DARE: Adaptive Data Replication for Efficient Cluster Scheduling

Cristina L. Abad*‡, Yi Lu†, Roy H. Campbell*
*Department of Computer Science
†Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Email: cabad,yilu4,rhc@illinois.edu

***Abstract*—Placing data as close as possible to computation is a common practice of data intensive systems, commonly referred to as the data locality problem. By analyzing existing production systems, we confirm the benefit of data locality and find that data have different popularity and varying correlation of accesses. We propose DARE, a distributed adaptive data replication algorithm that aids the scheduler to achieve better data locality. DARE solves two problems, how many replicas to allocate for each file and where to place them, using probabilistic sampling and a competitive aging algorithm independently at each node. It takes advantage of existing remote data accesses in the system and incurs no extra network usage. Using two mixed workload traces from Facebook, we show that DARE improves data locality by more than 7 times with the FIFO scheduler in Hadoop and achieves more than 85% data locality for the FAIR scheduler with delay scheduling. Turnaround time and job slowdown are reduced by 19% and 25%, respectively.**

***Keywords*-MapReduce, replication, scheduling, locality.**

I. Introduction

Cluster computing systems, such as MapReduce [1], Hadoop [2] and Dryad [3], together with fault-tolerant distributed data storage [4, 5], have become a popular framework for data-intensive applications. Large clusters consisting of tens of thousands of machines [6] have been built for web indexing and searching; small and mid-size clusters have also been built for business analytics and corporate data warehousing [7–9]. In clusters of all sizes, throughput and job completion time are important metrics for computation efficiency, which determines cost of data centers and user satisfaction [6, 10].

Placing data as close as possible to computation is a common practice of data-intensive systems, referred to as the *data locality* problem. Current cluster computing systems use uniform data replication to (a) ensure data availability and fault tolerance in the event of failures [11–16], (b) improve data locality by placing a job at the same node as its data [1, 3, 17], and (c) achieve load balancing by distributing work across the replicas.

‡Also affiliated with Facultad de Ingeniería en Electricidad y Computación (FIEC), Escuela Superior Politécnica del Litoral (ESPOL), Campus Gustavo Galindo, Km 30.5 Vía Perimetral, Guayaquil–Ecuador.

Data locality is an important problem as it significantly affects system throughput and job completion times [6, 10]. The goal of this paper is to improve data locality in cluster computing systems using adaptive replication with low system overhead. It was suggested in a recent study [18] that the benefit of data locality might disappear as bandwidth in data centers increases. However, we found the difference between disk and network bandwidths remain significant in reality, as illustrated in Section II, and the use of virtualized clusters and the concerns with energy consumption of network usage make disk reads remain preferable. Furthermore, the amount of data being processed in data centers, both in business and science, keeps growing at an enormous pace [19].

Uniform data replication and placement are used in current implementations of MapReduce systems (e.g., Hadoop). Applications rely on the scheduler to optimize for data locality. However, we found that there is significant difference in data popularity and considerable correlation among accesses to different files in a cluster log from Yahoo!. We elaborate on the observation in Section III. Similar observation of the skew in data popularity in a large Dryad production cluster supporting Microsoft's Bing was presented in [6].

There are two ways in which a well-designed data replication and placement strategy can improve data locality:

1. Popular data are assigned a larger number of replicas to improve data locality of concurrent accesses; we call it the **replica allocation problem**.
2. Different data blocks accessed concurrently are placed on different nodes to reduce contention on a particular node; we call it the **replica placement problem**.

Scarlett [6] addresses the replica allocation problem at fixed epochs using a centralized algorithm. However, the choice of epochs depends on the workload, and can vary from cluster to cluster and across time periods. While workload characteristics may remain similar in a cluster supporting a single application, they can vary significantly in environments with multiple applications. A dynamic algorithm that adapts to changes in workload is hence preferable.

A. Our Approach

We propose DARE, a distributed data replication and placement algorithm that adapts to the change in workload.

We assume a scheduler oblivious to the data replication policy, such as the first-in, first-out (FIFO) scheduler or the Fair scheduler in Hadoop systems, so our algorithm will be compatible to existing schedulers. We implement and evaluate DARE using the Hadoop framework, Apache's open source implementation of MapReduce. We expand on the details of MapReduce and Hadoop clusters in Section II.

In the current implementation, when local data are not available, a node retrieves data from a remote node in order to process the assigned task, and discards the data once the task is completed. DARE takes advantage of existing remote data retrievals and selects a subset of the data to be inserted into the file system, hence creating a replica without consuming extra network and computation resources.

Each node runs the algorithm independently to create replicas of data that are likely to be heavily accessed in a short period of time. We observe in the Yahoo! log that the popularity of files follows a heavy-tailed distribution. This makes it possible to predict file popularity from the number of accesses that have already occurred: For a heavy-tailed distribution of popularity, the more a file has been accessed, the more future accesses it is likely to receive.

From the point of view of an individual data node, the algorithm comes down to quickly identifying the most popular set of data and creating replicas for this set. Popularity not only means that a piece of data receives a large *number* of accesses, but also a high *intensity* of accesses. We observe that this is the same as the problem of heavy hitter detection in network monitoring: In order to detect flows occupying the largest bandwidth, we need to identify flows that are both *fast* and *large*. In addition, the popularity of data is relative: We want to create replicas for files that are *more* popular than others. Hence algorithms based on a hard threshold of number of accesses do not work well.

We design a probabilistic dynamic replication algorithm with the following features:

1. Each node samples assigned tasks and uses the ElephantTrap [20] structure to replicate popular files in a distributed manner. Experiments on dedicated Hadoop clusters and virtualized EC2 clusters both show more than 7-times improvement of data locality for the FIFO scheduler, and 70% improvement for the Fair scheduler. DARE with the Fair scheduler–which increases locality by introducing a small delay when a job scheduled to run cannot execute a local task; allowing other jobs to launch tasks instead–can lead to locality levels close to 100% for some workloads.
2. Data with correlated accesses are distributed over different nodes as new replicas are created and old replicas expire. This also helps data locality and reduces job turnaround time by 16% in dedicated clusters and 19% in virtualized public clouds. Job slowdown is reduced by 20% and 25%, respectively.
3. The algorithm incurs no extra network usage by taking advantage of existing remote data retrievals. Thrashing is minimized using sampling and a competitive aging algorithm, which produces comparable data locality to a greedy *least recently used* (LRU) algorithm, but with only 50% disk writes of the latter.

