relate to hash tables of programming languages. The storage records are tuples consisting of key/value pairs. While the key uniquely identifies an entry, the value is an arbitrary chunk of data. *Document-oriented data stores* are similar to key/value stores. However, they define a structure on the values in structured formats such as XML or JSON. Compared to key/value stores, document stores allow more complex queries, as document properties can be used for indexing and querying. Inspired by the graph theory, *graph-based data stores* rely on graph structures for data modelling. Hence, vertices and edges represent and contain data. Queries often relate to graph traversals.

#### IV. METHODOLOGY

Based on the presented challenges for DBaaS, this section elaborates on our methodology to evaluate the scalability and elasticity of three distributed databases. The following section describes the methodology used to select the databases, to decide on the workload issued on the databases and the way it is generated. Finally, we define two evaluation scenarios for capturing the scalability and elasticity properties of the databases.

##### A. Database Selection

With the emergence of cloud computing the number of available distributed database systems increased as well. However, not all storage models fit well to scalability and elasticity requirements. Relational databases offer distributed representatives such as MySQL Cluster<sup>1</sup> or Postgres-XL<sup>2</sup>. Yet, neither MySQL Cluster<sup>3</sup> nor Postgres-XL<sup>4</sup> support elasticity, as resizing the cluster will lead to a downtime for both databases. Additionally, relational databases offer limited horizontal scaling capabilities compared to NoSQL databases [8]. Therefore, we do not consider relational databases for our evaluation. We also do not consider graph databases, as their elasticity is either very limited, e.g. neo4j<sup>5</sup> provides scale-out capabilities only for reads, or the elasticity properties rely on the underlying storage backends (e.g. Titan<sup>6</sup> with Cassandra). The specific structure of graph-based data and the corresponding queries make it also difficult to compare its performance metrics.

Hence, we set the focus of our evaluation databases to the remaining storage models: column-oriented, key-value and document-oriented. Below, we provide a brief overview of the selected databases and their selection criteria. All of them provide automated data sharding and promise high scalability and elasticity.

1) *Apache Cassandra*: Apache Cassandra<sup>7</sup> is selected as one of the most common column-oriented data stores<sup>8</sup>. Cassandra aims to run on top of commodity infrastructure and scale up to hundreds of nodes. Its architecture follows the multi-master paradigm and is designed to handle high write throughput without sacrificing read efficiency [9].

2) *Couchbase*: Couchbase is a document-oriented data store which can also be operated as a key-value data store. Its architecture follows the multi-master paradigm and takes advantage of memcached<sup>9</sup> for in-memory caching. We select Couchbase because of these features and also due to fact that Couchbase has gained industry attention [10], [11] for its scalability. Yet, Couchbase has undergone only limited scientific evaluation so far.

3) *MongoDB*: MongoDB<sup>10</sup> has been selected as the most prevalent document-oriented data store<sup>8</sup>. MongoDB's architecture is built upon three different services: (1) *mongos* act as query router, providing an interface between clients and a sharded cluster, (2) *configs* store the metadata of the cluster, (3) *shards* contain the actual data.

##### B. YCSB Benchmarking Tool

In the course of the database systems evolution, the variety of database benchmarking tools grew as well. Common representatives of such database benchmarking tools include TPC Benchmarks<sup>11</sup>, RUBiS [12] and the Yahoo Cloud Serving Benchmark (YCSB) [6].

While TPC workloads (TPC-C and TPC-E) target online transaction processing (OLTP) applications and RUBiS workloads target a typical 3-tier web application, YCSB offers simple database-centric workloads based on create, read, update and delete (CRUD) operations. Therefore, we chose the YCSB tool as it enables a comparable evaluation of the selected databases and their corresponding data models.

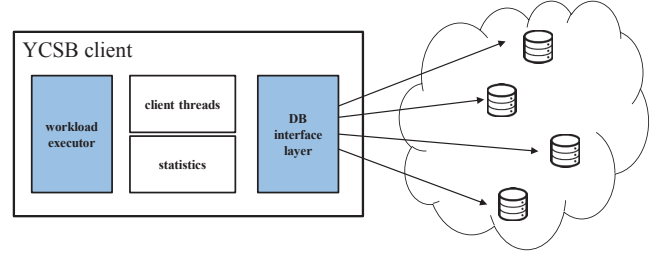


Fig. 2: YCSB client architecture

1) *YCSB Architecture*: YCSB offers an extensible architecture shown in Fig. 2. A YCSB client comprises the four depicted modules, *workload creator*, *client threads*, *statistics* and the *database interface layer*. The workload creator and the database interface layer offer a convenient interface to customise or extend the respective module. The design of YCSB enables the usage of an arbitrary number of YCSB clients in a distributed setup to generate the desired load.

2) *YCSB Metrics*: The statistics module of YCSB aggregates and reports the latency metrics (average, 95th and 99th percentile, minimum and maximum). The throughput is aggregated and reported as average and actual throughput at a specific timestamp. We focus in our evaluation on the average throughput and latency and the time series representation of the throughput to evaluate the elasticity of distributed databases, which will be described in more detail in the remainder of this section.

