A Decentralized Replica Placement Algorithm for Edge Computing

Atakan Aral, *Member, IEEE,* and Tolga Ovatman, *Member, IEEE*

***Abstract*—As the devices that make up the Internet become more powerful, algorithms that orchestrate cloud systems are on the verge of putting more responsibility for computation and storage on these devices. In our current age of Big Data, dissemination and storage of data across end cloud devices is becoming a prominent problem subject to this expansion. In this paper, we propose a distributed data dissemination approach that relies on dynamic creation/replacement/removal of replicas guided by continuous monitoring of data requests coming from edge nodes of the underlying network. Our algorithm exploits geographical locality of data during the dissemination process due to the plenitude of common data requests that stem from the clients within a close proximity. Our results using both real-world and synthetic data demonstrate that a decentralized replica placement approach provides significant cost benefits compared to client side caching that is widely used in traditional distributed systems.**

***Index Terms*—Data Replication, Replica Placement, Replica Discovery, Facility Location, Cloud Computing, Edge Computing.**

I. Introduction

DURING the last few years, the point of computation in cloud computing systems has begun spanning towards the terminal nodes of network infrastructure due to the availability of more powerful and smarter devices. This expanse in computational power triggered a diverse terminology including fog computing [1], nano data centers [2], and cloudlets [3]. Even though these concepts have their own differences and merits, they can be roughly clustered around the approach of disseminating tasks among a broader span of distributed nodes in cloud infrastructure instead of a small group of interconnected servers. We will refer such approaches as edge computing in the rest of this paper.

Bringing computation power to the edge of network reduces latency and enables code offloading to cloud. However, many services need to access data that is stored centrally. Thus, data access latency can be a bottleneck and override the benefits of edge computing especially for data-intensive services. Continuous increase in the volume of data absorbed, circulated and processed in cloud systems requires smart data distribution approaches. In this paper, we focus on the dissemination of data in a distributed cloud computing system with a large number of nodes that are accessible for computational purposes. We propose a decentralized approach to decide on replication and placement of data originating from a central server towards the end devices in cloud network. We consider a cloud specific trade-off between cost and latency which is the main criteria in shaping an aggression parameter to decide the extent that data is pushed towards the edge entities.

A. Aral was with the Department of Computer Engineering, Istanbul Technical University, Istanbul, Turkey and is now with the Institute of Software Technology and Interactive Systems, Vienna University of Technology, Vienna, Austria. e-mail: atakan.aral@tuwien.ac.at

T. Ovatman is with the Department of Computer Engineering, Istanbul Technical University, Istanbul, Turkey. e-mail: ovatman@itu.edu.tr

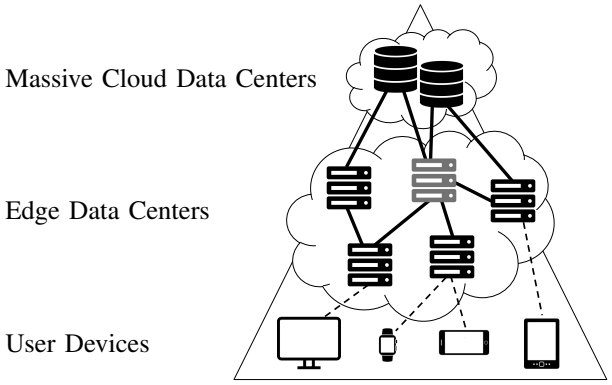

Fig. 1. Edge computing architecture.

Client side caching, which can be seen as a special case of the approach above with a strict replication policy, is traditionally used in distributed systems to reduce data access latency. Cloud caches are close to clients and provide requested data locally in the case of cache hit or retrieve requested data remotely from central storage in the case of cache miss [4]. However, caching methods typically result in low utilization of data copies since a cache can serve only the clients where it is stored. Replication, on the other hand, has the potential to provide a more cost-effective solution when replicas are placed on critical network nodes and serve requests from multiple nearby locations. In that sense, replication can exploit geographical locality of requests in addition to temporal locality.

Figure 1 illustrates the edge computing architecture [5] and assumed service model. User devices are connected to the closest edge nodes which are also interconnected with a certain network topology. Assume that all shown user devices are frequently requesting a data object. In the case of caching, all three edge nodes utilized by these users must store the same data to avoid fetching from central storage (massive cloud data center) at each request with high latency. However, with smart replication, a single copy of the data can be placed on the node highlighted in grey, which is well connected to these three nodes, thus reducing cost and maintaining similar latency to local access. Here, data object is an abstract term for a single or multiple files of any type. Our approach does not make any assumptions about the size of data objects.

Data replication can be employed with several different objectives, e.g. increase availability, security, fault tolerance or reduce response time and bandwidth consumption. It is also effective in distributing central storage load and increasing scalability [6]. In this study, we focus on performance benefits of replication with specific consideration of replica–client proximity to reduce latency and bandwidth consumption. Our aim is to answer following questions to minimize average replica–client distance in a bandwidth- and cost-effective way.

- Which data objects to replicate?
- When to create or destroy a replica?
- How many replicas for each object to create?
- Where to store each replica?
- How to redirect requests to the closest replica?

Although replicating all objects to all storage nodes results in optimum proximity and latency, it is quite wasteful since demand for each object varies and can be regional [7]. Moreover, in cloud paradigm, a service provider does not typically own the storage infrastructure but leases it from an IaaS provider on a pay-per-use basis. Thus, optimization of replica count and locations by considering the popularity of data objects is crucial [8]. Smart replica placement techniques can be especially effective in a multi-cloud scenario due to the availability of large number of geo-distributed storage options and possibility to exploit pricing discrepancies across regions and providers [7].

Rest of this paper is organized as follows. In subsections I-A and I-B, we introduce an analysis of data reference locality which motivates our approach and present our contribution to replica placement in edge computing, respectively. Later, we provide a detailed review and classification of the literature on replica placement methods in section II. Facility Location Problem (FLP) and its adaptation to replica placement is explained in section III. We propose Distributed Replica Placement (D-ReP) algorithm and present its experimental evaluation in sections IV and V, respectively. Finally, we conclude the paper in section VI.

A. Locality of Reference

Data accesses by geographically distributed users exhibit various patterns. These reference patterns can be summarized in three categories; temporal locality, spatial locality, and geographical locality [9]. Temporal and spatial locality are well-studied and addressed problems, however geographical locality, which we specifically focus in our study, gained importance due to the increase in magnitude of data being stored and processed in large-scale distributed systems. We use the locality classification described below through the rest of this paper.

Temporal Locality A data object that is accessed by a user is likely to be accessed again by the same user. Caching systems exploit temporal locality to answer requests locally after the first request [10].

Spatial Locality A data object that is near or relevant to a previously accessed object by a user is likely to be accessed by that user. When continuous blocks of data are stored instead of individual objects, caching can also make use of spatial locality with the assumption that sequentially stored data objects are relevant to each other. There exists approaches (e.g. [11]) where relevant data objects are predicted using past reference record and prefetched for local access. In this study, data objects are assumed to be independent of each other and thus spatial locality is not considered.

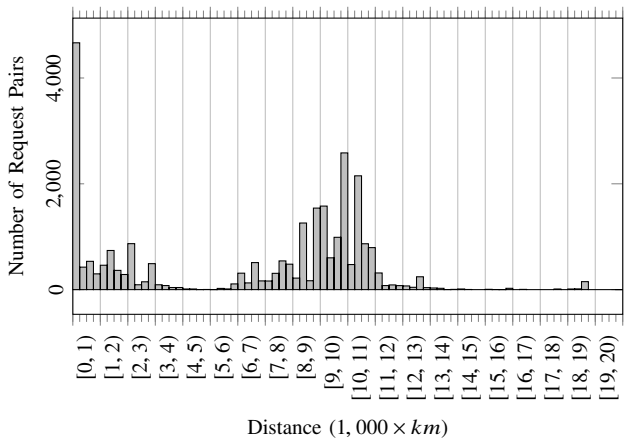

Fig. 2. Histogram of distances between data request pairs.