The contribution of the paper is two-fold. First, we analyze existing production systems to obtain effective bandwidth, data popularity distributions, and uncover characteristics of access patterns. Second, we propose the distributed dynamic data replication algorithm, which significantly improves data locality and task completion times.

The rest of this paper is organized as follows. We present our motivation in Section II, including a detailed discussion on the effect of data locality in virtualized clusters on public clouds. Section III presents the results of an analysis of data access patterns in a large MapReduce production cluster. Section IV describes the design of our proposed replication scheme. In Section V we present and discuss the evaluation results. Section VI discusses the related work and we summarize the contributions in Section VII.

II. Background and Motivation

A. MapReduce clusters

MapReduce clusters [1, 2] offer a distributed computing platform suitable for data-intensive applications. MapReduce was originally proposed by Google and its most widely deployed implementation, Hadoop, is used by many companies including Facebook, Yahoo! and Twitter [9].

MapReduce uses a divide-and-conquer approach in which input data are divided into fixed size units processed independently and in parallel by *map* tasks, which are executed distributedly across the nodes in the cluster. After the *map* tasks are executed, their output is shuffled, sorted and then processed in parallel by one or more *reduce* tasks.

To avoid the network bottlenecks due to moving data into and out of the compute nodes, a distributed file system typically co-exists with the compute nodes (GFS [21] for Google's MapReduce and HDFS [14] for Hadoop).

MapReduce clusters have a master-slave design for the compute and storage systems. The master file system node handles the metadata operations, while the slaves handle the read/writes initiated by clients. Files are divided into fixed-sized blocks, each stored at a different data node. Files are read-only, but appends may be performed in some implementations. For the sake of simplicity, in this paper we will refer to the components of the distributed file system using the HDFS terminology, where *name node* refers to the master node and *data node* refers to the slave.

MapReduce clusters use a configurable number of replicas per file (three by default). While this replica policy makes sense for availability, it is ineffective for locality and load balancing when access patterns of data are not uniform.

As it turns out, the data access patterns (or popularity distribution) of files in MapReduce clusters are not uniform;

it is common for some files to be much more popular than others (e.g., job configuration files during initial job stages) while others may be significantly unpopular (e.g., old log files rarely processed). For job files, its popularity can be predicted (i.e., launching job creates a hotspot), so a solution adopted in currently implemented systems is to have the framework automatically increase the replication factor for these files [1, 14]. For other cases, the current approach is to manually increase or decrease the number of replicas for a file using organization heuristics based on data access patterns. For example, Facebook de-replicates aged data, which can have a lower number of replicas (as low as one copy) compared to other data [18]. The manual approach described above is not scalable and can be error-prone.

We provide more insight on the data access patterns of MapReduce clusters in Section III.

B. Hadoop in virtualized environments

Hadoop is used on many small and medium sized companies that frequently use third-party virtual clusters instead of in-house Hadoop data centers. Even if these service providers upgrade to higher-end network fabrics, contention still exists for the shared network resource. Additionally, virtually allocated nodes may not be on the same rack, which further stresses the network and adds latency to network communications.

To illustrate this issue, we evaluate the difference in network bandwidth, disk bandwidth, and network round-trip time (RTT), in a 20-node dedicated cluster in the Illinois Cloud Computing Testbed (CCT) [22], and in a 20-node virtual cluster on Amazon's EC2[1]. The configurations of these clusters are described in the evaluation section.

We use `ping`, `hdparm` and `iperf` to evaluate the network latency, disk bandwidth, and network bandwidth, respectively. Tables I and II summarize the results.

Table I
ALL-TO-ALL PING ROUND-TRIP TIMES, FOR A DEDICATED CLUSTER (CCT) AND A VIRTUALIZED CLUSTER IN A PUBLIC CLOUD (EC2).

	Min	Mean	Max	Std. Deviation
CCT	0.01 ms	0.18 ms	2.17 ms	0.34 ms
EC2	0.02 ms	0.77 ms	75.1 ms	3.36 ms

Table II
DISK (READ) AND NETWORK BANDWIDTH IN MB/S.

	Min	Mean	Max	Std. Dev.
CCT disk bandwidth	145.3	157.8	167.0	8.02
CCT network bandwidth	115.4	117.7	118.0	0.65
EC2 disk bandwidth	67.1	141.5	357.9	74.2
EC2 network bandwidth	5.8	73.2	109.9	16.9

As shown in Table I (and in [23]), the RTTs between EC2 instances have much higher variability than non-virtualized machines, making performance difficult to predict.

[1]EC2 (Elastic Compute Cloud) is an infrastructure-as-a-service (IaaS) API for pay-as-you-go virtual machines on Amazon Web Services (AWS).

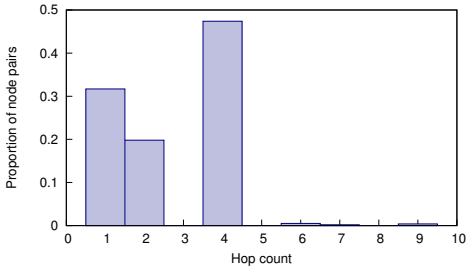


Figure 1. Distribution of the number of hops between any two nodes in an EC2 20-node cluster.

While the CCT and EC2 bandwidth results are not directly comparable because the nodes in each cluster have different hardware characteristics, a key insight is that the ratio $network\,bandwidth/disk\,bandwidth$ is 40% higher (74.6% vs. 51.75%) for the CCT cluster than for the virtualized EC2 cluster. This means that the gain of local reads (vs. remote reads through network) is higher for the EC2 cluster than for the CCT cluster.

There are several things that can affect performance in virtualized public clouds. This includes impact of virtualization on disk and network I/O, other clients sharing the physical nodes, and network topology. With respect to the last issue, many cloud IaaS APIs will allocate nodes for a user in different racks, a practice that provides better load balancing for the provider and better availability for the client, but decreases network bandwidth and increases latency.