##### C. Workloads and Workload Generation

As introduced in Section IV-B YCSB issues CRUD operations by default. In addition, it provides a set of built-in distributions to perform the required random choices during

<sup>1</sup><https://www.mysql.com/products/cluster/>

<sup>2</sup><http://www.postgres-xl.org/>

<sup>3</sup><http://dev.mysql.com/doc/mysql-cluster-manager/1.4/en/>

<sup>4</sup><http://files.postgres-xl.org/documentation/tutorial-createcluster.html>

<sup>5</sup><https://neo4j.com/>

<sup>6</sup><http://titan.thinkaurelius.com/>

<sup>7</sup><http://cassandra.apache.org/>

<sup>8</sup>[http://db-engines.com/en/ranking\\_trend](http://db-engines.com/en/ranking_trend)

<sup>9</sup><https://memcached.org/>

<sup>10</sup><https://www.mongodb.com/>

<sup>11</sup><http://www.tpc.org/default.asp>

the load generation. The supported distributions are *uniform*, *zipfian*, *latest* and *multinomial*. We choose the zipfian distribution for our workloads, as it represents a typical distribution of web workloads [13]—some records are very popular and are often accessed, while most of the records are not.

For the distribution of the operations, we select two common workloads of cloud based applications [6]: read-heavy and a read-update. The distribution of operations as well as the selected distribution of each workload is provided in Table I.

TABLE I: Composition of Chosen Workloads

Workload	Create	Read	Update	Delete	Distribution
Read-heavy	0.0%	0.95%	0.05%	0.00%	zipfian
Read-update	0.0%	0.50%	0.50%	0.00%	zipfian

#### D. Scalability Evaluation Scenario

The scalability evaluation bases on static cluster sizes, which allow to evaluate the scalability of the selected databases by comparing the average throughput and latency for each cluster size and determine and compare *throughput* and *latency* across the different cluster sizes.

Four static cluster sizes and a fixed number of YCSB clients are considered as depicted in Fig. 3. The cluster sizes comprise a 1-node, 2-node, 4-node and 8-node cluster of the selected databases (cf. Section IV-A). Small node sizes highlight the effects induced by changes to the cluster size. For all three databases and all four cluster sizes, we use the same, fixed number of YCSB clients to generate the workload for each database cluster.

All evaluations are performed on a previously loaded database cluster containing 1,000,000 records. All scalability evaluation runs comprise 10,000,000 operations that are equally distributed among all YCSB clients involved. We select the size of the database (i.e. 1,000,000 records) such that it entirely fits in RAM, which allows us to exclude disk I/O as bottleneck and to focus on distribution and scaling effects.

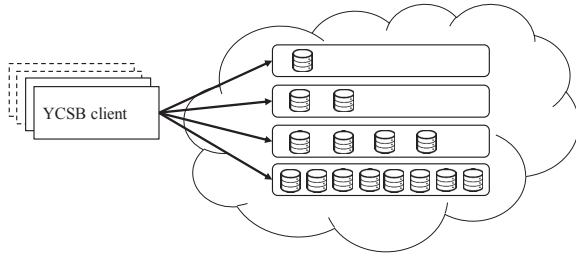


Fig. 3: Static Cluster

In order to achieve comparable results for all databases we calibrate the number of YCSB clients. For the scalability evaluation scenario, we determine the number of YCSB clients such that all database clusters are subject to comparable load. In particular, we pick the number of clients such that no database node is overloaded ( $>90\%$  CPU utilisation) nor idling ( $<20\%$  CPU utilisation), even in a 1-node configuration.

On the database side, no specific configurations are made. Only for MongoDB, we needed to decide on a distribution of the components not available for the two other database types (`mongos` and `configs`; cf. Section IV-A3). Therefore we monitored the resource consumption of each component and derived an appropriate distribution which is described in Section VI-A.

#### E. Elasticity Evaluation Scenario

The elastic evaluation scenario investigates a sudden growth in workload can be overcome by scaling the cluster under load. Starting from a 1-node cluster, the cluster is scaled-out as soon as the node running the database gets overloaded, i.e. the throughput drops significantly. We use the CPU utilisation ( $>90\%$ ) and a drop in throughput as indicators for an overloaded node and extend the cluster when both conditions occur. In order to ensure overload is reached for all three databases, we vary the number of YCSB clients accordingly (cf. Fig. 4).

When an overloaded state for the 1-node cluster is reached, we manually add a new node. During that time, the load is kept at the same level. Based on the time series representation of the current throughput, we are able to evaluate the elasticity during the redistribution of the data and in the resulting 2-node cluster of each database.

As before, the 1-node cluster starts with 1,000,000 pre-loaded records. For operations, we need to guarantee a high load over a longer period of time, ensuring the overload state is reached, the redistribution of the data is finished and the database has time to stabilise. Therefore, we increase the total number of operations to 500,000,000, which are again distributed over all YCSB clients.

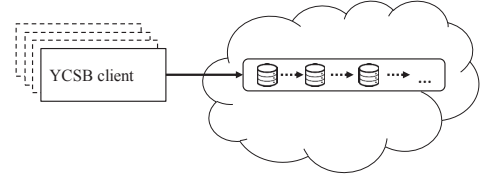


Fig. 4: Elastic Cluster

## V. EVALUATION ENVIRONMENT

The overall evaluation takes place in a private, OpenStack-based cloud<sup>12</sup> with full and isolated access to all physical and virtual resources. The cloud infrastructure<sup>13</sup> is operated with OpenStack version Kilo. In order to reduce possible side effects and to guarantee reproducible results we use the *availability zones* feature of OpenStack to dedicate physical hosts for spawning the required VMs. Hence, we dedicate one physical host to the YCSB clients and one to the database nodes. This reflects the optimal VM distribution for a distributed database in a cloud environment as one physical host is capable to run all database nodes (cf. Table II) and no

<sup>12</sup><https://www.openstack.org/>

<sup>13</sup><https://www.uni-ulm.de/in/omi/institut/blog/details/article/our-openstack-physical-testbed-part-3up-to-date-with-software-and-requirements/>

additional communication on a physical level is required. In the following, we describe the technical details of the cloud infrastructure, the YCSB client and database VMs.