Geographical Locality A typical phenomenon in geographically distributed systems is that data objects that are accessed by a user are likely to be accessed by other nearby users. Although geographical locality is present in most distributed systems, for extreme cases one may consider a traffic congestion where drivers in a certain area are demanding map/traffic data more intensely compared to a sparsely populated area. This kind of intensifying data demands can be very dynamic and hard to predict in practice. Another example might be a social event such as a sports game or concert where users continuously request similar data such as video stream of a specific moment in the event. Geographical locality can also be almost permanent as in the scenario where residents of a town accessing to online public services or visitors accessing tourist guides.

In order to demonstrate the extent of geographical and temporal locality in global Internet requests, we examined CAIDA Anonymized Internet Traces 2015 Dataset [12] for a period of one-hour (between 13:00-14:00 on February 19, 2015). Analyses with different dates, times, and durations resulted in nearly identical distributions. The dataset contains general Internet traffic collected via monitors on high-speed backbone links which, we believe, is a good estimation for prospective traffic characteristics of edge computing. Cloud computing, being based on ultra-scale centralized data centers, has a traffic pattern that does not fit well to the decentralized edge computing case. Further information about the CAIDA dataset can be found in section V-A.

A histogram of origin distances of requests is shown in figure 2. It summarizes the distribution of distances between 26890 request pairs for a single randomly chosen data object. According to our analysis, more than 20% of the requests

are originating from 1000 km distance to each other. When we increase the diameter to 2000 km, it covers nearly 30% of all requests. Similar results to ours are obtained for the interactions between Facebook users in [13]. Caching cannot benefit from geographical locality because users are unable to access and are unaware of the caches of other nearby users. Hence, a smart strategy for data replication and replica discovery is needed to reduce storage cost while maintaining similar data access time.

B. Contribution

Our main contribution in this study is two-fold. First, we propose a completely decentralized, dynamic, and online algorithm for placement of data replicas across IaaS providers in an edge computing scenario. Second, we propose a low-overhead messaging methodology to notify edge entities about nearby replicas so that they can submit their future data requests to them, instead of remote central storage.

Replica Placement Algorithm We present D-ReP algorithm where storage nodes that host replicas analyze observed demand on replicas and act as local optimizers. They evaluate cost of storing replicas as well as expected latency improvement to make a migration or duplication decision to one of neighbours. They may also decide to remove the local replica. Such decisions are made to maximize an objective function based on FLP. The algorithm also allows user to control the balance between cost- and latency-optimization using an input parameter. Experimental results on both real and synthetic workload traces demonstrate significant improvements in replica access latency as well as network overhead and storage cost.

Messaging Methodology In order to gain promised benefits of D-ReP algorithm, edge entities should be aware of the closest replica when they request a data object. However, complete awareness is only possible with centralized control or by broadcasting replica locations periodically. We instead propose a replica discovery approach where the most relevant nodes are identified and only they are notified of replica creations or removals.

II. Related Work

Table I summarizes the literature on replica placement. Among these, we provide comments only on the most relevant approaches to ours for brevity. We categorize studies on the basis of three binary classification rules (i.e. Centralized/Decentralized, Complete/Partial Information, and Static/Dynamic). In addition, we provide information on their intended environment, network topology restriction and optimization objectives. More details on columns of the table are given below.

Decentralized (DC): A check mark (✓) indicates that replica placement algorithm is executed on multiple locations in parallel. Others manage placements from a single node.

Partial Information (PI): Centralized methods always use complete demand and topological information for placement. However, some of the decentralized algorithms require only local and partial information to run.

TABLE I
Summary of the Literature on Replica Placement

	DC	PI	DY	Environment	Topology	Objectives
[14]				Web	Tree	PX
[15]				CDN	Any	PX
[16]				N/A	Any	PX
[17]				Data Grid	Tree	PX, CT, LB
[18]				N/A	Tree	PX
[19]				N/A	Any	PX
[20]				Cloud	Any	PX
[21]				Cloud	Any	BW, LB
[22]				Cloud	Any	PX, CT
[23]			✓	N/A	Any	AV
[24]			✓	Data Grid	Multi-Tier	PX, CT, BW
[25]			✓	CDN	Any	PX, CT
[26]			✓	Cloud	Any	PX, BW, LB
[27]			✓	Data Grid	Multi-Tier	PX
[28]			✓	Data Grid	Any	PX, BW, AV
[8]			✓	Cloud-CDN	Any	PX, CT
[29]			✓	Data Grid	Tree	PX, BW, AV
[30]			✓	Data Grid	Multi-Tier	PX, BW
[31]			✓	Cloud	Tree	PX, AV
[32]			✓	Cloud	Any	BW, LB
[33]			✓	Cloud-CDN	Any	CT, AV
[34]			✓	Cloud	Any	PX, AV
[35]	✓			Cloud-CDN	Any	PX
[36]	✓		✓	Cloud	Complete	PX, CT, BW, AV
[37]	✓		✓	Data Grid	Any	PX, CT, BW, AV
[38]	✓		✓	P2P	Any	CT, BW, AV
[13]	✓		✓	Web	Any	PX, BW
[39]	✓	✓		N/A	Multi-Tier	PX, BW
[40]	✓	✓	✓	Web	Any	PX, CT, LB, BW
[41]	✓	✓	✓	P2P	Any	PX, CT
[42]	✓	✓	✓	Web	Any	PX, CT
[43]	✓	✓	✓	Web	Any	PX, CT
D-ReP	✓	✓	✓	Cloud	Any	PX, CT, BW

Dynamic (DY): Replication is dynamic if the number and location of replicas change over time based on observed demand and/or cost. Network status is another source of dynamicity. In static replication, replicas are not duplicated, migrated or deleted after their initial creation.

Environment: Execution environment of an algorithm can be one of the following: Cloud, Content Delivery Network (CDN), Cloud-Based Content Delivery Network (Cloud-CDN), Peer-to-Peer System (P2P), Data Grid, Web (e.g. Internet Services), or unrestricted/unspecified (N/A).

Topology: If there is a restriction on the network topology graph among the nodes for an approach to work, we indicate it in this column. Options are tree, complete graph, multi-tier, and unrestricted (i.e. any graph).

Objectives: This column lists the criteria which are aimed to be optimized via replication. The criteria that are encountered in the literature review are listed below.

- **Proximity (PX)** indicates user access time to replica. Optimizations of network latency, response time, hop count, and distance are all grouped under this criterion.
- **Cost (CT)** indicates monetary cost of storing replicas. Methods that do not explicitly consider monetary cost but aims to minimize replica count are also included.
- **Bandwidth (BW)** indicates network overhead and bandwidth utilization.
- **Availability (AV)** indicates fault tolerance or reliability.
- **Load Balance (LB)** indicates avoidance of hotspots by spreading demand across several nodes.

Another classifications of replica placement algorithms can be found in [44] and [45]. The interested reader can also refer to surveys [6] and [46] for more details on dynamic replica placement in data grids.

Centralized methods lack scalability and create a performance bottleneck in replica placement. They are particularly infeasible for large-scale distributed systems such as the Internet and Cloud. However, most of the literature in replica placement is centralized approaches due to the extra complexity decentralization brings. First, efficient synchronization of inputs (e.g. demands, latencies, costs) and outputs (e.g. number and location of replicas) of the replication algorithm is challenging especially in a dynamic environment. Second, local knowledge about the environment deteriorates quality of placement in comparison to global knowledge. Thus, some decentralized approaches assume global knowledge at each location and suffer from scalability issues as centralized ones.