We use `traceroute` to infer the inter-node distance of our 20-node EC2 cluster. In our test cluster most nodes are 4-hops apart (see Fig. 1); in an in-house data center of that size all nodes would have been 1 or 2 hops apart.

A study of network performance in the Amazon EC2 cloud was performed by Wang *et al.* [23], who found that processor sharing can cause very unstable network performance in EC2 instances, which lead to large packet delay even when the network is not heavily congested.

III. DATA ACCESS PATTERNS IN PRODUCTION CLUSTERS

We analyze Hadoop logs of a Yahoo! 4000-node production cluster [24] from the second week of January 2010.

In this paper, the term *file* denotes the smallest granularity of data that can be accessed by a MapReduce job. A file is comprised of N fixed-size data *blocks* (of 64 to 256 MBs).

File popularity: The data access distribution, or file popularity, of data in a MapReduce cluster can be nonuniform due to reasons described in Section II. Fig. 2 shows a heavy-tailed distribution in which some files are significantly more popular than others. In this case, the reason is that the cluster is used mainly to perform different types of analysis on a common (time-varying) data set. Similar results were obtained by analyzing the 64-node CCT Hadoop production

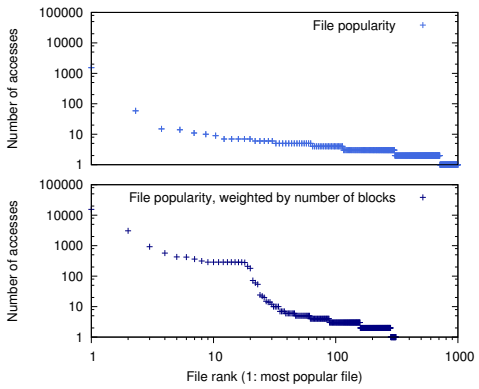


Figure 2. Number of accesses per file. Files are ranked according to their popularity (measured by number of accesses and number of accesses weighted by the number of 128MB blocks in file).

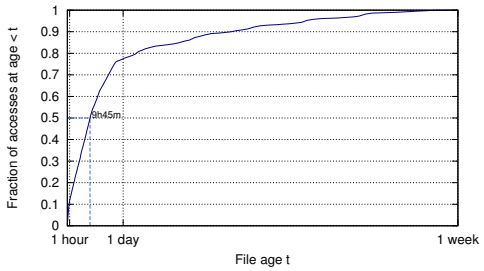


Figure 3. Cumulative distribution function (CDF) of file age at time of access. There is a high temporal correlation in accesses.

cluster, and have previously been observed at Microsoft's Bing [6]. This suggests that uniformly increasing the number of replicas is not an adequate way of improving locality and achieving load balancing.

Duration of the majority of accesses: The cumulative distribution function of the age of a file at the time it is accessed is shown in Fig. 3. We found that around 80% of the accesses of a file occur during its first day of life. A similar graph is presented by Fan *et al.* [25], obtained from the Yahoo! M45 research cluster; they found that most of the accesses of a block occur very shortly after its creation (50% at 1 minute age). In our analysis, files are typically accessed in a larger period of time (50% of accesses occur at almost 10 hours of age). We believe this difference is partly due to the fact that we excluded system files (e.g., job.jar, job.xml, job.split) which are created, accessed, and deleted for every MapReduce job in the system, and also that the M45 cluster is an academic research cluster shared among very different groups and institutions, so data sharing may be less common than in a single-organization cluster.

Duration of popularity bursts: Fig. 4 shows the percentage of time windows that contain 80% or more of the total accesses. In other words, for every file we found the smallest number of consecutive time slots (each slot is one hour long) that contain 80% or more of the total accesses for that file. The spike at window 121 shows that most files are accessed daily. Fig. 5 shows the same analysis, for a sample day (day 2 in the data set). Within a day, most significant file accesses lie withing one hour.

IV. Design

Our goal is to design an adaptive replication scheme that seeks to increase data locality by replicating "popular" data while keeping a minimum number of replicas for unpopular data. Additionally, we want our scheme to: (a) dynamically adapt to changes in file access patterns, (b) use a replication budget to limit the extra storage consumed by the replicas, and (c) impose a low network overhead.

Role of scheduler in improving locality: Current data-intensive frameworks rely on the job scheduler to optimize for data locality [1, 14, 26] on top of a uniform data replication policy implemented by the distributed file system. Schedulers are able to increase locality by cleverly placing tasks based on proximity to their data. For popular data, especially files with concurrent accesses [6], DARE helps the scheduler improve locality by providing more choices where the tasks can run.

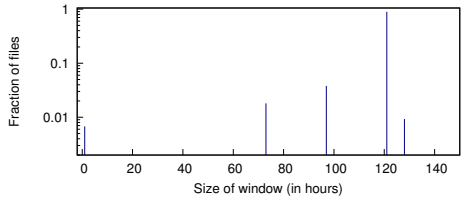
A. Greedy approach

First, we design a greedy reactive scheme that incurs no extra network traffic by taking advantage of existing data retrievals as follows: when a map task is launched, its data can be local to the node or remote to the node (i.e., located in a different node). In the cases of remote data, the original MapReduce framework fetches and processes the data, without keeping a local copy for future tasks. With DARE, when a map task processes remote data, the data are inserted into the HDFS at the node that fetched it. By doing this, the number of replicas for the data is automatically increased by one, without incurring explicit network traffic. Data are replicated at the granularity of a block.

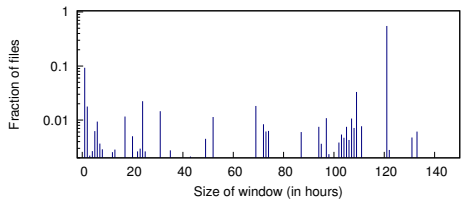
Replication with a budget: DARE uses a replication budget to limit the extra storage consumed by the dynamically replicated data. The budget is configurable, but a value between 10% and 20% is reasonable. An analysis of the sensitivity of this parameter is presented in Section V-D.

Aging and replica eviction policy: When the storage space assigned to dynamically created replicas is not infinite, an eviction mechanism is needed. Traditional eviction schemes include *least recently used* (*LRU*) and *least frequently used* (*LFU*). Choice between LRU and LFU should be made after profiling typical workloads.