#### A. Physical Infrastructure

The VMs for the YCSB clients and database nodes are placed on a different physical host. Table II shows the technical details for the physical hosts of the YCSB clients and the database nodes. The physical hosts are connected via 1Gb.

TABLE II: Physical Host Details

	YCSB Client Host	Database Host
CPU	2xIntel Xeon E5-2630v3 8-Core Haswell 2.4Ghz	2xIntel Xeon E5-2637 8-Core SandyBridge 2.6Ghz
Memory	64 GB ECC DDR4	64 GB ECC DDR4
Storage	2x1 TB HDD 7.2k rpm	NAS
Network	gigabit ethernet	gigabit ethernet
Operating System	CentOS 7, Linux 4.2.0-1.el7.elrepo.x86_64	CentOS 7, Linux 4.2.0-1.el7.elrepo.x86_64
Hypervisor	KVM, QEMU 1.5.3	KVM, QEMU 1.5.3

#### B. YCSB Client

The YCSB client VMs are provisioned with the details shown Table III. Observing the resource consumption of YCSB has shown that, with respect to the VM flavour, the mainly consumed resource is CPU whereas memory and disk I/O are negligible. Hence, the provisioned VM flavour is focused on CPU. For the YCSB tool we use our fork<sup>14</sup> of the original YCSB version 0.8 with an updated version of the Couchbase DB interface from 1.4.10 to 2.2.2. Additional details of YCSB configuration are shown in Table IV.

TABLE III: YCSB VM Details

VM Detail	Configuration
CPU	4 vCores
Memory	2 GB
Storage	10 GB
Network	gigabit ethernet
Operating System	Ubuntu Server 14.04.2 AMD64 LTS 3.13.0-59-generic x86_64
YCSB Detail	Configuration
Version	0.8
# of records	1.000.000
record size	1 KB
# of operations (for scalability scenario)	10.000.000
# of operations (for elasticity scenario)	500.000.000
# of threads	20

#### C. Database Node

Each database VM is provisioned with the details shown in Table IV. The size of the database VMs represents a typical commodity hardware configuration<sup>15</sup> and is selected to fit the

required system requirements of all the selected databases.

The specific version of the evaluated databases is shown in the second part of Table IV. In order to guarantee a fair comparison we use each database in its vanilla version without custom optimisations. We only configure the available memory for each database to 6 GB to reserve 2 GB for the operating system. Each database is configured with the minimum replication configuration.

TABLE IV: DB VM Details

VM Detail	Configuration
CPU	4 vCores
Memory	8GB
Storage	80GB
Network	gigabit ethernet
Operating System	Ubuntu Server 14.04.2 AMD64 LTS 3.13.0-59-generic x86_64
DB Detail	Configuration
Apache Cassandra version	2.2.6
Couchbase version	4.0.0 community edition
MongoDB version	3.2.7

## VI. EVALUATION RESULTS

In this section, we present our scalability and elasticity evaluation results. For the scalability evaluation, we show the throughput and latency development for static cluster sizes. For the elasticity evaluation, a dynamic cluster is scaled at run-time under continuous load and the throughput development is presented as time series for each database.

#### A. Calibrating the Scalability Scenario

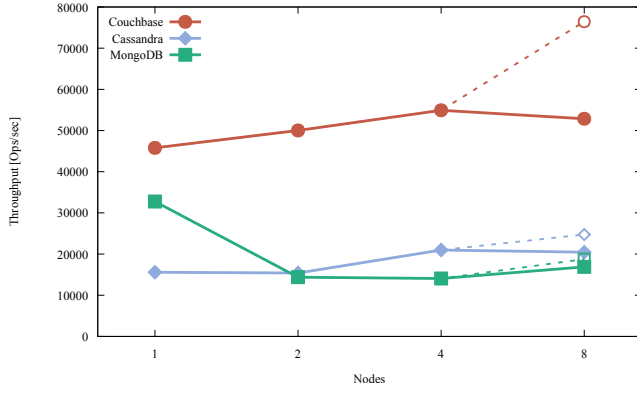
Initial evaluation results show that three YCSB clients overload a MongoDB 1-node cluster, so that two clients is the maximum possible amount of clients. At the same time, using only one client barely loads Couchbase. Hence, we decide to use two YCSB clients for the entire scalability scenario. For any kind of 1-node and 2-node clusters, all database nodes stay within the load boundaries (cf. Section IV-D). The calibration results for 8-node cluster show that 2 YCSB clients are not sufficient to saturate Couchbase and Cassandra. Hence, all evaluations of the 8-node clusters are performed with 2 and 4 YCSB clients to ensure saturation and comparable results. Regarding the distribution of the different MongoDB nodes our calibration efforts show that the `shard` services create the main load. The `mongos` and `config` services each only generate a negligible load of <10% CPU utilisation. Therefore, for the 1-node cluster, we run all three MongoDB services in the same virtual machine. For larger clusters, we only deploy new shard services. This setup still allows a fair evaluation against the other databases