A. Centralized Methods

Geographic placement of shared data for cloud services is investigated in [26]. Suggested technique places data to the weighted geographical center of their users and maps them to the closest data center by also considering load balance. In [28], first number of replicas is calculated based on the popularity, recentness and customer-assigned importance of data. Then the computed number of replicas are placed on data centers by minimizing a distance metric.

In [20], authors suggest a methodology to place data replicas to achieve dual objectives, i.e. to increase security by placing complementary pieces to nonadjacent locations and to reduce data access time by placing them in central locations. They make use of betweenness, closeness and eccentricity centrality measures as well as graph coloring to achieve these objectives. Suggested methodology always create a single replica of each data piece without considering its popularity. Replica is placed to the node with the highest number of accesses to that data.

In another study [8], a request redirection strategy in addition to replica selection and placement is proposed. It optimizes data storage and transfer cost for distributing content to users over storage clouds. A mixed integer programming and multiple heuristic solutions are provided. Experimental comparison demonstrate superiority of online and dynamic algorithms in terms of cost and number of QoS violations.

A two-phase mapping of tasks to replicas and replicas to data centers is suggested in [22]. Proposed genetic algorithm solution assumes that number of replicas is foreknown and aims to reduce cost and latency by decreasing number and size of data movements between data centers.

There also exists studies focusing on a specific type of graph topologies on which replicas are distributed. Tree networks are considered in [14] with a specific emphasis on read and write costs. Another tree topology based solution [31] aims to minimize number of QoS violations in terms of latency. Suggested method exploits already implemented replication practice (which is intended for availability) by placing data of applications with high QoS requirements on high-performance nodes. More recently, combination of replication and erasure coding mechanisms is exploited to leverage availability in multi-clouds [47], [48].

Typically, centralized methods yield optimal or near-optimal placement of replicas by making use of centrality metrics. However, these have following drawbacks in comparison to distributed and decentralized methods [42].

- Collecting and transferring complete system state (e.g. demand for each file, storage cost, network information, etc.) causes a network overhead especially in dynamic and large-scale systems.
- Similarly, distributing control data (e.g. replicated files and their locations) uses up bandwidth and causes delay. This increases response time of the algorithm.
- Optimization is computationally expensive and does not scale well with the number of nodes.
- Algorithms are usually not iterative. Thus, complete re-optimization must be carried out even for minor changes.
- In the case of median-based algorithms, number of replicas should be given a priori.
- Central replica controller is a single point of failure.

B. Decentralized Methods

In one of the earliest attempts to dynamic replica placement [40], authors propose a dissemination tree to replicate and synchronize data. The aim is to place minimum number of replicas on access paths while respecting latency guarantees and balancing load. Caching is also used to that end, in addition to replication.

A file replication algorithm which places replicas to so called traffic hubs on client-server paths of a P2P system is proposed [41]. The goal is that the replication will be more cost-efficient than creating replicas on all nodes of the path and yield higher utilized replicas than client side caching. Traffic hubs are chosen as the nodes where multiple client-server paths for the same file coincides. Suggested algorithm is decentralized and self-adaptive in the sense that each node can decide to store replicas by analyzing the traffic running through it and determining the most popular files. Although such analysis can be possible in certain P2P scenarios, it would cause privacy issues in a cloud environment.

Authors of article [39] suggest a distributed replica placement algorithm for systems that consist of replication groups (e.g. departments in a university). When a data is requested, it can be provided from other servers in the group resulting in decreased access time than fetching from the origin server. However, such an algorithm is not applicable to cloud or edge computing models where inherent replication groups do not exist and the topology is immense and highly dynamic.

In [37], data grid topology is partitioned into network regions where replication and placement decisions are optimized. Main focus here is to avoid network congestion. Similarly, authors of the study [35] suggest partitioning cloud topology graph and greedily deciding the number and location of replicas at each cluster in an attempt to minimize the number hops between content provider and its users. Although replica placement is conducted in a distributed manner in these approaches, complete topology information is still required for the centralized partitioning phase.

One of the most similar works to ours is [36]. Authors propose an algorithm based on a game-theoretical model which is executed at each node autonomously and may replicate or migrate its data or remove it depending on its availability, communication cost and monetary cost. Their approach mainly differ from ours in the sense that each node is aware of locations of other replicas as well as rent price, demand, and bandwidth of all other nodes. Moreover, their knowledge must be periodically updated which would cause a performance bottleneck in edge computing scenario with hundreds, if not thousands of nodes.

Primal-Dual based distributed approximation algorithms for FLP are presented in graph theory literature and their variations are applied to the problem of Internet service placement [42], [43]. In the former, a small scale FLP is solved iteratively for each group of nodes that are in close proximity (called r-ball). The latter study introduces a conditional betweenness centrality metric and removes the restriction that facility location must be within the r-ball. To that end, most central nodes in the topology is calculated and FLP is solved on this selected subset of nodes at each iteration. Contrary to these two approaches, we aim to eliminate any message passing or broadcasting between non-adjacent nodes. Reporting demand estimates, centrality values or facility location decision over multiple hops is acceptable for a single instance of FLP such as replicating an Internet Service. However, in our case, number of data objects and thus number of FLP instances can easily reach thousands or even millions which would cause an infeasible network overhead. Hence, any replica placement algorithm which requires frequent communication between the nodes is impractical for granular data objects.

III. Facility Location Problem

Facility Location Problem (FLP) concerns with placement of facilities in order to serve demands of geographically distributed customers with minimum cost [49]. Cost of opening a facility at a certain location depends on the distance from customers who will be served by that facility and their demand. It may also include any fixed costs. Different objective functions and constraints lead to several variations of the problem that share these main ideas.

In this section, we first present an overview of available problem models in the literature with their strengths and weaknesses, and then elaborate on how problem is adapted to the case of placing data replicas on distributed cloud infrastructure. Table II list the notation that is used throughout the paper along with their definitions.

A. Background on Problem Models

For brevity, here we discuss only discrete FLP models where facilities can be opened at finite number of locations. In such models, customer and possible facility locations with their distances can be represented as a simple, weighted and undirected graph. Most basic form of discrete FLP looks for the optimum location of a single facility which minimizes sum of distances from all customer locations to the facility. In [50], three classes of problems are defined: median, covering and center problems.

TABLE II
Notation for D-RePP Algorithm and FLP

Notation	Explanation
h_i	Level of demand from given customer in units
d_{ij}	Distance or transportation cost between given customer and facility per unit demand
f_i	Cost of opening a facility at given location
X_{ij}	Binary variable which indicates whether given facility is the closest open facility to given customer
Y_i	Binary variable which indicates whether a facility is open at a given location
$unit\_price_i$	Price (in USD) of storing unit data (e.g. Byte, MB, etc.) per minute in given edge node
$replica\_size$	Size of the data object to be replicated in same unit as $unit\_price_i$
$epoch$	Predefined epoch duration in minutes
$num\_requests_i$	Total number of requests for given replica during the previous epoch
$latency_{ij}$	End-to-end latency between given user and replica locations in milliseconds
λ	Level of replica expansion

In median problems, number of facilities to be located (k) is foreknown. In such problems, cost function for each customer is defined as the demand of that customer (h_i) multiplied by its distance to the closest facility location (d_{ij}). Optimum solution is the locations of k factories which minimize sum of all costs as given in equation 1. Here X_{ij} is a binary variable which is set if facility at j is the closest facility to customer i.

$$Total\ Cost = \sum_i \sum_j h_i \cdot d_{ij} \cdot X_{ij} \tag{1}$$

Fixed cost of opening a facility is introduced in covering problems where the objective is to minimize total fixed cost by maintaining a predefined maximum acceptable service distance. In such problems, optimal number of facilities does not need to be provided beforehand. The third class of FLP is named center problems in [50]. Given the number of facilities to be located, goal is to minimize the maximum distance between a customer and the nearest facility.

