Trilogy: Data Placement to Improve Performance and Robustness of Cloud Computing

Chin-Jung Hsu
Department of Computer Science
North Carolina State University
chsu6@ncsu.edu

Vincent W. Freeh
Department of Computer Science
North Carolina State University
vwfreeh@ncsu.edu

Flavio Villanustre
LexisNexis Risk Solutions
Flavio.Villanustre@lexisnexis.com

***Abstract*—Infrastructure as a Service, one of the most disruptive aspects of cloud computing, enables configuring a cluster for each application for each workload. When the workload changes, a cluster will be either underutilized (wasting resources) or unable to meet demand (incurring opportunity costs). Consequently, efficient cluster resizing requires proper data replication and placement. Our work reveals that coarse-grain, workload-aware replication addresses over-utilization but cannot resolve under-utilization. With fine-grain partitioning of the dataset, data replication can reduce both under- and over-utilization. In our empirical studies, compared to a näive uniform data replication a coarse-grain workload-aware replication increases throughput by 81% on a highly-skewed workload. A fine-grain scheme further reaches 166% increase. Furthermore, a surprisingly small increase in granularity is sufficient to obtain most benefits. Evaluations also show that maximizing the number of unique partitions per node increases robustness to tolerate workload deviation while minimizing this number reduces storage footprint.**

***Index Terms*—workload-aware data placement; data management; cloud computing**

I. Introduction

In large-scale, distributed systems the dataset, which is too large for a single node, is partitioned among the nodes. Incoming workload (*i.e.,* requests) is routed among nodes by a load balancer. For extreme horizontal scaling to be effective, it is necessary for nearly all requests to be routed to a node containing the needed data locally, which avoids unnecessary node-to-node interactions. Consequently, data replication and data placement are components of load balancing and have a substantial impact on system performance. In the literature, for example, AUTOPLACER [1] and MET [2] optimize data placement to fit workload characteristics in NoSQL databases. Replicating hotter data in storage is a common approach to balance load [3]. In relational databases, sharding is used to distribute load by partitioning tables to achieve effective horizontal scaling [4], [5], [6], [7], [8].

Cloud computing has changed how computing resources are used. Before cloud computing, infrastructure is purchased and used for many years before it is upgraded or replaced. However, with Infrastructure as a Service (IaaS) equipment is rented for a short time, including as short as one execution of an application. This shift from buying to renting greatly increases the flexibility of infrastructure available for a given application. Therefore, rather than tune an application to run well on a specific machine, a cloud user instead can tune the infrastructure to accommodate each application on each dataset. When an infrastructure is resized in response to changes in workloads, it is necessary to replicate and place data partitions among nodes. It is still unclear what are the better strategies for this decision—prior work on data placement does not totally address these issues [1], [2], [3].

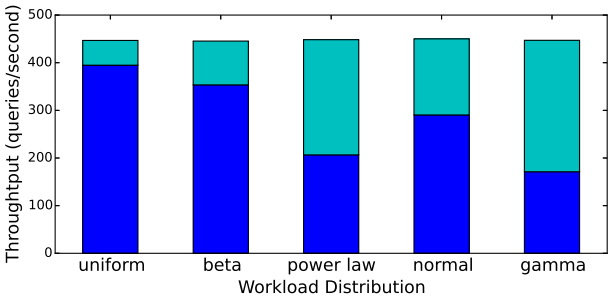


Fig. 1. Uniform data placement is suboptimal. The lower bar is the measured throughput of uniform placement while the upper bar is the performance loss to the idealistic placement. (Data is a subset of data shown in Table II on Section IV.)

Efficient deployment of distributed systems with an irregular workload requires both cluster sizing and data placement. Näive uniform data replication (where all data partitions have the same number of replicas) is effective only when the workload is also uniform—requests are equally distributed among all partitions. *Workload-aware* data placement replicates and places partitions to match the distribution of requests in the anticipated workload.

Fig. 1 illustrates the cost of uniform data replication on non-uniform workloads. The results for five different workloads are presented on the x-axis; the skew (degree of imbalance) in the workload increases from left to right. The *complete* placement solution, where each node has all data, is an idealistic upper bound on the potential gains of matching data replication and workload. Because any node can process a request, new requests are sent to the least loaded node and the performance of *complete* is flat across all workloads. In this example, the cluster has twice the minimum capacity, so uniform replication has two copies of each partition. Therefore, a request must be sent to one of the two nodes that hold the data associated with the request. Throughput decreases as the workload skew

978-1-5386-2715-0/17/$31.00 ©2017 IEEE

increases because some nodes are over loaded and others are under utilized. While uniform placement achieves 88% of ideal on uniform workload, it is only 38% of ideal on the highly-skewed gamma workload. This example illustrates the need to properly *replicate* and *place* data on the nodes of a cluster.

This paper explores the three dimensions that affect data placement. The first dimension is *granularity* of data partitions. Fine-grain (more than one partition per node) placement has costs (overhead) and benefits (flexibility). The second dimension is how many *replicas* of each partition. We let the anticipated demand per partition (*i.e.*, the *workload*) determine the replication factor for each data partition. The hotter the data, the greater the replication. The number of data partitions influences how closely the replication matches the workload. However, a coarse-grain partition (one per node) is unlikely to match the workload. Last, fine-grain partitioning introduces the *placement* dimension because there are several ways to distribute the partitions among the nodes.

This paper present results from a simulation program that examines these three dimensions. We find that coarse-grain placement does not provide sufficient flexibility to balance non-uniform loads. A surprisingly small amount of additional partition granularity is sufficient to load balance and obtain most benefits. The paper presents and evaluates the tradeoff between several fine-grain placement strategies that either increase robustness to tolerate workload deviation or reduce storage footprint.

To further examine these conclusions, our empirical study on an HPCC cluster[1] shows that proper data replication and placement affect system performance greatly. The coarse-grain scheme improves system throughput by 25% and 85% for the normal and power law distribution, respectively. A fine-grain placement strategy improves query throughput by 52% and 105%. On the most highly-skewed case, the improvement is 166% increase over the näive solution.

Because data placement relies on a prediction of the upcoming workload, which will invariably be wrong to some degree. This paper, therefore, considers the *robustness*, which is a measure to describe how sensitive a data placement scheme is to slightly mis-predicted workloads, of several placement strategies. Results show that maximizing the number of unique partitions per node increases the robustness of a placement.

The remainder of this paper is organized as follows. The next section describes a model for data replication and placement. Section III presents simulation results of different placement schemes. Section IV presents an empirical evaluation. Section V reviews the literature and the final section summarizes the paper.