During the calibration, we execute the read-heavy and read-update workloads 10 times in a 2-node Couchbase cluster environment. We encounter a standard deviation of the average throughput of <500 operations per second. This leads us to the conclusion that our private cloud environment causes only little noise and provides stable results. In consequence, we

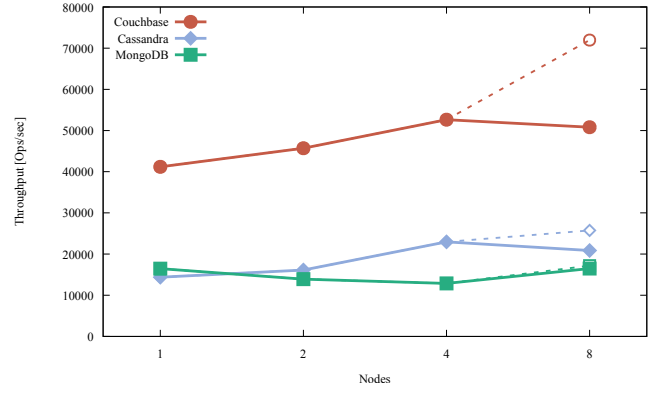
<sup>14</sup><https://github.com/sebybi87/YCSB/releases/tag/scalability-evaluation>

<sup>15</sup><https://docs.mongodb.com/manual/administration/production-notes/#hardware-considerations>

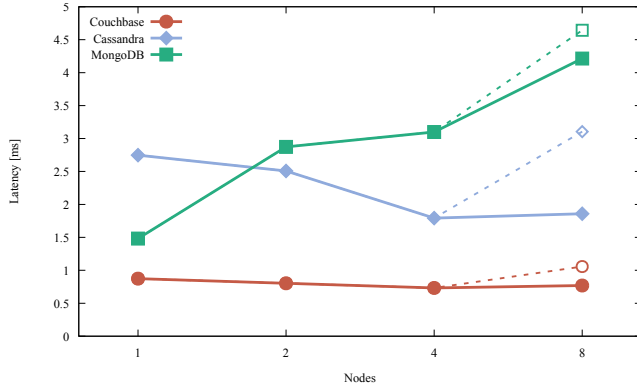




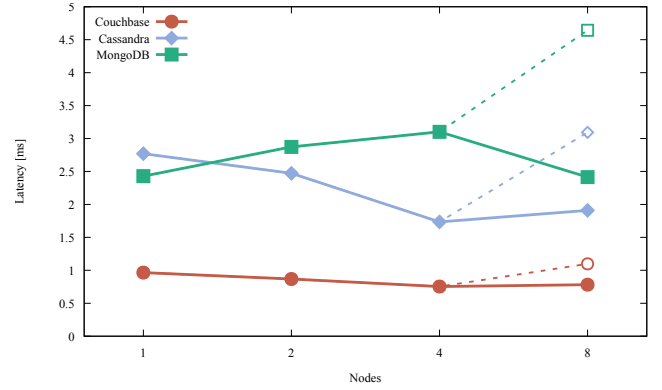
(a) Throughput Read-Heavy Workload



(b) Throughput Read-Update Workload



(c) Latency Read-Heavy Workload



(d) Latency Read-Update Workload

Fig. 5: Troughput and latency results of different workloads.

consider a repetition factor of 3 for all scalability evaluations as sufficient.

In order to exclude the network as possible bottleneck for the evaluation results, we monitor the YCSB clients and database nodes using Ganglia<sup>16</sup>. As depicted in Fig. 6 the network load on a single YCSB client with 20 threads reaches at most  $\approx 600$  KB/sec (in/out combined). Accordingly, in worst case (cf. Table V), we can expect  $\approx 3.0$  MB/s at a database node, which is far below the maximum bandwidth provided by our 1Gb network and allows us to neglect network saturation.

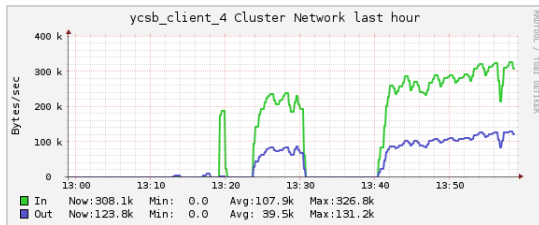


Fig. 6: YCSB Network Load

## B. Results of the Scalability Evaluation Scenario

In the following, we present the results of the scalability evaluation scenario. For both workloads we describe the measured throughput and latency per database and with growing cluster sizes. The dashed lines show the additional results by using 4 YCSB clients in an 8-node cluster (cf. Section VI-A).

1) *Read-Heavy Workload*: Fig. 5a depicts the throughput achieved by all databases in all cluster configurations. It is noticeable that Couchbase has the highest overall throughput for all cluster sizes. It is capable of increasing throughput by 17% when going from a 1-node to a 4-node cluster. Increasing the cluster size from 4 to 8 nodes decreases the throughput by 4% because 2 YCSB clients are not sufficient to saturate the 8-node cluster. By using 4 YCSB clients instead the throughput again increases by 29%.