Algorithm 1 presents the greedy approach, with a budget and LRU eviction (with lazy block deletion at idle times).

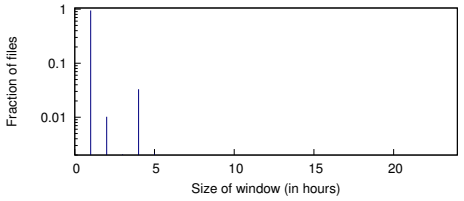


(a) All accesses are weighted equally

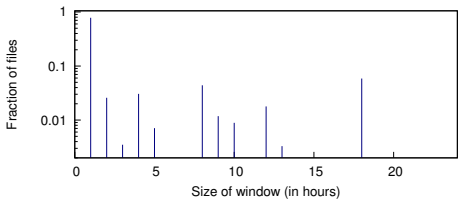


(b) Each entry is weighted by the number of accesses

Figure 4. Percentage of time window/slots with 80%+ of the accesses for each file; only "big files" (80% or more of the total accesses) are considered; system (job related) files are excluded.



(a) All accesses are weighted equally



(b) Each entry is weighted by the number of accesses

Figure 5. Percentage of time window/slots with 80% or more of the accesses for each file, within day 2 of the Yahoo! data set; only "big files"; system files are excluded from analysis.

Algorithm 1 Greedy approach

```
Dynamic replication logic at a node/slave
if a non-data-local map task is scheduled then
    if replication budget will be exceeded then
        Call markBlockForDeletion(block)
    end if
    Insert block in local data node
    Add block to set of dynamically replicated blocks
    Increase budget use by number of bytes in block
end if

function markBlockForDeletion(evicting)
repeat
    // Least recently used victim is selected
    victim ← front of blocksInUsageOrder queue
    if victim belongs to same file as evicting then
        continue
    else
        Remove victim from blocksInUsageOrder
        Set done
    end if
    Mark victim for lazy deletion
    return victim
until done
// NOTE: blocksInUsageOrder queue is refreshed on every
// read; blocks are inserted in tail and removed from front.
```

B. Probabilistic approach

A problem that may arise from inadequate eviction polices is *thrashing*. In this context, thrashing is a high rate of replica creation and eviction. To add stability and prevent thrashing, we modify the algorithm so that blocks are not dynamically replicated immediately after a remote read, but rather they are replicated with a probability p. Our algorithm is an adaptation of the ElephantTrap [20], a mechanism to identify the largest flows on a network link. A coin is tossed to decide whether to replicate the block and also whether to update the circular list that keeps tracks of block accesses.

The greedy approach assumes that any nonlocal data access originated by a remote map task is worth replicating. Since jobs with a small number of map tasks tend to achieve poor locality [10], the greedy approach unnecessarily replicates unpopular data. By adopting a probabilistic approach (see Algorithm 2), we are able to ignore most unpopular nonlocal accesses while replicating popular data.

While our approach has some similarities with caching, it is considered to be a replication scheme because the knowledge of dynamically replicated blocks is transmitted to the name node (during a heartbeat), so that this information will be made available to the scheduler and other users of the file system, which in turn can use it to achieve higher locality levels. Replicas created by DARE are first-order replicas and as such they also contribute to increasing availability of the data in the presence of failures.

Aging and replica eviction policy: A probabilistic approach is also applied to the aging mechanism; instead of refreshing (updating) the number of block accesses per dynamically replicated block on each access, we do it with probability p. When the budget is reached and it is decided that a nonlocal access will trigger a replication, the algorithm iterates through the list of dynamically replicated blocks, reducing their access count by half each time. This is to quickly evict files with decreasing popularity and prevent new popular files from being evicted too soon. The iteration through the list stops when a victim has been found and marked for deletion (*access count* $<$ *threshold*) or when we have iterated through the whole list. Blocks marked for deletion are lazily removed to avoid conflicting with other operations.

Section V-D provides an analysis on the sensitivity of the *budget*, *threshold*, and p parameters.

Algorithm 2 Probabilistic approach

```
Dynamic replication logic at a node/slave
if a map task is scheduled then
    Generate a random number, r ∈ (0, 1)
    if r < p then
        if map task is non-data-local then
            if replication budget will be exceeded then
                Call markBlockForDeletion(block)
                if return value of call is null then
                    // Couldn't find a block to evict; will not replicate
                    return
                end if
            end if
            Insert block in local data node
            Add block to (circular) list of dynamically replicated blocks,
            right before evictionPointer
            blocks2accessCount[block] ← 0
            Increase budget use by number of bytes in block
        else // map task is data-local
            Increment blocks2accessCount[block]
        end if
    end if
end if
```

```
function markBlockForDeletion(evicting)
victim ← block pointed by evictionPointer
while blocks2accessCount[victim] ≥ threshold or evictionPointer has
not gone through entire list of dynamically replicated blocks do
    oldAccessCount ← blocks2accessCount[victim]
    blocks2accessCount[victim] ← oldAccessCount /2
    Increment evictionPointer
end while
if victim belongs to same file as evicting or
blocks2accessCount[victim] ≥ threshold then
    return null
else
    Remove victim from blocks2accessCount,
        and from list of dynamically replicated blocks
end if
Mark victim for lazy deletion
return victim
```

V. Experimental Evaluation

We present the evaluation and analysis of DARE with respect to: system and user metrics, sensitivity to the parameters, and uniformity of the replica placement. We also evaluate DARE in a virtualized public cloud.

A. Methodology

We evaluate DARE in two cluster environments: a private 20-node cluster in the Illinois Cloud Computing Testbed (CCT) [22] and a 100-node cluster running on EC2 small instances. We decided to use small instances–the default in EC2–because they provide higher I/O performance per dollar (they use all available disk bandwidth when no other VMs on the host are using it [26]). Table III summarizes the configuration of the clusters.

Implementation: We implemented the adaptive replication scheme in Hadoop 0.21.1, which had to be modified to: (a) insert replica blocks in the HDFS triggered by remote-data accesses, (b) tolerate over-replicated blocks created by the dynamic replication mechanism, and (c) add configuration parameters (*p*, *threshold* and *budget*).