II. Modeling Data Replication and Placement

Our work concerns systems in which the dataset must be partitioned among the nodes because the dataset is too large to be completely replicated on each node. We replicate subsets of the whole dataset in order to increase throughput and decrease latency. While replication for availability is critically important, it is not a subject of this research.

Distributed, large-scale systems such as Apache Hadoop, Spark, Cassandra, and Ceph largely exploit data locality while reducing node-to-node communication for achieving high horizontal scaling [9], [10], [6], [11]. Data partitions are replicated as the system scales out. An inefficient data placement scheme is unable to achieve the optimal system performance and service-level objective (SLO) violations may occur [1], [2], [12], [13].

The goal of this work is to place data partitions onto nodes such that the performance is maximized for the upcoming workload. There is a large body of work supporting workload prediction [14], [15], [16]. This work assumes that a reliable (though not necessarily perfect) prediction is provided by some other work. Instead of solely relying on accurate workload prediction, systems can dynamically adjust replication factors and data locations for handling workload changes [3], [12], [2]. This work focuses on determining the optimal partition granularity, replication factors, and placement strategy. Our work is complementary to a dynamic approach.

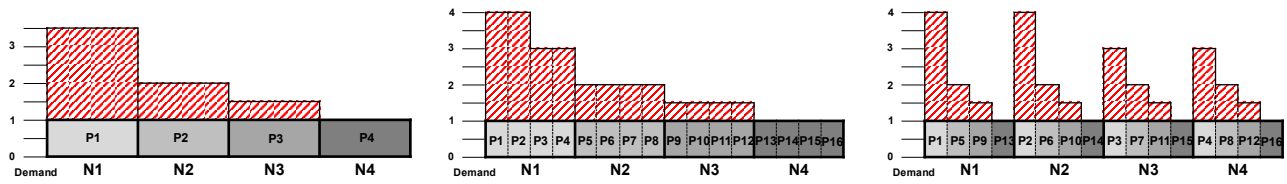
The following model characterizes the *data replication and placement problem* in large-scale, distributed systems. Let $M > 1$ be the minimum cluster size that is sufficient to hold all data. The storage capacity is strictly limited by the amount of data it physically can store locally. In many real-world applications, M is in the hundreds of nodes. However, for this model it is only necessary that M not be equal to one, which does not require data partitioning.

Let N ($N \geq M$) be the number of nodes in a cluster. When the workload changes, the cluster expands (N increases) to meet increased demand and minimize QoS violations, or it contracts (N decreases) to reduce resources and cost. But the cluster cannot contract smaller than the M nodes needed to hold the data.

The data is partitioned into $k \geq 1$ equal-sized data partitions on each node. Thus, the dataset has $P = Mk$ unique partitions. Because the cluster has N nodes, there are $S = Nk$ *slots* for data partitions. We define the *replication factor*, R, as $R = N/M$. When $R > 1$, then $N > M$ and $S > P$ and some partitions will be replicated among the "extra" slots. *Coarse-grain* data placement occurs when there is only one partition per node, $k = 1$. *Fine-grain* data placement, $k > 1$, which has more total data partitions and partition slots, supports more distinct placements than coarse-grain providing a better opportunity to match the workload, and increases performance.

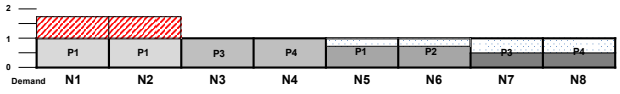
Model parameters	
Minimum number of nodes:	$M > 1$
Per node granularity:	$k \geq 1$
Replication factor:	$R \geq 1$
Derived terms	
Instantiated nodes:	$N = RM$
Unique data partitions:	$P = Mk$
Slots:	$S = Nk = RMk$

[1]HPCC Systems is an open source *data-analytics computer*—a highly scalable, distributed framework for processing and analyzing large datasets—supported by LexisNexis Risk Solutions at https://hpccsystems.com/.

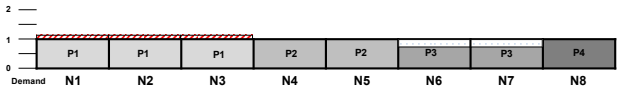


(a) Coarse-Grain ($M = 4, R = 1, k = 1$) (b) Fine-Grain ($M = 4, R = 1, k = 4$) (c) Fine-Grain alternative placement

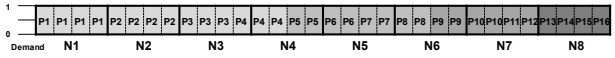
Fig. 2. The workload demand exceeds the system capacity.



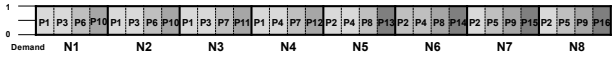
(a) Uniform data placement ($M = 4, R = 2, k = 1$)



(b) Coarse-grain data placement ($M = 4, R = 2, k = 1$)



(c) Fine-grain *compact* data placement ($M = 4, R = 2, k = 4$)



(d) Fine-grain *balanced* data placement ($M = 4, R = 2, k = 4$)

Fig. 3. Different data placement schemes.

We present a motivating example below. The base cluster has four nodes, $M = 4$. The current demand is twice the current capacity. Fig. 2(a) shows the load per partition on four nodes. The load is unevenly distributed among the partitions. In particular, in terms of node capacity the load on the four partitions in Fig. 2(a) is 3.5, 2, 1.5, and 1 (which conveniently totals 8). Fig. 2(b), shows the same aggregate demand as it is distributed among 16 partitions, four per node: $k = 4$. Fine-grain replication gives rise to a *placement* decision. Redistributing loads helps reduce load imbalance among nodes by placing highly requested partitions onto least loaded nodes. In Fig. 2(c) the load is nearly balanced: two nodes have a load of 2.125 and two have 1.875. However, this alone does not help solve the overloading issue because the workload demand is twice the system capacity.

Suppose the cluster doubles in size to eight nodes exactly meeting anticipated demand: $R = 2$ and $N = 8$. Uniform replication onto eight nodes creates two replicas of each partition as shown in Fig. 3(a). The red box above the two left-most nodes shows the excess demand on those nodes as 3.5 units of node capacity are serviced by two nodes. The white regions in the four right-most nodes show the underutilization because the demand is less than the capacity.

A coarse-grain, non-uniform solution is clearly better than uniform because *hotter* partitions can be replicated more than *colder* ones. For example, in Fig. 3(b) the 3.5 units of P1 are distributed over three nodes. Unsurprisingly, a fine-grain solution can better match the demand to the capacity. Fig. 3(c) shows that demand is perfectly matched to capacity in this idealized example.