Cassandra achieves the lowest throughput in a single node cluster, but outperforms MongoDB for the 2-node and 4-node cluster due to a throughput increase of 26%. Similar to Couchbase, the throughput decreases by 1% in an 8-node cluster with 2 YCSB clients but increases by 16% using 4 YCSB clients.

In contrast to its competitors, MongoDB performs better as single node than in a cluster. Its throughput decreases by 58% when moving from a 1-node to a 4-node cluster. For an 8-node

<sup>16</sup><http://ganglia.info/>

cluster throughput increases by 18% for 2 YCSB clients and by 26% for 4 YCSB clients compared to a 4-node cluster.

The latency results shown in Fig. 5c unveil a similar ranking as for the throughput. Couchbase achieves the lowest latency for all cluster sizes. Increasing its cluster to four nodes decreases latency by 17% compared to a 1-node configuration. In a 8-node cluster the latency remains constant for 2 YCSB and increases for 4 YCSB clients by 44% compared to a 4-node cluster.

Cassandra benefits the most from increasing the cluster size as a 35% latency decrease from one to 4 nodes is achieved. For eight nodes the latency increases by 3% for 2 clients and by 73% for 4 clients.

MongoDB reaches the highest latency for all cluster sizes and even with an increasing cluster size the latency increases by 100% from one to four nodes. In an 8-node cluster the latency increases by 35% for two and by 49% for four YCSB clients.

2) *Read-Update Workload*: For the read-update workload we measure similar trends as for the read-heavy results: Couchbase achieves the highest overall throughput and the lowest latency. Throughput is summarised in Fig. 5b, latency in Fig. 5d: Couchbase increases its throughput by 22% from a 1-node to a 4-node cluster and by 36% for an 8-node cluster with 4 YCSB clients. Cassandra again reaches the highest throughput increase of 38% from 1-4 nodes and 12% in an 8-node cluster with 4 YCSB clients. MongoDB's throughput decreases by 27% from 1-4 nodes and increases by 28% (2 YCSB clients) and by 33% (4 YCSB clients) in an 8-node cluster. Couchbase achieves a latency decrease of 20% when increasing the cluster size from one to four nodes. For 8 nodes and 2 YCSB clients the latency remains constant and for 4 YCSB clients it increases by 45%. For Cassandra we measure a 37% latency decrease from 1-4 nodes and for 8 nodes an increase of 10% for 2 YCSB clients and 78% for 4 YCSB clients. The latency of MongoDB increases by 38% from 1-4 nodes and decreases in an 8-node cluster with 2 YCSB clients by 22%. With 4 YCSB clients the latency increases by 49%.

### C. Calibrating the Elasticity Scenario

As the elasticity evaluation bases on an overloaded 1-node database cluster (cf. Section IV-E) the required number of YCSB clients has to be determined for each database separately. For that purpose, for each database we proceed as follows: Starting with three clients (cf.

Section VI-A), we issue load on a 1-node cluster of that database, while monitoring CPU usage and throughput. Then, we increase the number of clients until the metrics show a constant overload situation. The resulting number of required YCSB clients is summarised in Table V.

For the distribution of MongoDB services, we use the same approach as in the scalability scenario.

The calibration results show that a benchmark run-time of 15 minutes is sufficient time for all the databases to redistribute the data and balance the load across the nodes. Therefore, each benchmark run is manually terminated after 15 minutes.

TABLE V: Number of clients used for the elasticity scenario.

	Cassandra	Couchbase	MongoDB
Clients	4	5	3

### D. Results for the Elasticity Evaluation Scenario

The evaluation of the elastic cluster is performed with the read-update workload and a each database is previously loaded with 1,000,000 records.

1) *Apache Cassandra*: For Cassandra our experimental results show that four YCSB clients are required to create an overload situation for a single node. The four YCSB clients are started in 30s gaps to ensure a warm-up time for each YCSB client. We continuously monitor the throughput of the Cassandra instance and the CPU utilisation of the hosting VM. The throughput graph of Cassandra's throughput is depicted in Fig. 7a as time series. All four YCSB are running after 90s and produce the overload situation with a dropping throughput at 110s. In addition the CPU utilisation reaches >90% at this point. At 120s a new Cassandra node is added to the cluster and the data is getting rebalanced. As depicted in Fig. 7a the throughput stops dropping after rebalancing the data but the 2-node cluster does not reach the throughput of the one node cluster in the beginning of the benchmark.

2) *Couchbase*: Our experimental results have shown that five YCSB Clients are required to overload one Couchbase node. Again, the YCSB clients are started in 30s gaps and all clients are started after 120s. As depicted in Fig. 7b the throughput starts dropping at 130s and a second node is added at 150s. After rebalancing the data, the two node cluster increases the throughput significantly and compared to the initial one node cluster

3) *MongoDB*: For overloading a single MongoDB node, three YCSB clients are required. The three clients are started in 30s gaps and produce an overload situation for the MongoDB instance at 100s as depicted in Fig. 7c. By adding a second node at 120s the overload situation is balanced as the throughput increases but only slightly above the throughput of the initial one node cluster.