All these classes of problems require some a priori knowledge about either number of facilities or maximum acceptable distance. A more general definition of FLP without this limitation is presented in [51]. Equation 2 is their general cost using the same notation as equation 1 with another binary variable Y_j which indicates if a facility is open at j, and f_j which is the fixed cost of opening a facility at j. In addition, here d_{ij} should be considered as unit service cost instead of distance.

$$Total\ Cost = \sum_j f_j \cdot Y_j + \sum_i \sum_j h_i \cdot d_{ij} \cdot X_{ij} \tag{2}$$

Further variations of FLP include models with limited facility capacities, limited knowledge of parameters (e.g. demands and costs), multiple types of demand, multiple types of facilities, and uncertainty of future parameters [49], [50].

B. Adapting the Problem

In replica placement problem, optimum number of replicas cannot be known a priori. Thus, cost function in equation 2 better fits our case. In addition, we consider uncapacitated version of the problem since allocation of cloud resources is out of this study's scope. Readers may refer to our previous work [52] for decentralized resource allocation in cloud systems.

FLP models with multiple products (i.e. multi-commodity FLP) aim to determine facility locations which optimize distance to demands for all products. This is critical for industrial cases where building a facility has a fixed cost and it is infeasible to build a different facility for each product [49]. However, in Storage as a Service (STaaS) scenario, upfront costs and commitments are eliminated and customer only pays for the storage used. Thus, it is not necessary to cluster multiple replicas in a provider and location of each replica can be optimized individually. Hence, we solve a single-product FLP instance for each replica. Facility opening cost (f_j) translates to storage cost that would be charged by IaaS provider until the next iteration of algorithm (i.e. next epoch).

$$f_j = unit\_price_j \cdot replica\_size \cdot epoch \tag{3}$$

Customer demand (h_i) is proportional to the number of requests received for a replica during the previous epoch.

$$h_i = num\_requests_i \cdot replica\_size \tag{4}$$

Finally, distance metric d_{ij} is chosen as the latency between a virtual machine and a replica location. Since f_j is a monetary unit, d_{ij} should also be converted to the same unit so that two sides of the addition are commensurable. That's why we suggest a unit conversion factor, λ, which represents expendable unit cost in exchange for a unit decrease in latency per unit demand. Value of λ parameter also determines the level of replica expansion as we explain in section IV.

$$d_{ij} = latency_{ij} \cdot \lambda \tag{5}$$

Final definition of the cost function for adapted FLP is presented in equation 6. Note that, $replica\_size$ is removed from both sides of the addition since it is constant for each problem instance and has no effect on minimization objective.

$$\begin{aligned} Total\ Cost \ &= \ \sum_j unit\_price_j \cdot epoch \cdot Y_j \\ &+ \ \sum_i \sum_j num\_requests_i \cdot latency_{ij} \cdot \lambda \cdot X_{ij} \end{aligned} \tag{6}$$

We assume certain knowledge of current demand for each data object and storage prices of IaaS providers. There exists strategic FLP variations which model uncertainty in future demands and costs. They place facilities as long-term investments which will be less vulnerable to such changes because building a facility is a cost- and time-critical decision. Again, in the case of cloud computing, such constraints are no longer valid. By favour of pay-as-you-go storage, replicas can be relocated momentarily for little or no cost. We avoid any interruption in service by keeping the replica at its source node until it is migrated to its destination.

IV. Decentralized Replica Management

A. Requirements and Assumptions

Main requirement for D-ReP algorithm is to place replicas across storage nodes (cloud based storage providers and edge entities) in a way that the cost function given in equation 6 is minimized. There are also a number of nonfunctional requirements involved in our scenario.

First, the algorithm must be distributed and completely decentralized. As explained in section II, centralized algorithms are not suitable for the edge computing scenario due to the large number of nodes. Collecting global topological and demand information in a centralized node and executing complete optimization algorithm does not scale well with node count [43], [53]. Communication between nodes regarding replica placement should also be kept to a minimum to avoid additional overhead. Second, multi-cloud and edge computing environments are highly dynamic. Edge users continuously enter and leave the network through connections with various latencies. Demand from each node and for each data object as well as storage prices can also vary greatly over time. Thus, number and location of the replicas should be dynamic. Finally, the algorithm should be online since it is not possible to obtain a priori knowledge of future requests and environment.

Inputs of D-ReP algorithm remain limited to the following items by taking the abovementioned constraints into consideration. Each algorithm instance executing in a node is only aware of that node and its immediate neighbours in the undirected network topology graph. We assume that below listed information can be collected using standard monitoring tools (e.g. Skitter), DNS queries or routing tables.

- Number of requests for each replica that is stored in that node
- The neighbour node that each request is received through
- Perceived latency to each neighbour node
- Unit storage price of each neighbour node

Another assumption in our study is regarding the consistency of replicas. Since cloud and edge computing scenario do not make significant difference in data consistency with respect to traditional distributed systems, we leave it out of scope. We assume either read-only data (as typically in content distribution networks [9], [27], [28]) or an independent consistency service. In the latter case, any primary copy based distributed protocol can be implemented, e.g. Viewstamped Replication, Paxos or Zab.

B. D-ReP Algorithm

Two versions of the algorithm (i.e. source and edge) are triggered in equally spaced epochs and iteratively push replicas from the source node (central storage) towards requesting edge nodes. A group of nodes in close proximity that repeatedly demand same data objects cause creation and migration of replicas towards them. Replicas are discarded or migrated to other locations when demand fades.

Source version of the algorithm runs in the same node as central storage and can only create replicas of data objects in neighbours of that node. Edge version, on the other hand,

runs at each node where at least one replica is present (i.e. an active node). A tiny virtual machine is provisioned from cloud provider whenever a storage space is leased to store the first replica there. This virtual machine can turn itself off when no replicas are left or terminate after a period of inactivity. This version may decide to duplicate or migrate replica to neighbours, remove it, or do nothing. All operations in both versions are decided by comparing expected benefit with cost. There is no communication or interaction between the two versions or among edge instances of the algorithm.

1) Source Version: A replica of data object k is created in neighbour n of central storage node c if expected latency improvement is worth the storage cost of replica. In other words, cost value in equation 6 should decrease after the replication. Equation 7 is the condition to create a new replica.

$$num\_requests_{knc} \cdot latency_{nc} \cdot \lambda > unit\_price_n \cdot epoch \quad (7)$$

Here $num\_requests_{knc}$ is the total number of requests for k received through n by c in the most recent epoch(s). If the condition does not hold for any of the neighbours, no replica is created for that data object in current iteration.

2) Edge Version: A replica can duplicate itself to a neighbour n of its current host node h if expected additional latency benefit of new replica is greater than its storage cost. Equation 8 is the same as new replica creation condition in the source version (equation 7) except c is replaced with h.

$$num\_requests_{knh} \cdot latency_{nh} \cdot \lambda > unit\_price_n \cdot epoch \quad (8)$$

If duplication condition does not hold for a neighbour, migration condition in equation 9 is evaluated instead. Here, N is the set of all neighbours of h. While latency of the request received through n will decrease by $latency_{nh}$, we assume that latency of the requests from all other neighbours will increase by the same amount since they will be answered through h. Actually, these neighbours may have lower latency connections to n and they may bypass h. But the algorithm would not be aware of such connections since it has only local topology information for the sake of decentralization.

$$\left(num\_reqs_{knh} - \sum_{\substack{i \in N \\ i \neq n}} num\_reqs_{kih}\right) \cdot latency_{nh} \cdot \lambda \quad (9)$$
$$> \left(unit\_price_n - unit\_price_h\right) \cdot epoch$$