Fig. 3(d) shows a second placement that also perfectly balances load but has four unique partitions on each node. The nodes in Fig. 3(c) have either one or two unique partitions. Fewer unique partitions per node reduces the footprint of both primary (memory) and secondary (disk) storage. On the other hand, more unique partitions per node increases the number of nodes that can respond to a given request, which may better share the load among nodes.

This simple example was constructed to clearly show the benefits of non-uniform, fine-grain data placement. Unless the demand is uniformly distributed among the partition (an extremely unlikely occurrence as explained in Section III-A) then a näive uniform placement leads to over- and underutilization.

III. Data Placement Schemes

This section presents the data placement problem. First, it characterizes the workloads used in the paper. Next, it expounds on the three dimensions of the problem: (1) granularity, (2) replication, and (3) placement. Last, it explains and analyzes three data placement methods, comparing performance in terms of load imbalance, storage footprint, and robustness.

A. Workload Characteristics

A workload can be described with several critical characteristics, such as arrival rate and autocorrelation. Because this paper considers data replication and placement, the workload characteristic that matters most is access frequency of the individual elements of the dataset, such as, the pages of a web server or the keys in an index. Other characteristics are not factored into this work because they do not have a direct impact on data replication or placement.

A uniform workload is atypical and likely artificial, which is unsurprising because non-uniform workloads are common in many natural settings [17]. For example, the normal (Gaussian) distribution can be observed in class score distribution, while the log-normal distribution is useful to describe the response file size in web servers [18]. In addition, the power law probability distribution is widely applicable to web hits, word frequency, personal income, *etc*. and tends to be highly skewed towards a small subset of the full dataset [19]. That is, a small number of partitions accounts the vast majority of key access

TABLE I
LOAD-IMBALANCE OF WORKLOADS.

Distribution	Requests for individual partitions				max:mean	skew
Uniform	**7,565**	7,548	7,449	7,438	1.01	0.15
Beta	**10,313**	10,288	4,715	4,684	1.38	0.39
Power law	**17,344**	8,795	3,361	500	2.31	0.60
Normal	**14,882**	14,827	149	142	1.98	0.74
Gamma	**23,542**	6,329	129	0	3.14	0.77

and most of the partitions are touched infrequently. Several studies show that frequency of access to different pages or keys often follows a Zipf or power law distribution. This has been shown in web servers [20], [21], video streaming [22], and Wikipedia traffic [23] to name a few.

In this work, we consider the *normal* and the *power law* distribution for their wide appearance in many workloads. We also consider the *uniform* distribution for a näive baseline, *beta*, which is less skewed than *normal* and *power law*, and *gamma*, which generates the highest skewed workload and has been used in modeling workloads in storage systems [24].

Table I shows the five workload distributions used in the paper. This table presents the distribution of the requests on each of four partitions ($M = 4$, $k = 1$). There are 30,000 unique requests among the 1024 keys, and the average is 7,500 requests per partition. The requests-per-node values are ordered in decreasing magnitude. As expected there are approximately the same number of requests for each partition in the uniform distribution. The *max:mean* column shows the ratio of the maximum number of requests to the average. Because the total running time is largely dependent on the slowest or most heavily loaded node, the max-mean ratio foretells the performance penalty for each workload on a uniform distribution. The ratio for uniform is 1.01, meaning the maximum is 1% greater than the average. But the highly-skewed gamma distribution has one partition that receives no requests and one has more than three times the average. It is clear from Table I that in non-uniform workloads the maximally loaded partition demands more resources than the other partitions. The final column presents access imbalance using the *skew* metric [17].

This paper evaluates the problem using synthetic workloads. An alternative is to evaluate using real-world traces. But such traces are in short supply. Moreover, a trace represents a very specific situation that may not be representative of a general class. Additionally, a trace has many characteristics that are hard to control. Using synthetic workloads enables us to evaluate more distributions and confine the observed effects to the change in distribution.

B. Data Placement Steps

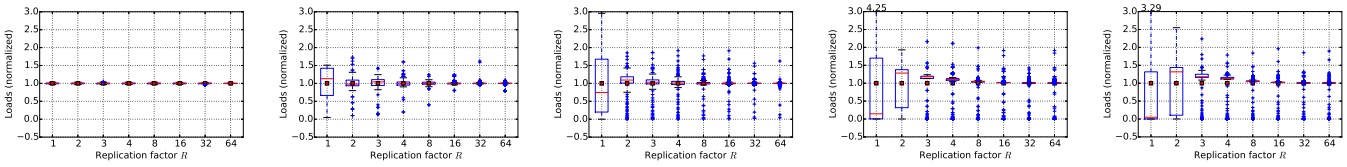
A data placement method requires determining partition granularity, replication factors, and placement schemes. Partition granularity represents the smallest unit for replicating data and calculating loads. The coarsest granularity is one partition per node ($k = 1$), and the finest is one partition per key. A small k decreases the likelihood of balancing a non-uniform workload. However, a large k increases management overhead.

Once the partition granularity k is determined, the next step in deriving a solution is determining the number of replicas for each partition based on the expected workload. For example, suppose there are four partitions ($P = 4$) with a replication factor of four ($R = 4$), then there are sixteen slots for these partitions ($S = 16$) in a coarse-grain solution. Given a uniform expected workload the replication factor vector would be [4, 4, 4, 4], that is there are four copies of each of the four partitions. For a workload with a normal distribution the replication factor vector might be [2, 6, 6, 2] and for power law it might be [1, 2, 4, 9]. In general, there is no perfect match between the vector and the anticipated workload.

Assuming that the predicted load on each partition is λ_i and the total load is $\Lambda = \sum_{i=1,P} \lambda_i$. The replication problem is to determine the *replication vector*, $\vec{R}, R_i \in \mathcal{I}$, such that $R_i \geq 1 \ \forall i$ and $S = \sum_{i=1,P} R_i$. In words, the replication vector contains the number of replicas (an integer value) of each partition, such that the total number of replicas equals the number of slots available. The replication error is $E = \sum_{i=1,P} |R_i/S - \lambda_i/\Lambda|$, which is the accumulation of difference between the actual relative replication of each partition (R_i/S) and the anticipated relative workload per partition (λ_i/Λ).