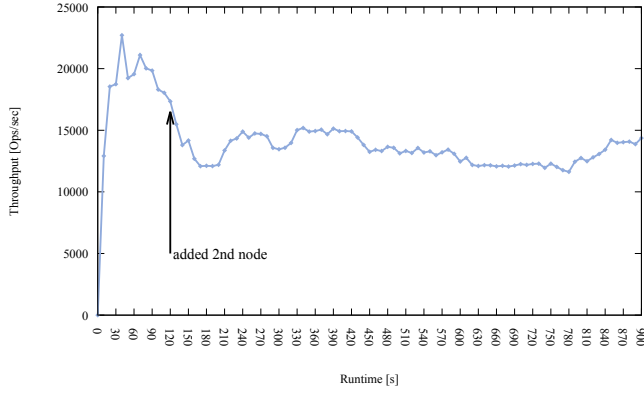
## VII. DISCUSSION

The main insight from our evaluation is that *more instances* does not necessarily mean *better performance*. Indeed, in the case of MongoDB more instances decreases throughput and increases latency.

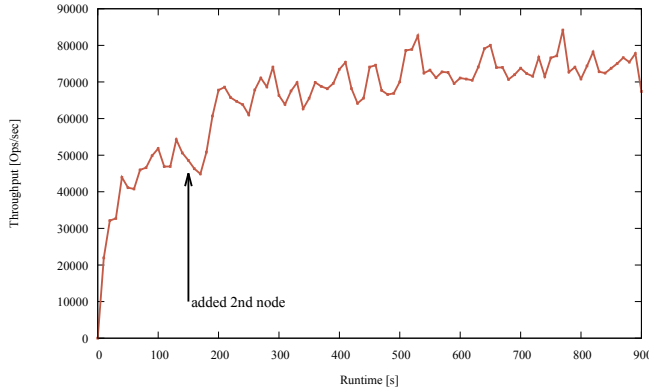
This section discusses the results from a high level view. It clarifies the impact of the evaluation methodology on the results received and draws conclusions. It also addresses impact on the design of distributed architectures.

### A. Interpretation of Scalability Results

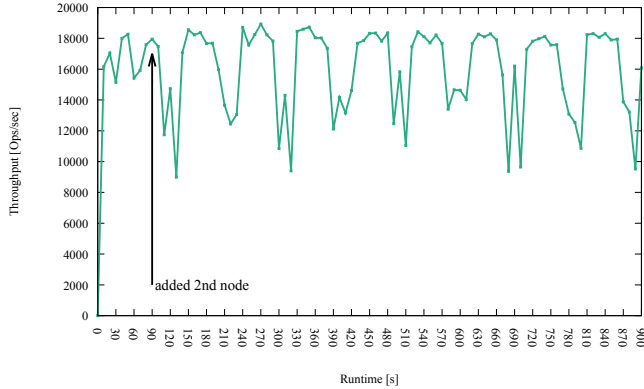
Key to the interpretation of the scalability evaluation is the fact that for all three databases, two clients were used (the four clients results in the 8-node cluster are discussed at the end). Both clients use 20 threads leading to a maximum



(a) Cassandra



(b) Couchbase



(c) MongoDB

Fig. 7: Elasticity benchmark results: adding an additional node while the database system is under load.

parallelism of 40 requests. This number was selected such that none of the databases was overloaded in a single node set-up. In consequence, for the scalability evaluation, no database instance was overloaded, and the performance metrics of the databases are directly comparable.

In an ideal case going from one to two nodes, throughput would double and going from one to four nodes it would quadruple (cf. Section II-A). The fact that this has not hap-

pened could have three causes: (i) the clients or the network are maximally utilised and so that clients cannot issue new queries and updates faster (we excluded this by carefully calibrating the number of clients and checking the network bandwidth). (ii) The database instances are overloaded (this is prevented by the set-up). (iii) Queuing at the databases occurs.

In case a single database instance can only serve  $n < 40$  parallel requests this will cause the remaining requests to be queued. As each client thread issues only one request at a time, this load will cause the system to adjust and lead to an (on average) constant queue size. Adding a second database node to the cluster will decrease the load on the first one, as data is sharded between the two nodes. Due to using a zipfian distribution for the access pattern, an unequal distribution of load amongst nodes could be expected. As all evaluated databases apply hash-based sharding in the used default configuration, the prevention of hot spot nodes is addressed at database level. Yet, even with an uniform load distribution, throughput would only double when each database instance could process  $n \geq 20$  requests at the same time. As we do not see a large increase in throughput for any of the databases (even not for 4-node clusters), we conclude that the parallelism for each database is  $n < 10$ .

Looking at the results for the 8-node cluster, the throughput decrease compared to a 4-node cluster might be surprising in the first place. Yet, it leads us to the assumption of a too large cluster size in respect to the issued load by two YCSB clients. Hence, an overhead of inter-cluster communication is caused by request routing as the results show no further latency decrease compared to a 4-node cluster. Increasing the load by using four YCSB clients confirms our assumption as the clusters are saturated and the throughput increases in respect to two YCSB clients. Yet, the increased load also increases the latency significantly for all three databases.

This analysis is accurate for Couchbase, which does not apply any replication strategy out of the box. The impact of their lazy (i.e. asynchronous) replication on throughput behaviour for Cassandra and MongoDB remains unclear. Also, the routing strategy of the respective database can impact the throughput (as well as the latency): while Couchbase and Cassandra route messages through the database nodes in a peer-to-peer manner, MongoDB uses a centralised routing service. Even if this service does not impose measurable load on a virtual machine, the fact that all messages from the clients have to proceed through one place, may have significant impact on the drop of throughput and increase in latency when adding more MongoDB instances. A reason may be that MongoDB relies on smart indexes for achieving high throughput; YCSB does only use simple indexing to provide comparable results.