A replica may get underutilized due to the creation of new replicas or changes in demand. In such cases, it should be discarded to avoid unnecessary storage cost. Replica or cache replacement is a well studied area of research in distributed systems and more specifically in data grids. Techniques include not only traditional time-out, LRU, LFU, or FIFO but also more sophisticated weight-based [54] and prediction-based [55] approaches. In order to stay focused on replica placement, we resort to a threshold-based weight calculation such that a replica is removed when its utilization in the last epoch drop below a dynamic threshold. The threshold is calculated as a predefined proportion (i.e. $fade\_rate \in [0, 1]$) of the expected utilization at replica creation (i.e. $original\_num\_reqs_h$) as shown in equation 10.

$$\sum_{i \in N} num\_reqs_{kih} < original\_num\_reqs_h * fade\_rate \quad (10)$$

Input: Set of currently stored replicas K, Set of neighbour nodes N, Current node h, $num\_requests$, $latency$, $unit\_price$, $epoch$, KRL
Output: Set of operations to be executed O

```
1:  for all {k ∈ K} do
2:    for all {n ∈ N | (n, k) ∉ KRL_h} do
3:      if num_requests_knh · latency_nh · λ
                > unit_price_n · epoch then
4:        O ← O ∪ {duplicate k to n}
5:        break
6:      end if
7:      if (num_reqs_knh − Σ_{i∈N, i≠n} num_reqs_kih) · latency_nh · λ
                > (unit_price_n − unit_price_h) · epoch then
8:        O ← O ∪ {migrate k to n}
9:        break
10:     end if
11:   else
12:     if Σ_{i∈N} num_requests_kih
                < original_num_requests_h * fade_rate
        then
13:       O ← O ∪ {remove k}
14:     end if
15:   end for
16: end for
```

Fig. 3. Pseudo code for one iteration of the D-ReP Algorithm.

Pseudo-code of D-ReP algorithm is presented in figure 3. At each epoch, the algorithm iterates each replica that is stored locally where Equations 8 and 9 are tested for each neighbour that do not already have the same replica. In lines 5 and 9, `break` statements guarantee that at most one duplication or migration decision can be made for a replica in an epoch. If the second `for` loop (lines 3–10) terminates normally (and not due to `break` statements), then `else` part (lines 12–14) is executed where equation 10 is tested. This `for...else` structure guarantees that a replica can only be removed if it is chosen for neither duplication nor migration to a neighbour. It should also be noted duplication and migration operations suggested by the algorithm can only be executed if their destination has enough storage space. Time complexity of the algorithm is $O(|K||N|)$ where $|K|$ is the number of stored replicas on that node and $|N|$ is the number of neighbours.

User-provided parameter λ can be tweaked by a service provider to control cost and latency levels. Greater λ values result in more aggressive expansion of replicas across the network, thus lower latency values in exchange for higher storage cost. D-ReP can also be extended for more precise latency guarantees. In this sense, end-to-end latency of all requests need to be monitored and used in decision making.

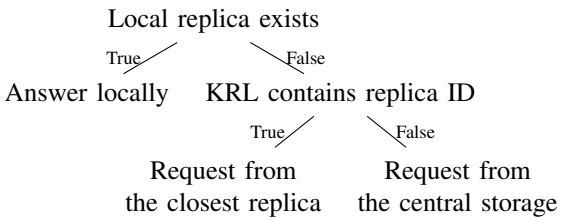

Fig. 4. Decision tree to answer a data request at an edge node.

C. Replica Discovery

In replica-blind services, requesting nodes are unaware of replica locations and they always submit their request to central storage. If there exist a node with a replica of the requested data on the path, it answers the request [25]. Replica-blind services are typically implemented in domain-specific, single-tenant distributed systems such as CDNs. However, servers cannot analyze the requests flowing through them in a multi-tenant system such as cloud, hence replica-awareness is mandatory. Replica-aware services also make it possible to answer requests by nearby replica locations that are not on the path to central storage.

We propose a messaging system for replica discovery that complements D-ReP algorithm. In accordance with decentralized design of the algorithm, messaging system is free of broadcast messages and central control. Our objective is to notify a node about a replica only if that node is expected to request that replica in the near future. This expectancy is inferred from both temporal and geographical locality of requests. Each active node keeps a Known Replica Locations (KRL) table. The table stores replica id, replica node, and latency to replica node. It may contain multiple locations for each replica and is updated in the following occasions.

- When a replica is migrated or duplicated from n_1 to n_2, new host n_2 notifies all nodes that requested that replica from n_1 in the most recent epoch(s). List of such nodes is transferred from n_1 to n_2 together with the creation command and replica.
- n_2 also notifies its neighbours at both creation and removal of a replica.

Latency to the replica node is approximated with latency observed for notification message from n_2 and it is updated when a request is answered by n_2. Figure 4 demonstrates the decision tree that is evaluated when a user request a data object at its local edge node. If a replica of the data is present at that node, then request is answered with minimal latency. If the replica is not stored locally but the node is aware of one or more replicas for that data, a request is submitted to the one with the least latency. Otherwise, it is requested from central storage with high latency. To avoid a performance bottleneck in searching KRL table, associative arrays such as hash tables can be implemented.

D. A Use Case Scenario

Let us continue with the traffic data dissemination example mentioned in section I-A and detailed in figure 5 to illustrate how replica placement and discovery are carried out. In this example, users of a cloud based navigation service are accumulated in an area due to traffic congestion. They utilize a nearby edge data center (E–1) for their computation needs regarding navigation and routing. Their data requirements (e.g. area map, traffic density, etc.) are similar considering that they are on the same road.

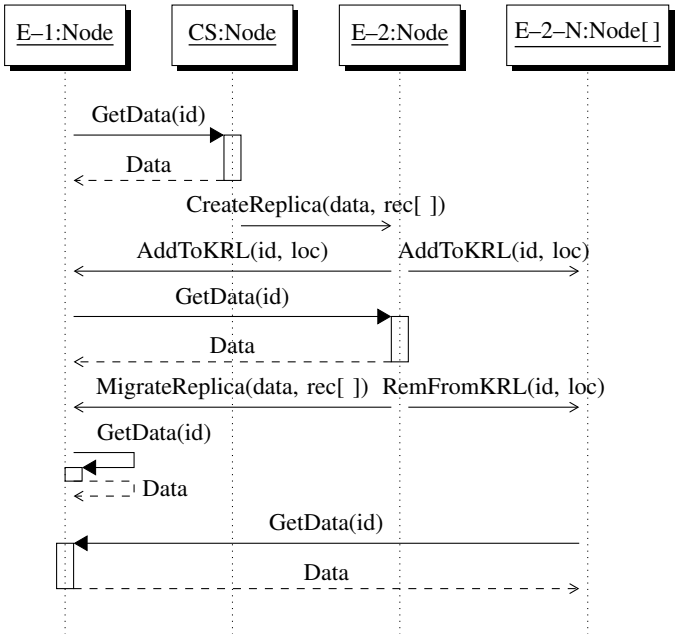

Fig. 5. Sequence diagram to depict a use case scenario of replica placement and discovery.

Assume that E–1 does not initially store a replica of the required navigation data and its KRL table does not contain any entries for it. Thus, data will be requested from central storage (CS) querying with its id. Also assume that, source version of D-ReP algorithm that runs in CS decides to create a replica of this navigation data in one of its neighbours, edge data center 2 (E–2), due to high demand from that direction. Replica creation message includes the location of recent accesses to that data (`rec[]`) as well as data itself. Since E–1 recently requested that data, it will be notified by E–2 along with E–2's neighbours (E–2–N) when the replica is successfully created. Notification message instructs to add a new location (`loc`) for the data (`id`) to their KRL tables. If the latency between E–1 and E–2 is lower than the latency between E–1 and CS, subsequent requests for the same navigation data will be directed to E–2, instead (see figure 4).

Let us further assume that edge version of the algorithm, which now runs in E–2, decides in the future epochs to migrate the replica to E–1 for further latency improvement. E–2's neighbours will be notified by this action so that they remove E–2 from their KRL tables for that data and their subsequent requests may be directed to E–1 depending on the entries in their KRL tables (for brevity, we omit how E–1 may be added to their KRL tables). Moreover, data requirements for similar navigation tasks in E–1 will be answered locally.