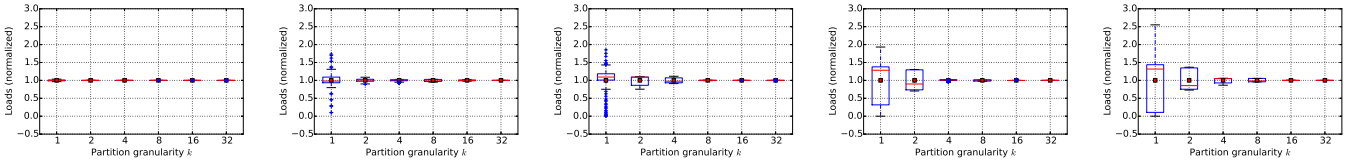
The last step is assigning the replicas to the slots on the nodes. For coarse-grain, the number of replicas equals the number of slots and there is only one possible placement. But for fine-grain, $k > 1$, there are many possible placements. One placement strategy is to minimize the number of unique partitions on each node in order to reduce dataset footprint on the nodes. We call this strategy *compact*. If there are multiple choices, it picks the node with the least load. The opposite strategy to *compact* is *full*, which maximizes the number of different partitions on each node and also picks the node with the least load. The third strategy is placing partitions in order to balance loads as much as possible. We call this strategy *balanced*. The paper shows that although *full* and *balanced* have different goals, the resultant placements and effects are similar. Furthermore, *compact* and *full* balance loads among the nodes with their best efforts within their given constraints. Consequently, all three strategies achieve good load balancing, of course *balanced* is a slightly better at it. To reiterate: The goal of *compact* is reducing the number of unique partitions on each node, which reduces the footprint of the data set. On the other hand, the goal of *full* is to distribute replicas for the same partition to as many nodes as possible—balancing the load—with the intent of increasing the availability of hot partitions.

Our data placement procedure is described in Algorithm 1. This is a framework and therefore, different placement strategies can be used. The complexity for generating the replication vector is proportional to the number of partitions, $\mathcal{O}(Mk)$. Different placement strategies implement distinct *pick_node* functions, which is $\mathcal{O}(N)$. This function is executed for each slot (Nk). Therefore, the total complexity is of the placement algorithm is $\mathcal{O}(kN^2)$. Although it is quadratic, it is in the



(a) uniform (b) beta (c) power law (d) normal (e) gamma

Fig. 4. The load distribution among nodes under the coarse-grain data placement ($M = 64, k = 1$).



(a) uniform (b) beta (c) power law (d) normal (e) gamma

Fig. 5. The load distribution among nodes under the fine-grain data placement with various k ($M = 64, R = 2$).

number of nodes that, even in a very large cluster, is tractable. A straightforward Python implementation executes in under few seconds when $N = 256$ and $k = 16$. Furthermore, this solution is a heuristic, it does not find the optimal solution. However, the solution is nearly optimal and because it is based on a prediction of anticipated load optimal is unnecessary.

Algorithm 1: Data Placement Procedure

Input: an historical workload
Output: partition placement on each node

```
M := the minimum number of nodes that hold all data
k := the selected partition granularity
R := the replication factor
loads := the predicted loads λ_i of partitions
replicas := the replication vector (see Section II)
for p_i in replicas do
    for j=1; j <= p_i; j = j + 1 do
        /* strategy is compact, balanced or full */
        n := pick_node(strategy)
        assign p_i to n
    end
end
```

C. Tradeoffs in Placement Strategy

Our data placement framework is shown in Algorithm 1. This framework yields several variants of data placement when choosing different placement strategies. This section compares their effects on load balancing and storage footprint.

1) Load Balancing: The primary goal of data placement is distributing workloads evenly among nodes. A highly skewed system is more likely to encounter performance bottlenecks. Therefore, a well-balanced system achieves a higher system throughput. To explore the benefit of fine-grain replication and placement, we evaluate the effectiveness of our data placement schemes in balancing the anticipated workload. We generated 100 instances of workloads, of 300,000 queries, for each of the five distributions. The workload instances vary because the access counts are generated probabilistically. Each generated workload is considered a prediction of the upcoming workload.

First, we evaluate how the replication factor R affects load balancing. This simulation is conducted with $M = 64$ and $k = 1$. We use the box plot, as shown in Fig. 4, to analyze the loads distributed among nodes. The bottom and top of the rectangle represent the first and third quartile of loads. The red line is the median, and the red dot is the mean. To facilitate comparison, the loads are normalized so that the means equal one. Above the box is the whisker that is calculated by adding 1.5 times the interquartile range (IRQ) to the third quartile. Similarly, the whisker below the box is the first quartile minus 1.5 times the IRQ. The plus signs represent the data points that have the loads beyond the two whiskers. These is the standard representation for a box plot. In a well balanced system, the median and mean values will be close and the box will be small.

This figure clearly shows that increasing the replication factor reduces load imbalance to a degree. However, this alone is insufficient to eliminate under-utilized loads totally because increasing the number of replicas only reduces the loads. The only way to reduce under utilization is to overlap under and over utilized partitions, which is not possible in the coarse grain method. Therefore, the replication factor alone is not sufficient for balancing loads because it does not reduce under-utilization. Take the *gamma* distribution for example, when $R = 1$, most of the nodes are under utilized (low median) and few nodes have extremely excess loads (large size box). Doubling the replication factor creates more overloaded nodes because $R = 2$ is not enough to distribute loads. As R increases, the median value moves closer to the mean and the variance (the size of box) also decreases, which suggests over-utilization is mostly addressed. However, increasing R is still insufficient because there are still many outliers (beyond

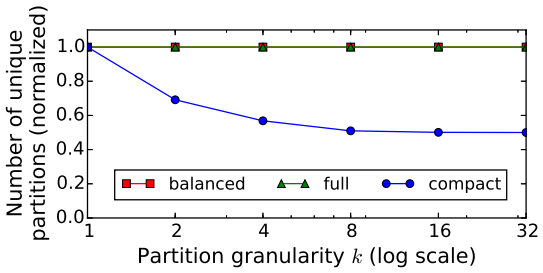


Fig. 6. The number of unique partitions per node (storage footprint) under different placement schemes.

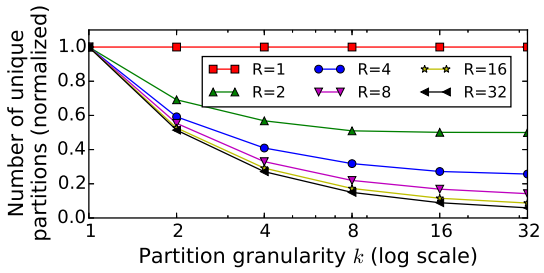


Fig. 7. The number of unique partitions per node under the *compact* method with various k. The number converges at $k = 32$, which is equal to $1/R$.

the two whiskers).