### B. Interpretation of Elasticity Results

The elasticity results raise even more questions than the results of the scalability evaluation. Only Couchbase exposes a behaviour a naive observer would expect. Adding a second node first takes some time for synchronisation, which also



causes a drop in throughput. After a while the system is fully operational again and is able to serve more requests per time unit than before.

In contrast to Couchbase, Cassandra requires more time to synchronise the new node with the existing cluster. While the drop in throughput (caused by the overload situation) stops at some point, Cassandra will not recover from the decrease, but continue performing at a lower average throughput than at peak time. This is remarkable as the 2-cluster throughput for the elasticity scenario stays below the 2-cluster throughput for the scalability. Here, the only difference is that the elasticity evaluation uses four clients instead of two for the scalability evaluation. Similar holds for MongoDB where the throughput does not change at all after scaling out. For this database, also the reason for the periodic drops in throughput raise questions. While we assume that garbage collection, internal compaction processes or similar mechanisms cause them, this need to be verified.

### C. Looking Ahead: Refining the Methodology

The scalability and elasticity evaluation provided throughout this paper has led to several surprising results. We on purpose go for a purely observative approach describing *how* the systems behaves. While we had not expected to achieve near linear performance when scaling out the databases—in particular not without tweaking their configurations—it is astonishing to see that after more than four decades of distributed systems research, the use of sharding can still lead to decreasing throughput and increasing latency.

The results gained here seem to indicate that we should move to a more analytical approach in order to understand *why* the platforms behave as they do. Nevertheless, we assume that such an approach is less lasting, as it is vulnerable to version upgrades and other changes to the algorithms used by the database implementation.

Instead, we suggest to take the work here as a starting point to develop an evaluation framework that is able to repeatedly and reliably characterise the scaling and elasticity behaviour of distributed databases. In order to do that, we shall refine our methodology. This allows us to isolate different triggers on the behaviour of the database (e.g. more load vs. higher replication degree or the relation between throughput and latency). In the long run, this will have to involve the collection large amounts of monitoring data and the execution of advanced data analytics.

## VIII. RELATED WORK

DBaaS providers typically deploy a distributed database system on IaaS resources to offer a elastic database service on demand. One of the first DBaaS are offered by Amazon's DynamoDB<sup>17</sup> or Google's BigTable<sup>18</sup> which use proprietary NoSQL databases. With the expansion of NoSQL, emerging open source NoSQL databases are being considered as

DBaaS storage backend, such as mlab<sup>19</sup> using MongoDB or OpenStack Trove<sup>20</sup> using Cassandra or Couchbase. Hence, performance evaluation of distributed databases is an ongoing research area.

Due to the evolution of distributed databases and cloud computing, new benchmarking tools are required with the focus on scalable and elastic databases. Released in 2010, YCSB [6] became the de facto standard benchmarking tool for distributed databases. The original version was developed by Yahoo to evaluate their column-oriented data store PNUTS [14] against Cassandra and HBase and the relational MySQL Cluster. The evaluation shows the scalability and elasticity results where Cassandra and PNUTS achieve the best results. The evaluation does not consider key-value and document-oriented data stores or overloading the database explicitly.

Rabl et. al. [4] evaluate six distributed databases (key-value, column-oriented and relational) in the light of storing monitoring data for application performance management (APM) systems. They use YCSB as benchmarking tool, but apply custom workloads reflecting typical APM workloads. Their evaluation focuses on two kind of workloads, memory-bound, i.e. records fitting completely in RAM and disk-bound, i.e. records do not completely fit into RAM. The cloud context is not considered as all evaluations are run on physical infrastructure. The results show for both workload types that Cassandra achieves the best scalability results while elasticity is not evaluated.

Gandini et. al. [15] evaluate the performance of three NoSQL databases (Cassandra, MongoDB and HBase) in the cloud. They are using Amazon EC2<sup>21</sup> as evaluation environment. The evaluation focuses on scalability in relation to the number of cores in a single-server setup and to the number of nodes in a distributed setup. In addition, they analyse how the replication degree influences performance. Their results show that Cassandra as well as MongoDB increase their throughput for static cluster sizes, whereas an increasing replication degree will decrease the throughput for both databases. However, the benchmarking methodology misses technical details like used database versions and YCSB version, which harms the comparison with our results.

Klein et al. [16] evaluate Cassandra, MongoDB and Riak in the light of a health care use case with a customised version of YCSB. The evaluation focuses on the performance and scalability with respect to differently strict consistency levels. Their results attest Cassandra the highest scalability, whereas similar to our results MongoDB does not scale.

Konstantinou et al. [17] analyse the distributed databases Cassandra, Hbase and Riak in the context of their TIRAMOLA framework for elastic NoSQL cluster provisioning. Their evaluation focuses on the cost in terms of time for scaling a database cluster and the optimal thresholds for executing a scaling action. Scalability and elasticity of distributed

<sup>17</sup>[https://aws.amazon.com/dynamodb/?nc1=h\\_ls](https://aws.amazon.com/dynamodb/?nc1=h_ls)

<sup>18</sup><https://cloud.google.com/bigtable/>

<sup>19</sup><https://mlab.com/>

<sup>20</sup><https://wiki.openstack.org/wiki/Trove>

<sup>21</sup><http://aws.amazon.com/ec2>

databases in respect to context of cloud computing is considered by [6] however without performing the evaluation in a fully transparent cloud environment. The elasticity aspect is considered by scaling a cluster at run-time, but without creating an explicit overload situation in the database.