TABLE III
RANGES AND DEFAULT VALUES OF SIMULATION VARIABLES

Simulation Variable	Range	Default Value(s)
Epoch Length (min)	[1, 20]	3, 10
λ	[0.01, 0.20]	0.10, 0.16
Cache Capacity	[10, 200]	30

V. EVALUATION

We extend widely used simulation framework, CloudSim [56], to evaluate performance of D-ReP as well as to examine its run time behaviour. A caching system and a central storage (no-replication) system are also modelled in CloudSim to compare with D-ReP. We present our findings and comments in this section.

A. Experimental Setup

In order to evaluate D-ReP algorithm and baseline methods realistically, we use CAIDA Anonymized Internet Traces 2015 Dataset [12]. Dataset contains anonymized passive traffic traces from CAIDA's 'equinix-chicago' high-speed monitor. Specifically, we use IPv4 packets data from February 19, 2015 between 13:00-14:00 (UTC) which contains more than 2.3 Billion records. Multitude of parameters, level of detail, and thus high computational complexity in our simulation constrain us from analyzing longer periods. We utilize GeoLite2 IP geolocation database[1] to map source IPv4 addresses to geographical locations. We use Amazon S3 prices[2] to calculate storage and data transfer costs.

Since CAIDA dataset does not include network topology information, an undirected network topology graph is generated with Barabási–Albert scale-free network generation model [57] using BRITE tool [58]. This model is widely used to represent human-made networks such as the Internet mainly due to two characteristics borrowed from real networks: incremental growth and preferential node connectivity. When new nodes are being added, a probability function for edge generation ensures that new nodes tend to link to the more connected nodes, i.e. hubs. Generated topology contains 1000 nodes placed on a 1000×1000 coordinate plane, 2994 edges, and a heavy-tailed distribution (Pareto with shape 1.2) of bandwidth in the range of 10 to 1024 Mbps. Lower and upper values are chosen to represent a wide range of connections from end-user network at the edge to the Gigabit Ethernet connections between massive data centers. Location of central storage is selected as the node with the greatest closeness centrality.

Different from D-ReP, replicas in the caching system are created directly at the nodes that request data objects and these replicas can only be utilized by their creators. The number of replicas that can be stored at each cache is limited. When cache capacity is full, least recently used (LRU) cache replacement strategy is used. Simulation variables are epoch length, λ and cache capacity. Ranges and default values of variables are given in table III. For each experiment, one of these varies while the others are assigned their default values. Range limits are chosen by ensuring the best performance of D-ReP and caching algorithms as well as considering practical constraints (e.g. unrealistically short and long epochs, or too small cache capacities are not used even if the algorithms perform well).

[1]http://www.maxmind.com/
[2]https://aws.amazon.com/s3/pricing/

TABLE IV
PROBABILITY DISTRIBUTIONS AND CHOSEN PARAMETERS

Probability Distribution	Parameter(s)	Value(s)
Uniform	Range	$[a, b] = [0, 1000]$
Exponential	Mean	$\mu = 200$
Normal	Standard deviation	$\sigma \in \{50, 100, 150\}$
	Mean	$\mu = 500$
Chi-squared	Degree of freedom	$k \in \{1, 2, 3\}$
Pareto	Scale	$x_m = 1$
	Shape	$\alpha \in \{3, 4, 5\}$

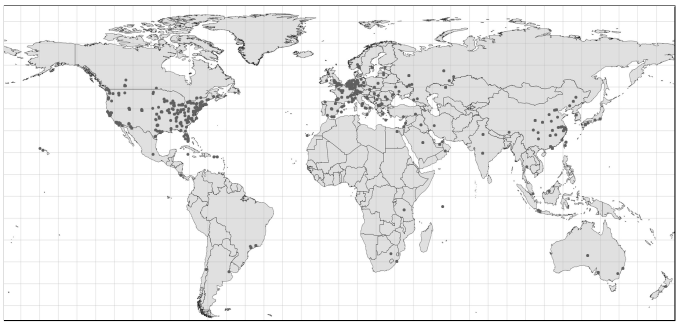

Fig. 6. Geographical distribution of edge nodes with the highest workload.

In addition to CAIDA Internet Traces, the algorithm is also evaluated with synthetic user demands with the purpose of generalizing results and conclusions. Same network topology, baseline methods, and simulation variables as described above are in effect here as well. Various probability distributions are utilized to generate locations for request so that different levels and forms of geographical locality are experimented. In table IV, uniform, exponential, normal, Chi-squared, and Pareto distributions are listed as five different distributions used in evaluations. Three different parameter values are used for the last three distributions to reach a total of eleven different cases. For each case, x and y coordinates of 100000 requests are randomly generated on a 1000×1000 plane using two instances (for x and y) of respective probability distribution.

Each data request is assigned to the edge node with the smallest Euclidean distance in the the coordinate plane. In the case of CAIDA traces, both geographical coordinates (i.e. latitude and longitude) of data requests are projected to the range $[0, 1000]$ for this purpose. On the other had, parameters of the random distributions in table III are chosen in such a way that vast majority of the generated values stay in the range $[0, 1000]$. Any outliers are reassigned to closest endpoint. Figure 6, shows 200 nodes with the highest number of data requests assigned from CAIDA traces. For demonstration purposes, locations of these nodes are reverse mapped from 1000×1000 plane to geographical coordinate system.

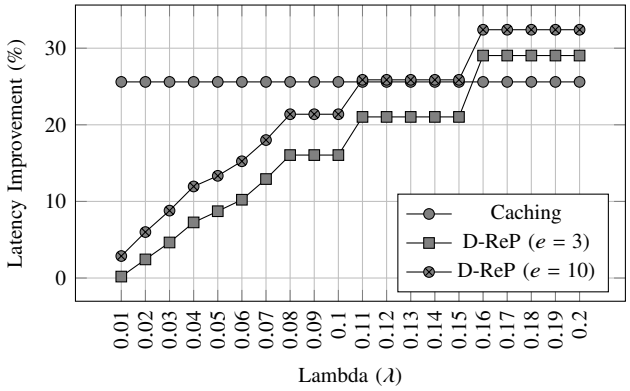

Fig. 7. Latency Improvement Rate with Variable λ.

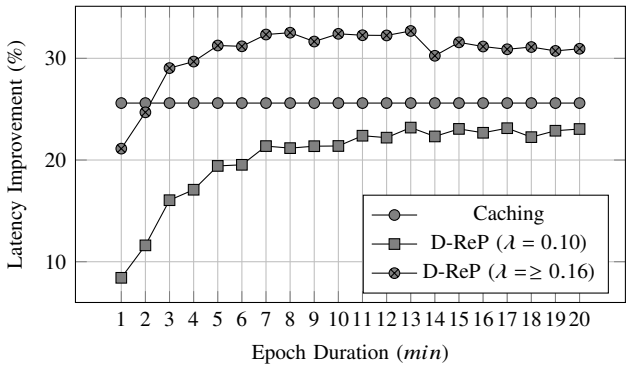

Fig. 8. Latency Improvement Rate with Variable Epoch Duration.

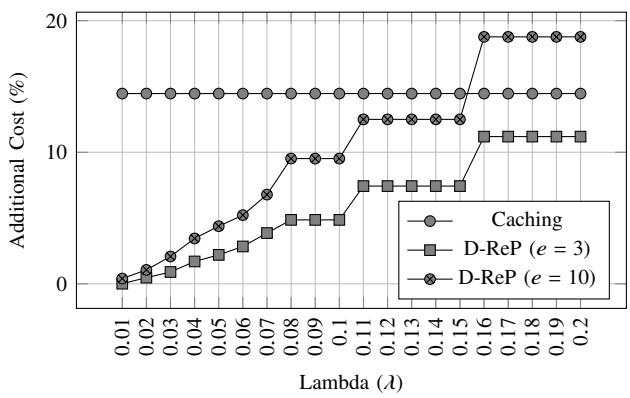

Fig. 9. Additional Cost Rate with Variable λ.