Next, we examine how much the partition granularity can reduce load imbalance. We run a set of simulations with $M = 64$, $R = 2$, and various k values, and choose *balanced* as the placement scheme. Fig. 5 clearly shows that fine partition granularity greatly reduces load imbalance. Even in the most skewed workload (gamma), $k = 16$ is able to almost perfectly balance loads. For most workloads, $k = 4$ is sufficient to reduce the load imbalance below 10%. When workloads are highly skewed (the *normal* and *gamma* case), their load imbalance (the size of box in the figure) drops greatly from $k = 1$ to $k = 2$ because finer partitioning provides higher flexibility to mix over- and under-utilized partitions on the same node. Increasing partition granularity eventually leads to (for all practical purposes) perfectly balanced loads.

2) Storage Footprint: There are $S = Nk$ choices for placing a partition replica. When placing multiple replicas of the same partition on the same node, it reduces the number of unique partitions per node. When the number of unique partitions per node is lower, it generally requires lower storage footprint. A lower storage footprint reduces memory pressure, and may lead to higher cache efficiency. We run a set of simulations with $M = 64$, $R = 2$ and various k. This paper only presents the results of the *power law* workloads. Other workload cases show very similar effects.

Fig. 6 shows the average number of unique partitions on each node. By definition all partitions in *full* are unique and it is always 1 (normalized). *Balanced* is very nearly 1 in all cases. On the other hand, as k increases, *compact* tends to 0.5, which is $1/R$. We further investigate how the replication factor affects storage footprint. Fig. 7 shows that the number of unique partitions tends to $1/R$ as k increases, which is the desired outcome of the *compact* scheme.

IV. Evaluation

This section presents our evaluation. We introduce our experimental setup and benchmark design. We then evaluate different placement schemes by measuring query throughput and testing their robustness to slight workload mispredictions.

A. Experiment Setup

We conducted our evaluation on Virtual Computing Lab (VCL), a cloud platform provided by NC State University. All servers are equipped with 2-core Intel Xeon CPU and 8GB memory and connected to a 10 Gbit switch. VCL only provides limited storage capacity of local disks or *instance storage*. This is also the case for many cloud configurations. In fact, a majority of EC2 instance types in Amazon Web Services (AWS) do not have any instance storage. Instead one must mount a remote volume served by a enterprise-level storage system. (AWS calls this *elastic block storage*–EBS.) We evaluate our approach using both instance storage (local disks) and remote volumes provided by network file system (NFS), backed by NetApp 2554 filer with dual controllers.

We evaluate our placement schemes on high performance computing cluster (HPCC) [25]. HPCC is an open source data analytics computer developed by LexisNexis Risk Solutions for processing big data. They maintain several clusters with more than 100 nodes, the largest with more than 500, to provide services to their clients. Experiments in this paper were conducted on a HPCC Roxie cluster. Roxie is a *data delivery engine* that responds to queries. It finds the answers to requests in an index that is partitioned and, if desired, replicated across the nodes. Roxie is optimized to handle massive amounts of concurrent requests with low latency.

Data replication and placement that fit workload demands have direct performance impact on performance. Roxie clusters partition and distribute data with two replicas per partition by default. We modified Roxie to incorporate our data placement schemes. Our approaches are not specific to Roxie. They should be able to apply to Apache Hadoop, HBase, Cassandra, and Ceph, providing benefit when workloads are not uniformly distributed across keys, partitions or nodes.

B. Workload Generator and Benchmark Suite

To evaluate the results learned in our simulation, we developed a distributed benchmark tool that is able to issue a large volume of concurrent queries to Roxie. This benchmark tool adopts the master-slave architecture, where the the master node generates workload according to a workload profile, and the slave nodes execute the query requests. This tool is customizable and supports any number of workload distributions. This benchmark suite is written in Python, and designed for testing query performance at large scale.

In our evaluation, we are interested in how placement schemes with different levels of partition granularity respond

TABLE II
STEADY-STATE THROUGHPUT COMPARISON (INSTANCE STORAGE)

	Uniform	Beta	Power Law	Normal	Gamma
base	394.8	353.5	206.6	290.4	171.2
coarse	-	367.8 (4.0%)	381.9 (84.9%)	364.0 (25.3%)	309.7 (80.9%)
compact	375.4 (−4.9%)	377.5 (6.8%)	383.2 (85.5%)	374.6 (29.0%)	374.2 (118.6%)
balanced	408.1 (3.4%)	412.6 (16.7%)	422.8 (104.7%)	442.5 (52.4%)	455.9 (166.3%)
complete	446.8 (13.2%)	445.4 (26.0%)	448.3 (117.0%)	450.1 (55.0%)	447.0 (161.1%)

unit: queries/second

TABLE III
STEADY-STATE THROUGHPUT COMPARISON (NFS)

	Uniform	Beta	Power Law	Normal	Gamma
base	446.1	383.1	220.2	393.3	176.4
coarse	-	396.4 (3.5%)	415.3 (88.6%)	379.6 (29.4%)	327.9 (85.9%)
compact	403.7 (−9.5%)	416.2 (8.7%)	401.3 (82.3%)	407.1 (38.8%)	407.8 (131.1%)
balanced	447.6 (0.3%)	440.8 (15.1%)	454.0 (106.2%)	469.7 (60.1%)	485.9 (175.5%)
complete	484.2 (9.0%)	485.6 (26.8%)	492.4 (123.7%)	495.0 (68.8%)	490.0 (177.8%)

unit: queries/second

to different types of workload distribution. We consider *uniform*, *beta*, *power law*, *normal*, and *gamma* distributions. The beta distribution is defined on the interval $[0, 1]$ with two shape parameters, α and β. We choose $\alpha = 2$ and $\beta = 2$ for the base case of beta distribution. The power law distribution is controlled by the *shape* parameter and we choose 3 for the base case. Regarding the normal (or Gaussian) distribution it has a *mean* and a *standard deviation* parameter, which is 0 and 1 in our case. The gamma distribution also has a *shape* parameter and the base case uses 5. A single instance of each workload is used in all the empirical tests of Roxie so that results can be compared across multiple runs and different configurations. The specific workloads used are those shown in Table I.

C. Benchmark Steps

To best measure the performance, our benchmark service runs one worker node for each Roxie server, which eliminates the performance impact at the client side. We use separate machines from the Roxie cluster on the VCL for the benchmark service. The Roxie controller node dispatches requests and synchronizes with workers. Worker nodes request jobs and execute them as soon as possible.

We generate the five workload distributions with different access counts to keys. All datasets are 128GB. Next, we specify the smallest cluster size M, the replication factor R (which determines the cluster size N), and the partition granularity k. In our evaluation, M is equal to 4. The coarse-grain schemes replicate data on a node basis. Fine-grain schemes, on the other hand, divides the data on a node into 32 equal-size partitions (1GB per partition).