## IX. CONCLUSION AND FUTURE WORK

Cloud computing provides “unlimited” resources, which can be allocated on demand. Based on these prerequisites, distributed databases gained significant focus in recent years. Whereas traditional relational databases offer distributed derivatives, a new class of distributed databases emerged, NoSQL databases. By promising horizontal scalability and elasticity, NoSQL databases are commonly used for the cloud service model Database as a Service (DBaaS).

In this paper, we present thoroughly evaluate the scalability and elasticity of the common NoSQL databases Cassandra, Couchbase and MongoDB with respect to DBaaS environment. Therefore, we define two evaluation scenarios, one for scalability and one for elasticity. The scalability evaluation comprises different static cluster sizes to determine the scalability with growing cluster sizes. The elasticity is evaluated by overloading a database and scaling-out the database at run-time to resolve the overload situation.

The scalability evaluation results show significant differences between the evaluated databases. Whereas Couchbase achieves the highest throughput and lowest latency results, Cassandra benefits the most from larger cluster sizes in contrast to MongoDB where a distributed setup even harms the performance. The elasticity results state that only Couchbase is able to resolve an overload situation at run-time in contrast to Cassandra and MongoDB.

The discussion concludes with the main outcome of the evaluation results: more instances does not necessarily mean better performance. An interpretation of the results is provided in greater depth by discussing possible bottlenecks on database and evaluation environment side. Whereas only Couchbase provides expected results for scalability and elasticity, the results of Cassandra and MongoDB raise additional questions for deeper evaluations. Therefore, the future work will focus on a deeper analysis of the gathered results by analysing the database distribution by including extensive resource monitoring and advanced data mining techniques. In addition, the evaluation of scalability and elasticity in larger scale clusters will also be targeted as well as the evaluation of distributed databases running on microservices platforms.

## ACKNOWLEDGMENT

The research leading to these results has received funding from the EC’s Framework Programme FP7/2007-2013 under grant agreement number 317715 (PaaSage) and the EC’s Framework Programme HORIZON 2020 (ICT-07-2014) under grant agreement number 644690 (CloudSocket). In addition, we thank Alexander Rasputin for assistance in performing the evaluation.

## REFERENCES

- [1] P. Mell and T. Grance, “The nist definition of cloud computing,” 2011.
- [2] S. Kächele, C. Spann, F. J. Hauck, and J. Domaschka, “Beyond iaas and paas: An extended cloud taxonomy for computation, storage and networking,” in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*. IEEE Computer Society, 2013, pp. 75–82.
- [3] N. R. Herbst, S. Kounev, and R. Reussner, “Elasticity in cloud computing: What it is, and what it is not,” in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, 2013, pp. 23–27.
- [4] T. Rabl, S. Gómez-Villamor, M. Sadoghi, V. Muntés-Mulero, H.-A. Jacobsen, and S. Mankovskii, “Solving big data challenges for enterprise application performance management,” *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1724–1735, 2012.
- [5] D. Agrawal, A. El Abbadi, S. Das, and A. J. Elmore, “Database scalability, elasticity, and autonomy in the cloud,” in *International Conference on Database Systems for Advanced Applications*. Springer, 2011, pp. 2–15.
- [6] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, “Benchmarking cloud serving systems with ycsb,” in *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010, pp. 143–154.
- [7] P. J. Sadalage and M. Fowler, *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2012.
- [8] R. Cattell, “Scalable sql and nosql data stores,” *Acm Sigmod Record*, vol. 39, no. 4, pp. 12–27, 2011.
- [9] A. Lakshman and P. Malik, “Cassandra: a decentralized structured storage system,” *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.
- [10] D. Nelubin and B. Engber, “Nosql failover characteristics: Aerospike, cassandra, couchbase, mongodb,” *Thumbtack Technology, Inc., White Paper*, 2013.
- [11] —, “Ultra-high performance nosql benchmarking: Analyzing durability and performance tradeoffs,” *Thumbtack Technology, Inc., White Paper*, 2013.
- [12] E. Cecchet, A. Chanda, S. Elnikety, J. Marguerite, and W. Zwaenepoel, “Performance comparison of middleware architectures for generating dynamic web content,” in *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*. Springer-Verlag New York, Inc., 2003, pp. 242–261.
- [13] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, “Web caching and zipf-like distributions: Evidence and implications,” in *INFOCOM’99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1. IEEE, 1999, pp. 126–134.
- [14] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, “Pnuts: Yahoo!’s hosted data serving platform,” *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1277–1288, 2008.
- [15] A. Gandini, M. Gribaudo, W. J. Knottenbelt, R. Osman, and P. Piazzolla, “Performance evaluation of nosql databases,” in *European Workshop on Performance Engineering*. Springer, 2014, pp. 16–29.
- [16] J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham, and C. Matser, “Performance evaluation of nosql databases: a case study,” in *Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems*. ACM, 2015, pp. 5–10.
- [17] I. Konstantinou, E. Angelou, C. Boumpouka, D. Tsoumakos, and N. Koziris, “On the elasticity of nosql databases over cloud management platforms,” in *Proceedings of the 20th ACM international conference on Information and knowledge management*. ACM, 2011, pp. 2385–2388.