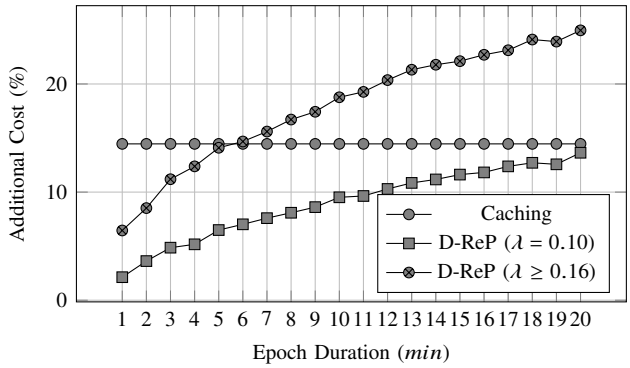

Fig. 10. Additional Cost Rate with Variable Epoch Duration.

B. Latency and Cost

We use latency improvement rate to measure the extent that D-ReP and caching solutions decrease average access latency to replicas with respect to no-replication solution. In figure 7, latency improvement of the algorithm with varying λ is presented. Since λ is only effective for D-ReP, latency improvement of caching is constant at 25.60%. As we increase λ, D-ReP is able to afford more replicas especially at outer locations. Hence, average latency to replicas decreases steadily to the levels comparable to caching and beyond. Increasing epoch duration (e), as shown in figure 8, also improves latency to some extent. However, increase is not as steady and converges after around 7 minute-long epoch. Longer epochs have the advantage of aggregating more data to make sense, but they lose their reactivity to small changes in demand.

We also measure the rate of replica storage cost that each method causes in addition to central storage cost. Figures 9 and 10 demonstrate that D-ReP incurs significantly lower cost than caching in all cases expect very high λ values ($\geq$ 0.16) and very long epochs ($\geq$ 15 *mins*). Here, additional cost incurred by caching is constant at 14.46%. To better interpret relative improvement rates in Figures 7 to 10, we present absolute cost and latency values of no-replication solution. Average data access latency is 1.82s, while data storage and transfer cost per request is \$0.24. Hence, a latency improvement rate of 30% corresponds to a 0.55s gain in time which is quite significant as a user can easily request hundreds of objects in one session.

C. Benefit-Cost Ratio

We use Benefit-Cost Ratio (BCR) indicator [59] to evaluate efficiency of algorithms to address the trade-off between data access latency and data storage/transfer cost. BCR summarizes overall value for money of a proposal and is useful to decide between options when the most profitable is not obvious, e.g. more expensive option is also more beneficial. Greater BCR values are favourable and proposals with a BCR less than 1 are generally rejected. It is calculated, in our case, as rate of latency improvement divided by rate of additional cost. Formula for BCR is provided in equation 11.

$$BCR = \frac{Latency\ Improvement\ (\%)}{Additional\ Cost\ (\%)} \tag{11}$$

As shown in figure 11, D-ReP is more cost-efficient in all λ values, with 1.93 times greater BCR than caching on average. Considering this result with Figures 7 and 9 reveals that λ can be used to control the desired level of latency in a cost-efficient way. First data point of 'D-ReP (e = 3)' is not displayed in this chart as BCR value is unrealistically high (18.87) because denominator (additional cost) approaches zero. Figure 12, on the other hand, shows that longer epoch durations can be less efficient than caching. Although, latency improvement converges in figure 8, cost continues to increase steeply in figure 10. Thus, epoch duration does not come up as an appropriate way of controlling latency.

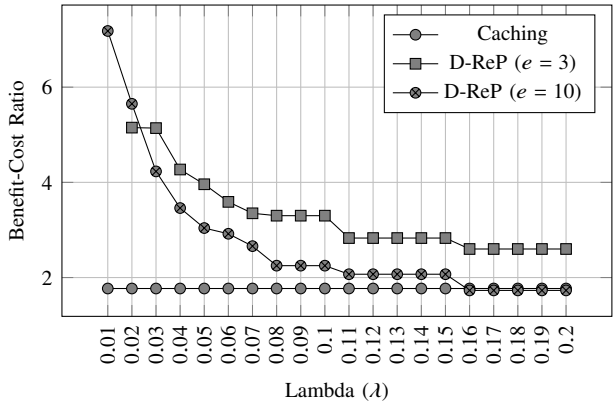

Fig. 11. Benefit-Cost Ratio with Variable λ.

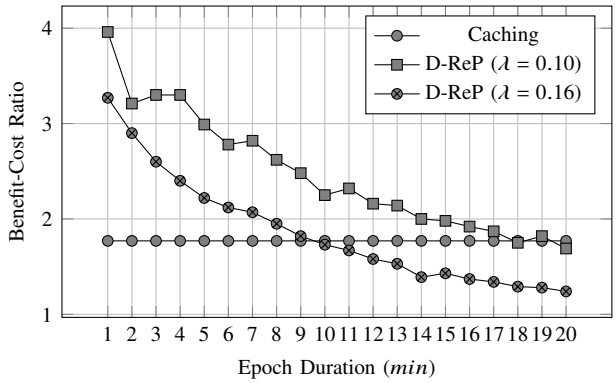

Fig. 12. Benefit-Cost Ratio with Variable Epoch Duration.

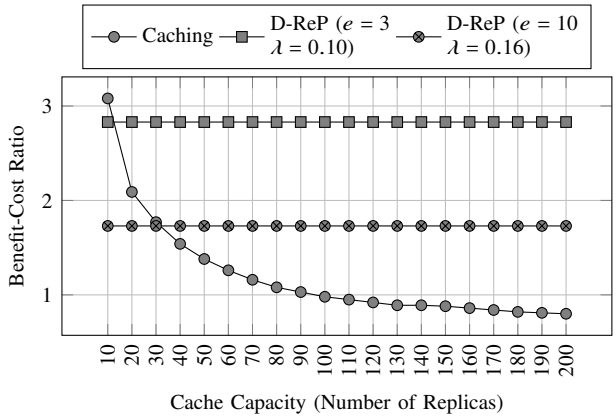

Fig. 13. Benefit-Cost Ratio with Variable Cache Capacity.

Although, we present a constant BCR for caching in these two figures, efficiency of caching also varies by its capacity. Larger capacity means higher possibility of cache hits and thus lower average latency. Figure 13, shows that BCR of caching is significantly lower than D-ReP in most cases. It is only comparable in very small cache capacities where latency improvement is limited. These results indicate that caching can only be effective to reduce latency in a small amount with very low cost. It is an expensive method to achieve significant latency improvements (i.e. $\geq$ 20%).

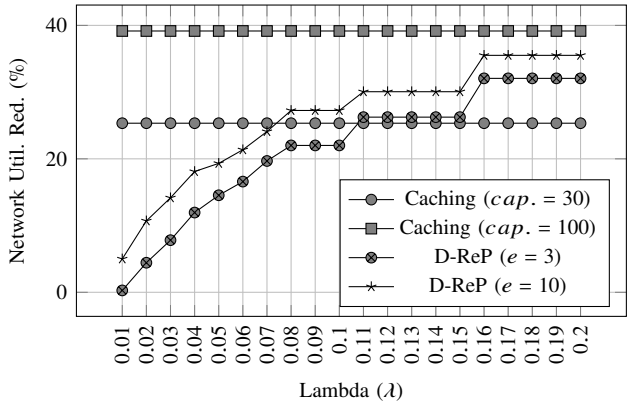

Fig. 14. Reduction in Network Utilization with Variable λ.