We compare five placement schemes in total. First, *base* represents the uniform data placement. It is coarse grain ($k = 1$) and not workload aware. The *coarse* scheme is also coarse-grain but replicates partitions based on anticipated workload. For the fine-grain schemes ($k = 32$), we consider *compact* to reduce storage footprint while maximizing cache locality, and *balanced* to minimize load imbalance among machines. In our evaluation, we found that the *balanced* and *full* scheme have comparable performance. Due to the page limit, we report the results of *balanced* in most cases. Last, the *complete* is an "idealistic" placement where each node holds the entire dataset. It represents an upper bound.

Our next step is to change the data layout in Roxie to reflect the desired data placement decision. The Roxie cluster is restarted to load the new data layout. To avoid cache interference, the file system cache is cleared before every benchmark run.

Last, a workload profile is submitted to the benchmark controller. The controller node generates the query plan accordingly. In this way, the same stream of requests is presented for each benchmark, which allows us to verify results with multiple identical runs and to compare results from different placements. We collect query throughput during the entire benchmark process.

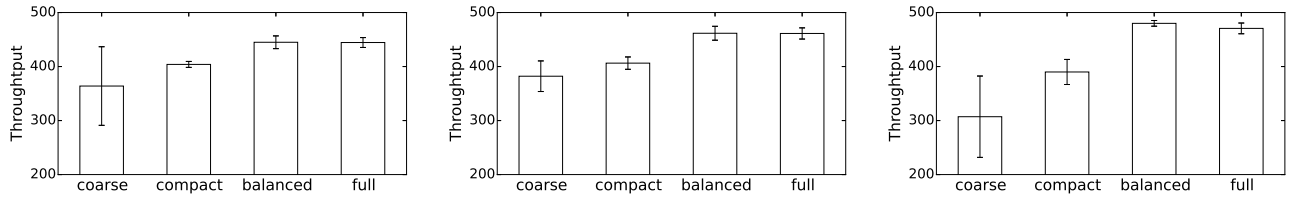
D. Steady-State Throughput

We conduct this evaluation to test steady-state throughput. We generate 30,000 requests for each of the five workload distributions. We then calculate the average throughput over the sampling period (the first and last 10% period are not included.) This measurement ensures we capture the stable throughput, but not the warm-up period (low throughput) and the long-tail period (system is not saturated).

1) Local Storage: Our evaluation starts with storing data required for Roxie queries on local disks. This evaluation involves 8 Roxie nodes: $M = 4$ and $R = 2$. Table II shows the throughput of proposed replication and placement schemes under different workloads. The values in parenthesis are the speedup relative to *base* performance at the top of each column. The *base* placement strategy is uniform. It does not perform well as the skewness of workload increases. For example, the power law workload in the uniform data placement can only achieve 52.3% of the throughput of a uniform workload. The second strategy is also coarse grain but replicates according to anticipated workload. On a uniform workload this is the same as *base*. It out performs *base* on skewed workloads. For example, it achieves 84.9% more throughput than *base* on *power law*.

Two fine-grain approaches, *compact* and *balanced*, which further improve performance over *coarse*, are also shown. In the normal workload case, *compact* and *balanced* improve on *coarse* by an additional 10.6 and 78.5 queries per second. In the gamma case, *balance* adds 146.2 queries, a 47.2% improvement over *coarse*. Workload-aware data placement is preferable for non-uniform workloads. The fine-grain strategies out perform *coarse* on all the skewed workloads. This is attributed to better load balancing. As skew increases, the benefit from fine-grain increases (because the load imbalance in *coarse* increases).

The *complete* solution out performs all others, including both fine-grain solutions. This is because while the workload was probabilistically generated over 30,000 requests. The



(a) Beta (b) Power Law (c) Gamma

Fig. 8. Compare robustness under slight workload mispredictions. The y-axis represents queries per second, and starts from 200 for better presentation to tell performance difference.

workload for each small window of requests does not always reflect the overall workload. In such cases, *complete* performs better. However, *complete* is generally not feasible when dataset is too large to fit into one node.

Overall, workload-aware data placement significantly increases query throughput. Using fine partition granularity better balances the load. *Balanced* performs better than *compact*, indicating that the benefit of a smaller footprint is less than the cost of poorer load balancing. The *balanced* scheme is occasionally competitive with *complete*.

2) Remote Storage: Next, we evaluate our proposed schemes against data storing on remote storage. The Roxie cluster size and the number of benchmark clients remain the same with the local storage case. Table III details the throughput numbers. This evaluation confirms the general observations seen in the instance storage test. However, the throughput is higher using remote storage. While somewhat counter intuitive, it is not unheard. This occurs because local storage uses plain commodity disks and the filer uses high-performance disks as well as aggressive caching. Moreover, the I/O demand does not exceed the capacity of the NFS server. Therefore, the additional network traffic is not creating a performance bottleneck.

E. Robustness Comparison

We are interested in how sensitive a placement scheme is to minor deviations in the anticipated workload. (Tables II and III show performance degradation for major deviations.) We say a placement scheme is more *robust* when the scheme works well even when the actual workload is slightly different from the anticipated workload. We pick different parameters for generating slight workload variance. For example, we change the shape parameter in the power law distribution. Therefore, it becomes either less or more skewed. We create two less and two more skewed workloads for each type.

Fig. 8 shows how different placement schemes react to workload shifts. The figure shows the average throughput and the standard deviation of the placement schemes under the four "shifted" workloads The figure indicates that the coarse-grain scheme under performs in both average (lower) and deviation (greater) compared to the fine-grain schemes.

The *compact* scheme is better than *coarse* but its performance is not as good as the *balanced* scheme. The *balanced* scheme overall exhibits higher throughput than *coarse* and *compact*. More importantly, *balanced* shows consistent standard deviation in three workloads. The highest performance degradations in each workload are 2%, 6% and 3% while the *compact* scheme shows 3%, 5% and 14% (increasing as skewness increases) degradation respectively. The above suggests than *balanced* is more robust then *compact*, which is robust than *coarse*. There is little difference between *balanced* and *full* in either average throughput or standard deviation. This is because there is little difference in the placement of partitions—that is, the balanced scheme tends to have a high degree of unique partitions on each node.

F. Micro Benchmark

We have presented the steady-state throughput in Section IV-D. In this section, we further examine why different placement schemes lead to large performance difference.

We investigate resource utilization of different placement schemes for understanding the tradeoff between the *compact* and *balanced* scheme. We collect system statistics (*dstat* [26] and *cachestat* [27]) during the entire benchmark runs. Table IV presents the system statistics, and metrics are normalized to the smallest value in each metric group, except the *max:mean* ratio. This normalization better shows the difference between placement schemes. Except *mean %CPU*, a system is more efficient when the metrics listed are small. These metrics are collected from the benchmark runs under the gamma workload. Other workloads present very similar trends.