Table III
Configuration of the test clusters.

	CCT	EC2
Type of cluster	Dedicated, single rack	Virtual (m1.small instances)
Nodes	1 master, 19 slaves	1 master, 99 slaves
RAM (per node)	16 GB	1.7 GB
Cores (per node)	2 Quad Core	1 virtual core with 2 EC2 compute units
Storage (per node)	2 TB	160 GB
Platform	64-bit	32-bit
Network	Gigabit Ethernet	Moderate I/O performance[2]
Operating system	CentOS release 5.5	Fedora release 8

Data structures were added to implement the eviction policies (LRU and ElephantTrap), which had to be protected through synchronization mechanisms. INodes were modified to contain information about which file they belong to, so that we can avoid choosing a victim belonging to the same file (i.e., has the same popularity) as the evicting replica.

The `DataNodeProtocol` was modified to add the `DNA_DYNREPL` operation, to receive a dynamically replicated block and update the corresponding data structures.

At the code level, the implementation required modifying the code of 12 classes, and adding 228 lines of code (including logging code for debugging).

Scheduler: Our scheme is scheduler agnostic, but different schedulers make different choices when running tasks, so we evaluate it using Hadoop's FIFO scheduler (default) and the Fair scheduler. The Fair scheduler uses the delay scheduling technique [10] to improve data locality by applying a small delay to map tasks that otherwise would run nonlocally.

Workload: For each test, we ran 500 jobs synthesized from a Facebook 600-machine production cluster trace. The job trace and replay scripts were published by Chen *et al.*, as part of their Statistical Workload Injector for MapReduce (SWIM) [27]. We chose two sets of consecutive jobs that represent two common Hadoop workloads: *wl1* (jobs 0−499) contains a long sequence of small jobs while *wl2* (jobs 4800−5299) contains a pattern of small jobs after large jobs. The FIFO scheduler produces lower completion times for *wl1* because of smaller variance in job sizes and the Fair scheduler produces lower completion times for *wl2* [27]. Fig. 6 shows the file access distribution used in the experiments.

Metrics: We evaluate DARE with respect to *system metrics* and *performance metrics*. Our main system metric is data locality (i.e., map task locality). Data locality is one of the most important goals in the design of MapReduce systems [1]; increases in data-locality mean reduced network traffic in data centers.

[2]As reported by Amazon in: http://aws.amazon.com/ec2/instance-types/

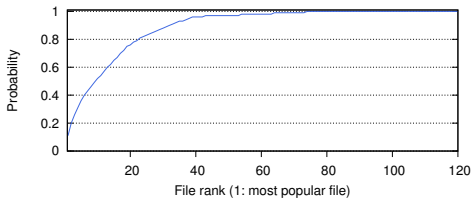


Figure 6. Access pattern (CDF) used in the experiments.

With respect to user metrics, we evaluate both *turnaround time* and *slowdown* which together show the effect of the dynamic replication mechanism on job performance.

The **geometric mean of the turnaround time (GMTT)** measures how long a job takes from its submission to completion. Previous studies have used this metric [17, 28] instead of the average turnaround time because the latter is dominated by long jobs. The GMTT is defined as follows:

$$GMTT = \left(\prod_{k \in K} TT_k \right)^{1/|K|}, \quad (1)$$

where TT_k is the turnaround time of job k, K is the set of jobs in the workload and $|K|$ is the number of jobs.

The **slowdown** of a job is defined as its running time on a loaded system divided by the running time on a dedicated system [29]; for the case of Hadoop, we calculate the latter as the running time (*job completion time − job arrival time*) in a completely free Hadoop cluster with 100% data locality.

To evaluate the **uniformity of the replica placement** with DARE, we assign a popularity value to each file based on its access count for each workload. We calculate the *popularity index (PI)* of data node i as $\sum_j blockSize_j \times blockPopularity_j$, for every block j in i. In this way, we obtain a distribution of the PI_i of the nodes in the system. As a measure of the uniformity of this distribution, we use the *coefficient of variation* (c_v), a normalized measure of the dispersion of a probability distribution. The more uniform the distribution is, the smaller its c_v. The c_v is defined as the ratio of the standard deviation σ to the mean μ: $c_v = \sigma/|\mu|$.

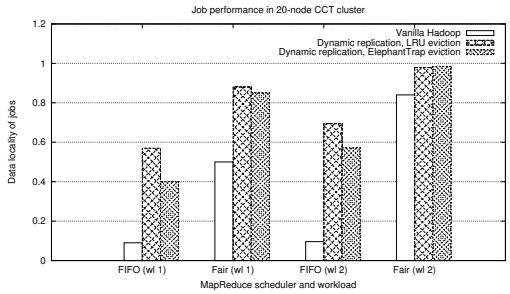
B. Performance: System metrics

Fig. 7a shows that DARE achieves significant improvement in data locality: as much as 7 times improvement for the FIFO scheduler and 70% for the Fair scheduler. When the job workload favors the Fair scheduler (*wl2* in Fig. 7a), the locality is quite high even without dynamic replication (83% in our tests) , but even for this case our scheme is able to bring locality up leading to locality levels close to 100%. The Fair scheduler is able to increase locality at the cost of a small delay for tasks that cannot run locally, hence increases turnaround time.

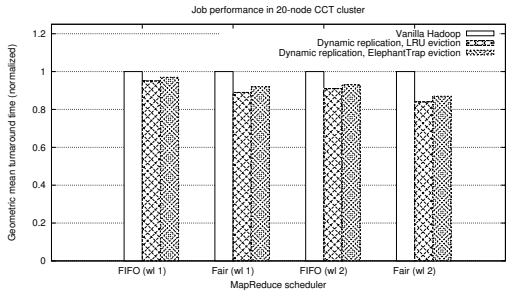
Data locality is an important metric because it reduces stress on the network fabric, which is desirable in data-intensive scenarios since network fabrics are frequently over-subscribed [30], especially across racks. Furthermore, the reduction in network traffic due to increased locality can be exploited by current and future energy proportional switches and network designs to reduce energy consumption [31, 32].

C. Performance: User metrics

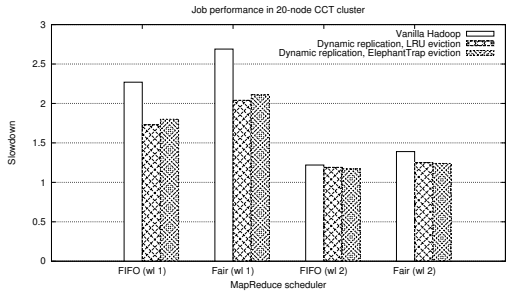
Fig. 7b and 7c show that DARE achieves up to 16% reduction in turnaround time (GMTT) and 20% in slowdown in the CCT tests. This improvement is similar to those of Scarlett [6] using a non-dynamic epoch-based scheme.



(a) Data locality (higher is better)



(b) Geometric mean turnaround time (lower is better)



(c) Mean slowdown (lower is better)

Figure 7. Data locality and performance (GMTT and slowdown) of DARE in a dedicated 20-node cluster, for two different replica eviction policies: a greedy least recently used (LRU) and a probabilistic one (ElephantTrap; $p = 0.3; threshold = 1; budget = 0.2$). Two different job workloads are evaluated; one that favors the FIFO scheduler (*wl1*), and one that favors the Fair scheduler (*wl2*).

We also evaluate reduction to map task completion time with the second workload. The mean reduction is 12% and 11% for the FIFO and Fair schedulers, respectively. We believe this is due to a mixture of input-bound and output-bound [27] tasks in the trace. Dynamic replication does

not expedite output-bound tasks, whose turnaround time is dominated by output processing. We plan to investigate the effect of different tasks further in future work.

D. Sensitivity analysis

We vary the values of different parameters to observe their effects on data locality. The results can be seen in Fig. 8 and Fig. 9. DARE with the ElephantTrap replica eviction mechanism is not too sensitive to changes in the *threshold* parameter. Increasing the *threshold* leads to a slow decrease in locality and a a slow increase in the number of replicas being created. A higher *threshold* value leads to some replicas being evicted too early, which has a detrimental effect in both locality and the number of replicas being created. The effect is, however, not too pronounced due to the way the eviction data structure is iterated and the heavy-tailed nature of the accesses; as long as the most popular object is not evicted, other evictions have a less significant incidence in the data locality achieved.

Increasing the *budget* leads to minor increase in data locality while observing a more noticeable reduction in the number of replicas created. The reason for this is the smaller the *budget*, the smaller the number of different dynamically replicated blocks that a data node can keep which leads to a higher eviction rate that in turns leads to some popular files being evicted too early. However, the effect that this has on locality is small because even small budgets allow DARE to replicate the most popular files, and achieve the most benefit of the dynamic replication mechanism.

Increasing the ElephantTrap sampling probability p leads to a higher data locality, at the cost of a higher number of blocks being replicated. In our experiments (see Fig. 8a), setting p to a value between 0.2 and 0.3 leads to the most gain in locality without incurring in excesive replica thrashing.

E. Performance in virtualized public clouds

We evaluate DARE in a 100-node EC2 virtual cluster (see Fig. 10). Our results show that for comparable improvement in locality, the improvement in GMTT and mean slowdown was more significant in the EC2 cluster (19% and 25%, respectively). We believe this is due to the fact that the ratio $network\,bandwidth/disk\,bandwidth$ is higher for the CCT cluster than for the virtualized EC2 cluster, due to the reasons discussed in Section II-B.

F. Uniformity of the replica placement

We evaluate DARE's placement of replicas using the *coefficient of variation* (c_v) of the distribution of the *popularity indices* of the data nodes, both before and after using DARE. Fig. 11 shows that the replica placement with DARE is less dispersed (smaller c_v) with respect to the popularity index of the nodes than with the default Hadoop replica placement policy. The parameters used in this test are: FIFO scheduler, wl1, probabilistic version of DARE ($budget$ = 20%, $threshold$ = 1). We observe that the popularity indices gain significant uniformity at $p = 0.2$.

VI. Related work

Data replication in distributed file systems is a technique commonly used to improve data *availability* [11–14, 16] in the presence of failures. Replication has also been studied in the context of improving *response time* [33], *read throughput* [14, 16], and less frequently, *job performance* [6, 17].

Our contributions are: (i) analysis of data access patterns of a production MapReduce cluster, (ii) analysis of the effect of improving locality through replication for dedicated clusters and for virtualized public clouds, and (iii) design, implementation, and evaluation of a dynamic replication mechanism for the replica placement problem, that is able to adapt to popularity changes.

Only two recent works deal with the specific case of dynamic replication in MapReduce clusters: CDRM [12] and Scarlett [6]. In [12], Wei *et al.* presented CDRM, a "cost-effective dynamic replication management scheme for cloud storage cluster". CDRM is a replica placement scheme for Hadoop, that aims to improve file availability by centrally determining the ideal number of replicas for a file, and an adequate placement strategy based on the *blocking probability*. The effects of increasing locality are not studied.

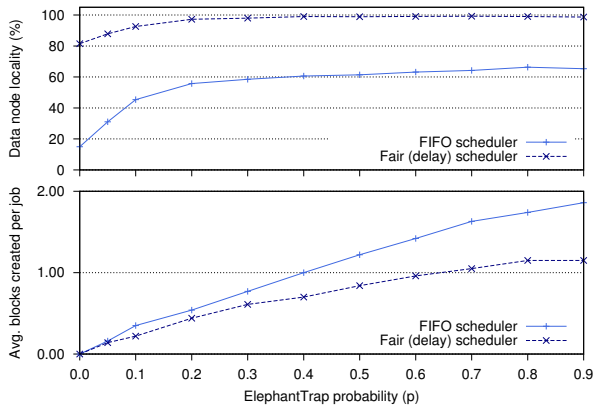
Parallel to our work, Ananthanarayanan *et al.* [6] proposed Scarlett, an off-line system that replicates blocks based on their observed probability in a previous epoch. Scarlett computes a replication factor for each file and creates budget-limited replicas distributed among the cluster with the goal of minimizing hotspots. Replicas are aged to make space for new replicas.