First, we examine CPU utilization across all Roxie nodes. The average CPU utilization indicates whether Roxie is fully saturated, and the *max:mean* metric tells whether loads are well balanced among Roxie nodes. In the *base* scheme, CPU utilization is the lowest and load imbalance is the highest, which explains why uniform data placement under performs. Workload-aware replication eliminates load imbalance while improving CPU utilization. Fine-grain partition further reduces load imbalance, as in the *balanced* scheme.

Second, we examine the benefits of packing multiple replicas into the same node, as in the *compact* scheme. Table IV shows that cache misses and dirty pages are significantly lower in the *compact* scheme. Besides, *compact* has much lower cache sizes, 17% lower than *balanced* and 51% less than the *coarse* scheme. Although the *compact* scheme outperforms others in cache locality and requires less cache, it does not generate the highest query throughput. A possible explanation

TABLE IV
NORMALIZED SYSTEM STATISTICS OF ROXIE SERVERS

	Metrics	*base*	*coarse*	*compact*	*balanced*	*complete*
Load Balancing	% CPU (mean)	**1.00** (13%)	2.29	2.37	3.11	2.97
	% CPU (max:mean)	2.01	1.34	1.23	1.1	1.14
Cache Locality	cache misses (sum)	1.13	1.24	**1.00** (659K)	1.26	1.22
	dirty pages (sum)	1.20	1.47	**1.00** (200K)	1.52	1.29
	cache sizes (max)	2.11	1.51	**1.00** (825MB)	1.17	1.15
Efficiency	I/O wait (mean)	1.39	6.73	**1.00** (2.35%)	2.48	2.55
	TCP connections (mean)	2.40	1.53	1.31	1.10	**1.00** (1357)

is that requests do not greatly benefit from better cache locality. We suspect the *compact* scheme is useful especially when query applications require costly read operations.

Third, we compare I/O wait time and the number of TCP connections for comparing their efficiency. A lower I/O wait time indicates that systems do not waste much time on slow I/O operations. The *compact* scheme, with maximum cache locality, has the lowest I/O wait value. The number of TCP connections at a given time is related to processing efficiency. We observe that the *balanced* scheme yields a lower number of TCP connections than the other schemes, suggesting that requests complete more quickly.

G. Summary

Our evaluation provides empirical data to support our findings in the simulation. First, workload-aware data placement, both the coarse and fine grain methods, reduces load imbalance, thereby improving system throughput. However, the coarse-grain approach is insufficient when workload is highly skewed. Finer partitioning further balances the loads and in many cases, the *balanced* scheme has comparable performance to *complete* (an "idealistic" placement). Furthermore, both *balanced* and *full* are robust while *compact* reduces storage footprint to $1/R$.

V. RELATED WORK

Partition granularity, data replication and data placement are three dimensions to handle non-uniform workload. These problems are widely studied in storage systems [3], memory caching [28], [29], content delivery systems [30], and database systems [1], [2], [31], [17], [8]. Our work extends this line of research to better support cloud computing.

Partition Granularity: Partitioning data is the first step towards load sharing among nodes. Partition granularity can be blocks, objects, and tables. A generic file systems uses, for example, blocks of 4 or 8KB, while Hadoop Distributed File System (HDFS), designed for large file, adopts a much larger block size, *e.g.*, 64MB, which reduces management overhead of handling large files [32]. In Ceph, a distributed object storage system, an object is the unit of partition [11]. Sharding, which partitions large tables into smaller pieces, is a common technique to distribute load among multiple databases instances [33]. It is used in both SQL and NoSQL databases such as BigTable [4], HBase [5], and Cassandra [6].

Much recent work in database systems focuses on partitioning methods to minimize transaction overhead and to maximize load balancing. Schism uses graph-partitioning algorithms to determine the optimal partitioning for minimizing the number of distributed transactions [8]. Pavlo *et al*. develop a database tool that is able to generate optimal database designs for parallel OLTP systems [17]. They propose a new search algorithm for partitioning a database that minimizes the coordination cost while maximizing load balance. Spanner is a semi-relational database that manages replicated data globally and performs resharding at runtime for better load balancing [31]. Slicer is an auto-sharding service designed by Google [7]. Slicer dynamically maps the key to a proper task for minimizing load imbalance. In generally, tuning the partition size is critical to improve system throughput or reduce query latency [34].

Data Replication: Replicating data helps alleviate hot spots of data partitions [35]. Many distributed storage systems such as HDFS and Ceph support configurable replication factors. Hotter data can be replicated more times for handling irregular workloads. AptStore proposes the Popularity Prediction Algorithm (PPA), which adjusts replication factors according to file popularity, mining from file system logs [36]. Scarlett combines offline log analysis and online prediction to accurately estimate block popularity [37]. Results show that replicating hotter data avoids bottlenecks, thereby improving the performance of MapReduce jobs. Our work does not focus on popularity prediction; instead, we determine the replication factor based on the popularity, *i.e.*, access frequency, of the data partitions. Users need to provide the historical workloads.

Data Placement: Placing replicas affects workload distribution to a system. An efficient method better balances loads in a distributed system. For example, HDFS use rake-awareness placement scheme for fault tolerance but not for optimizing performance. CoHadoop leverages application hints to co-locate related data on the same set of Hadoop nodes [38]. This placement scheme benefits Hadoop operations such as indexing and join. Another example is to address performance degradation in heterogeneous Hadoop clusters by placing data according to processing capacity of nodes [13].

The following studies have shown that non-uniform query key causes load imbalance, thereby degrading system performance. AUTOPLACER [1] optimizes the placement for the top-k objects in the distributed key-value store. AUTOPLACER proposes a probabilistic data structure to reduce routing latency. MET is an elasticity framework that resizes and reconfigures HBase automatically for fitting workload characteristics, in heterogeneous environment [2]. MET groups data partitions according to their access pattern, and places data partitions using Longest Processing Time (LPT) to minimize the cost while maximizing load sharing among nodes. The above implements a system that replicates and places data according to data access pattern. Our work explores how data replication and placement schemes affect system performance under different workload distributions.

In summary, our work is an integral study on partition granularity, data replication and data placement. Besides, we

consider the replication factor and the robustness that are critical to cloud computing. Our findings are applicable to systems that have the flexibility to replicate and place data, such as distributed storage systems and NoSQL databases.