While Scarlett uses a *proactive* replication scheme that periodically replicates files based on predicted popularity, we proposed a *reactive* approach that is able to adapt to popularity changes at smaller time scales and can help alleviate recurrent as well as nonrecurrent hotspots.

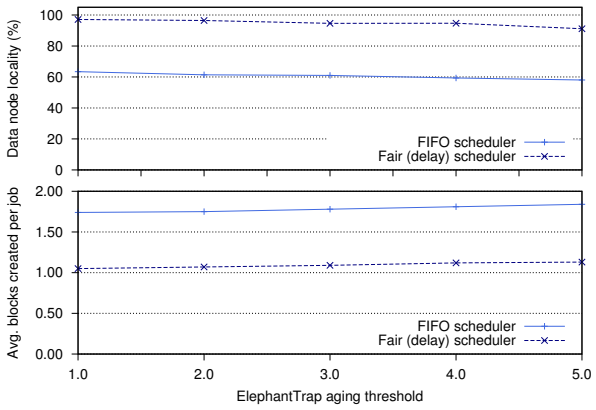
Zaharia's *et al.* delay scheduling [10] increases locality by delaying for a small amount of time a map task that–without the delay–would have run nonlocally. DARE is scheduler-agnostic and can work together with this and other scheduling techniques that try to increase locality. The delay scheduling technique is currently part of Hadoop's Fair scheduler, one of the two schedulers used in our evaluations.

VII. Conclusion

We found that data access patterns in MapReduce clusters are heavy-tailed, with some files being considerably more popular than others. For nonuniform data access patterns, current replication mechanisms that replicate files a fixed number of times are inadequate and can create suboptimal task locality and hinder the performance of MapReduce clusters. We proposed DARE, an adaptive data replication

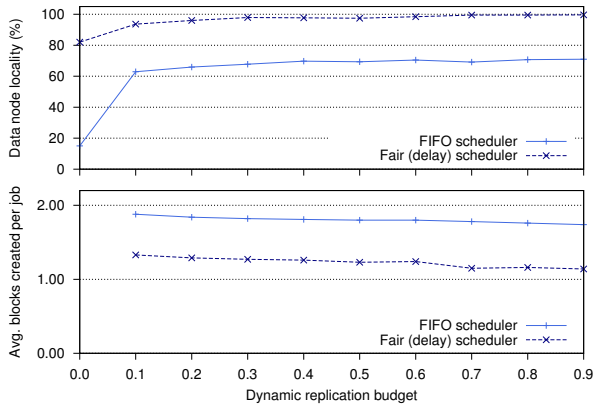


(a) Effect of changing replication prob.; $threshold = 1, budget = 0.20$

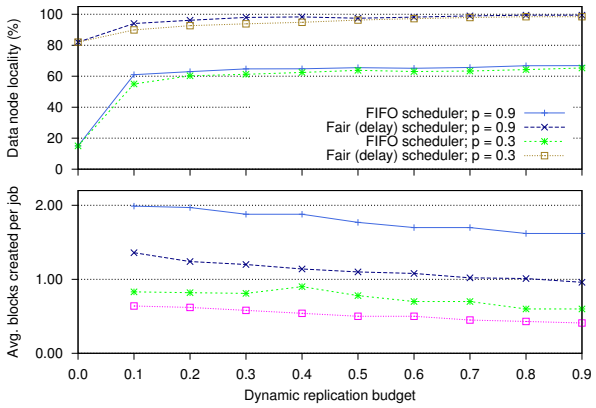


(b) Effect of changing eviction threshold; $p = 0.90, budget = 0.50$

Figure 8. Effect on locality (top) and number of blocks dynamically replicated (bottom) when using DARE with probabilistic (ElephantTrap) eviction policy, for different values of the p and $threshold$ parameters, for the full set of jobs in the second workload (*wl2*).

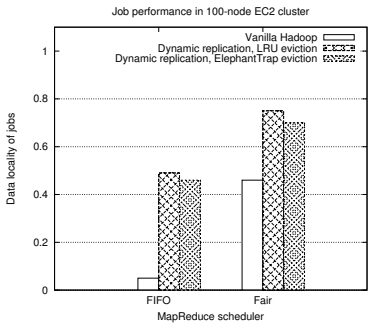


(a) DARE with LRU eviction

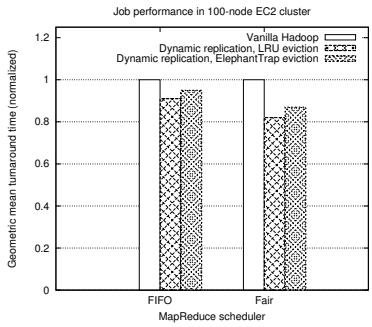


(b) DARE with ElephantTrap eviction; $threshold = 1$

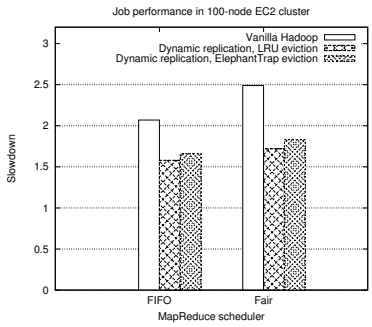
Figure 9. Effect on locality (top) and number of blocks dynamically replicated (bottom) when using DARE with greedy (LRU, left) and probabilistic (ElephantTrap, right) eviction policy, for different $budget$ values, for the full set of jobs in the second workload (*wl2*).



(a) Data node locality (higher is better)



(b) Geometric mean turnaround time (lower is better)



(c) Mean slowdown (with respect to job execution on a dedicated cluster; lower is better)

Figure 10. Performance of DARE in a virtualized 100-node cluster in the Amazon Web Services EC2 public cloud, for two different replica eviction policies: a greedy least recently used (LRU) and a probabilistic one (ElephantTrap; $p = 0.3; threshold = 1; budget = 0.2$), and one 500-job long workload (*wl1* in the CCT experiments).