VI. Conclusion

Efficient deployment of large-scale, distributed systems with an irregular workload requires both cluster sizing and data placement. We show the uniform distribution is (unsurprisingly) poor for typical, non-uniform workloads. This work further shows that coarse-grain replication can reduce over-utilization but is unable to address under-utilization. Finer partition granularity reduces both under- and over-utilization. With fine-grain partitioning there is a placement decision. Maximizing the number of unique partitions per node increases robustness to workload misprediction, while minimizing the number reduces storage footprint. Our empirical study using an HPCC Roxie cluster shows the benefit of footprint reduction does not offset cost due to poorer load balancing. However, we do not believe this is universally true.

This work focuses on the dimension and tradeoffs of various data replication and placement strategies. For our future work, we plan to implement an elastic controller that incorporates our proposed data placement schemes. In an elastic system, the gains of the optimal placement must be offset by the cost of data movement. Calculating this data movement cost remains as a future work.

References

[1] J. P. Rodrigues, P. Ruivo, P. Romano, and Luís, "AUTOPLACER: Scalable Self-Tuning Data Placement in Distributed Key-value Stores," in *ICAC'13*, 2013, pp. 119–131.

[2] F. Cruz, F. Maia, M. Matos, R. Oliveira, J. Paulo, J. Pereira, and R. Vilaça, "MeT: workload aware elasticity for NoSQL," in *Eurosys'13*, 2013, pp. 183–196.

[3] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," in *ICAC'10*. ACM, 2010, pp. 1–10.

[4] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[5] L. George, *HBase: the Definitive Guide: Random Access to Your Planet-Size Data*. " O'Reilly Media, Inc.", 2011.

[6] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[7] A. Adya, D. Myers, J. Howell, J. Elson, C. Meek, V. Khemani, S. Fulger, P. Gu, L. Bhuvanagiri, J. Hunter *et al.*, "Slicer: Auto-sharding for datacenter applications." in *OSDI*, 2016, pp. 739–753.

[8] C. Curino, E. Jones, Y. Zhang, and S. Madden, "Schism: a workload-driven approach to database replication and partitioning," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 48–57, 2010.

[9] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[10] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." in *HotCloud'10*, vol. 10, no. 10-10, 2010, p. 95.

[11] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: a scalable, high-performance distributed file system," in *OSDI*, 2006, pp. 307–320.

[12] B. Trushkowsky, P. Bodík, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "The scads director: Scaling a distributed storage system under stringent performance requirements," in *FAST*, 2011, pp. 12–12.

[13] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "Improving MapReduce performance through data placement in heterogeneous Hadoop clusters," in *IPDPSW*, 2010, pp. 1–9.

[14] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Workload analysis and demand prediction of enterprise data center applications," in *IISWC*, 2007, pp. 171–180.

[15] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[16] M. Akdere, U. Çetintemel, M. Riondato, E. Upfal, and S. B. Zdonik, "Learning-based query performance modeling and prediction," in *ICDE*, 2012, pp. 390–401.

[17] A. Pavlo, C. Curino, and S. Zdonik, "Skew-aware automatic database partitioning in shared-nothing, parallel OLTP systems," in *SIGMOD*, 2012, pp. 61–72.

[18] P. Barford and M. Crovella, "Generating representative web workloads for network and server performance evaluation," in *SIGMETRICS '98/PERFORMANCE '98*, 1998, pp. 151–160.

[19] M. Newman, "Power laws, pareto distributions and zipf's law," *Contemporary Physics*, vol. 46, no. 5, pp. 323–351, 2005.

[20] J. Dilley, R. Friedrich, T. Jin, and J. Rolia, "Web server performance measurement and modeling techniques," *Performance evaluation*, vol. 33, no. 1, pp. 5–26, 1998.

[21] V. V. Panteleenko and V. W. Freeh, "Web server performance in a wan environment," in *ICCCN*, 2003, pp. 364–369.

[22] K. Sripanidkulchai, B. Maggs, and H. Zhang, "An analysis of live streaming workloads on the internet," in *ACM SIGCOMM*, 2004, pp. 41–54.

[23] G. Urdaneta, G. Pierre, and M. Van Steen, "Wikipedia workload analysis for decentralized hosting," *Computer Networks*, vol. 53, no. 11, pp. 1830–1845, 2009.

[24] J. Wilkes, "Traveling to Rome: QoS Specifications for Automated Storage System Management," in *IWQoS*, 2001, pp. 75–91.

[25] A. Middleton and A. Chala, "Hpcc systems: Introduction to hpcc (high-performance computing cluster)," *White paper, LexisNexis Risk Solutions*, 2011.

[26] D. Wiers, "Dstat: Versatile resource statistics tool," 2017. http://dag.wiee.rs/home-made/dstat/

[27] B. Gregg, "perf-tools," 2017. https://github.com/brendangregg/perf-tools

[28] A. Leff, J. L. Wolf, and P. S. Yu, "Replication algorithms in a remote caching architecture," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 11, pp. 1185–1204, 1993.

[29] S. Zaman and D. Grosu, "A distributed algorithm for the replica placement problem," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 9, pp. 1455–1468, 2011.

[30] N. Laoutaris, O. Telelis, V. Zissimopoulos, and I. Stavrakakis, "Distributed selfish replication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 12, pp. 1401–1413, 2006.

[31] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild *et al.*, "Spanner: Googles globally distributed database," *ACM Transactions on Computer Systems (TOCS)*, vol. 31, no. 3, pp. 1–22, 2013.

[32] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *MSST 2010*. IEEE, 2010, pp. 1–10.

[33] R. Cattell, "Scalable sql and nosql data stores," *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2011.

[34] P. Scheuermann, G. Weikum, and P. Zabback, "Data partitioning and load balancing in parallel disk systems," *The International Journal on Very Large Data Bases (VLDB)*, vol. 7, no. 1, pp. 48–66, 1998.

[35] D. Kossmann, "The state of the art in distributed query processing," *ACM Computing Surveys*, vol. 32, no. 4, pp. 422–469, 2000.

[36] K. R. Krish, A. Khasymski, A. R. Butt, S. Tiwari, and M. Bhandarkar, "AptStore: Dynamic storage management for hadoop," in *CloudCom 2013*, vol. 1, 2013, pp. 33–41.

[37] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris, "Scarlett: coping with skewed content popularity in mapreduce clusters," pp. 287–300, 2011.

[38] M. Y. Eltabakh, Y. Tian, F. Özcan, R. Gemulla, A. Krettek, and J. McPherson, "Cohadoop: flexible data placement and its exploitation in hadoop," *Proceedings of the VLDB Endowment*, vol. 4, no. 9, pp. 575–585, 2011.