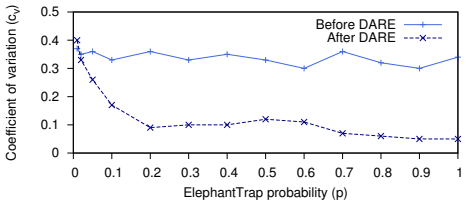


Figure 11. Uniformity of replica placement, before dynamic replication (without DARE) and after dynamic replication (i.e., at the end of the 500-job run, with DARE enabled during the run). Smaller c_v is better.

mechanism that can improve data locality by more than 7-times for a FIFO scheduler and 70% for the Fair scheduler, without incurring in extra networking overhead. Turnaround time and slowdown are improved by 19% and 25%, respectively. Furthermore, our scheme is scheduler-agnostic and can be used in parallel with other schemes that aim to improve locality.

ACKNOWLEDGMENT

This paper is based on research sponsored by the Air Force Research Laboratory and the Air Force Office of Scientific Research, under agreement FA8750-11-2-0084. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. We thank Amazon for sposonring our EC2 experiments through *AWS in Education*.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. USENIX Symp. Operating Syst. Design and Implementation (OSDI)*, 2004, pp. 137–150.

[2] "Apache Hadoop," Jun. 2011. [Online]. Available: http://hadoop.apache.org

[3] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proc. Eur. Conf. Comput. Syst. (EuroSys)*, 2007, pp. 59–72.

[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 Trans. Comput. Syst.*, vol. 26, no. 2, 2008.

[5] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *SIGOPS Operating Syst. Rev.*, vol. 44, no. 2, 2010.

[6] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris, "Scarlett: Coping with skewed popularity content in MapReduce clusters," in *Proc. Eur. Conf. Comput. Syst. (EuroSys)*, 2011.

[7] J. Dixon, "Pentaho, Hadoop, and data lakes," blog, Oct. 2010. [Online]. Available: http://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes

[8] J. Zuanich, "Twitter analytics lead, Kevin Weil, interviewed," Cloudera Blog, Sep. 2010. [Online]. Available: http://www.cloudera.com/blog/2010/09

[9] "Powered-by – Hadoop Wiki," Jul. 2011. [Online]. Available: http://wiki.apache.org/hadoop/PoweredBy

[10] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *Proc. Eur. Conf. Comput. Syst. (EuroSys)*, 2010.

[11] M. Satyanarayanan, "A survey of distributed file systems," *Annu. Rev. of Comput. Sci.*, vol. 4, pp. 73–104, 1990.

[12] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, and D. Feng, "CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster," in *Proc. IEEE Int'l Conf. Cluster Computing (CLUSTER)*, 2010, pp. 188–196.

[13] J. Xiong, J. Li, R. Tang, and Y. Hu, "Improving data availability for a cluster file system through replication," in *Proc. IEEE Int'l Symp. Parallel and Distrib. Processing (IPDPS)*, 2008.

[14] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *Proc. IEEE Symp. Mass Storage Syst. and Technologies (MSST)*, 2010.

[15] D. Ford, F. Labelle, F. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *Proc. USENIX Symp. Operating Syst. Design and Implementation (OSDI)*, 2010.

[16] J. Terrace and M. J. Freedman, "Object storage on CRAQ: High-throughput chain replication for read-mostly workloads," in *Proc. USENIX Annu. Tech. Conf. (USENIX)*, 2009.

[17] M. Tang, B.-S. Lee, X. Tang, and C.-K. Yeo, "The impact of data replication on job scheduling performance in the data grid," *Future Generation Comput. Syst.*, vol. 22, no. 3, 2006.

[18] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Disk-locality in datacenter computing considered irrelevant," in *Proc. USENIX Workshop on Hot Topics in Operating Syst. (HotOS)*, 2011.

[19] T. Hey, S. Tansley, and K. Tolle, Eds., *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, 2009.

[20] Y. Lu, B. Prabhakar, and F. Bonomi, "ElephantTrap: A low cost device for identifying large flows," in *Proc. IEEE Symp. High-Performance Interconnects*, ser. HOTI, 2007.

[21] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in *Proc. ACM Symp. Operating Syst. Principles (SOSP)*, 2003, pp. 29–43.

[22] "CCT: Illinois cloud computing testbed," (Last accessed: Jul 7, 2011). [Online]. Available: http://cloud.cs.illinois.edu

[23] G. Wang and T. S. E. Ng, "The impact of virtualization on network performance of Amazon EC2 data center," in *Proc. Conf. Inf. Commun. (INFOCOM)*, 2010.

[24] "Yahoo! Webscope dataset ydata-hdfs-audit-logs-v1_0," direct distribution, Feb. 2011, http://research.yahoo.com/Academic_Relations.

[25] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson, "DiskReduce: RAID for data-intensive scalable computing," in *Proc. Annu. Workshop Petascale Data Storage (PDSW)*, 2009, pp. 6–10.

[26] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Proc. USENIX Symp. Operating Syst. Design and Implementation (OSDI)*, 2008, pp. 29–42.

[27] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The case for evaluating MapReduce performance using workload suites," in *Proc. IEEE Int'l Symp. Modeling, Analysis & Simulation of Comput. Telecomm. Syst. (MASCOTS)*, 2011.

[28] W. Cirne and F. Berman, "When the herd is smart: Aggregate behavior in the selection of job request," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 2, 2003.

[29] D. Feitelson and L. Rudolph, "Metrics and benchmarking for parallel job scheduling," *Job Scheduling Strategies for Parallel Processing, LNCS*, vol. 1459, pp. 1–24, 1998.

[30] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *Proc. ACM Conf. Internet Measurement (IMC)*, 2009.

[31] P. Mahadevan, S. Banerjee, and P. Sharma, "Energy proportionality of an enterprise network," in *Proc. ACM Workshop on Green Netw.*, 2010, pp. 53–60.

[32] R. Bolla, R. Bruschi, F. Davoli, and F. Cucchietti, "Energy efficiency in the future Internet: A survey of existing approaches and trends in energy-aware fixed network infrastructures," *IEEE Commun. Surveys Tuts.*, vol. 13, no. 2, pp. 223–244, 2011.

[33] T. Loukopoulos and I. Ahmad, "Static and adaptive data replication algorithms for fast information access in large distributed systems," in *Proc. Int'l Conf. Distrib. Comput. Syst. (ICDCS)*, 2000.