TABLE V
PERCENTAGES OF NETWORK OVERHEAD FACTORS

Percentage of Notification Messages	2.63%
Percentage of Failed Requests	.819‰
Percentage of Latency Due to Failure	1.11‰

D. Network Overhead

Both D-ReP algorithm and caching also reduce overall network utilization since some requests are answered from nearby locations. Figure 14 demonstrates the reduction in network utilization relative to single storage method despite additional data transfers stemming from migration and duplication. Results are comparable for most λ values.

Different from caching, D-ReP algorithm requires a replica discovery strategy as described in section IV-C. Notification messages for replica discovery cause a network overhead in addition to regular data requests and responses. We propose a strategy that avoids broadcasting and central control of replica locations so that network overhead is minimized. However, replica discovery is not optimal due to this constraint. That is, some nodes may send their requests to replica locations in their KRL table which are actually removed. This synchronization failure may also incur a network overhead.

Table V presents percentage of notification messages among all messages, percentage of failed requests among all requests, and percentage of latency caused by these failed requests. Our results indicate that all of these are negligible.

E. Convergence

D-ReP is an approximation algorithm and it is not guaranteed to converge to the optimal solution. However, it is possible to consider; (i) its convergence to a point in the search space when user demands are static, and (ii) its behaviour in terms of total number of active replicas in the system when real-world dynamic demands are used. For the latter we provide an experimental evaluation, while we discuss the former theoretically.

Each iteration of the D-ReP algorithm works on a snapshot of data that is collected over the previous epoch. Hence, both `for` loops in lines 1 and 2 of the algorithm in Figure 3 terminate after finite steps even if new replicas are created

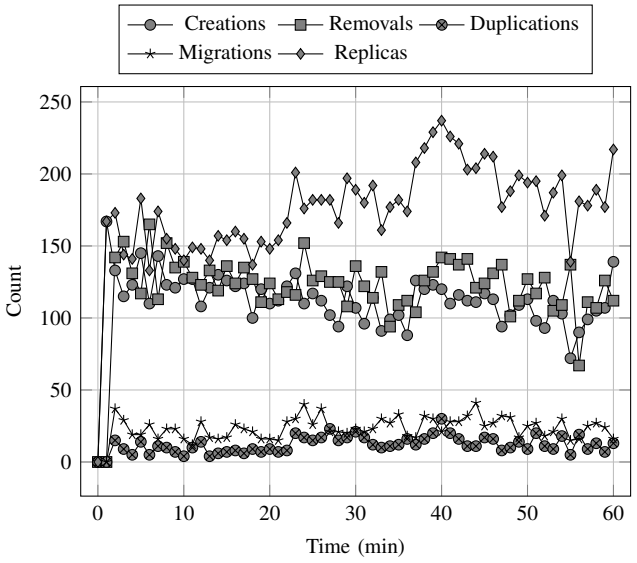

Fig. 15. Number of Replicas and Completed Operations in Time.

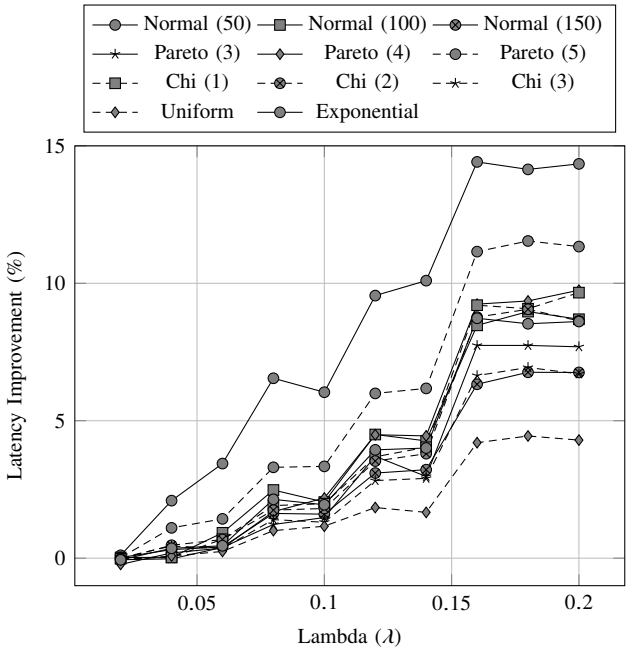

Fig. 16. Latency improvement rate with variable λ.

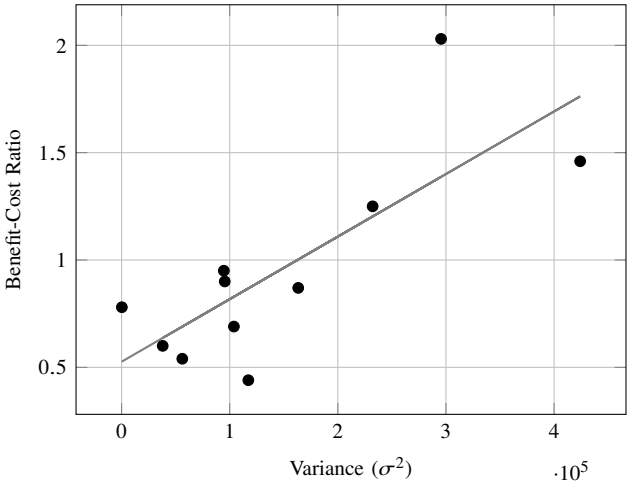

Fig. 17. Variance of distributions and BCR (λ = 0.1).

by other instances in the meantime. In a static environment, the worst case in terms of number of replicas is that each replica is created at each node ($i \times j$ replicas). After this point, domain of the inner `for` loop is an empty set because there is each neighbour n has replica k and tuple (n, k) must be in the KRL table of h. Hence `else` part (lines 12-14) is executed without checking migration and duplication conditions. Since demands are static and $fade\_rate$ is defined in the range [0, 1], `if` condition in line 12 is not satisfied. In conclusion, set of operations to be executed, O, is guaranteed to be empty and optimization (migration, duplication, and removal) stops.

We also evaluate the change in number of replicas present in the system in time using CAIDA traces. Figure 15 presents replica counts in one-minute intervals as well as occurrence of all four operations. For replica removals, the value 0.5 is assigned to the parameter $fade\_rate$. Results show that number of creations and removals are roughly equal and replica count converges around 200 replicas. Although, number of migration and duplication operations are relatively smaller than creations, these are the main factors of D-ReP algorithm that allow replicas to move closer to requesters.

F. Synthetic Data

Results in figure 16 demonstrate that D-ReP yields latency improvements in each and every case of generated request locations. However, rate of improvement depends on the extent of geographical distribution. Normal distribution with a mean of 500 and standard deviation of 50 produces the best latency improvement while uniform distribution produces the worst. BCR results (omitted for brevity) are similar in terms of relative performance order of the distributions.

These results are not unexpected since uniform distribution induces no geographical locality in data requests, i.e. request locations for data objects are simply random. D-ReP specifically makes use of geographical locality and thus has little or no impact for uniform distribution. However, as non-uniformity increases, outcome improves. We used variance of the probability density function (PDF) of a distribution to estimate its uniformity. For instance, PDF of perfect uniform distribution is a constant function with a variance of zero.

To demonstrate the effect of uniformity, figure 17 is presented where variance of a distribution's PDF (not to be confused with variance of distribution itself) is mapped to BCR achieved with that distribution. There is a strong positive linear relationship between these two variables with a Pearson correlation coefficient (r) of 0.78. It can be concluded from these results that D-ReP algorithm is most effective in cases where PDF variance of location distribution is high, or in other words, geographical locations of request are densely clustered in certain areas. This is usually the case in real workloads.

VI. Conclusion

As the volume and velocity of data in cloud is increasing, geographical distribution of where it is produced, processed and consumed is also gaining more significance. It is getting less feasible to move data to a distant data center for processing and move output back to consumer location. In this work, we tackle the problem of latency-aware and cost-efficient placement of data replicas at the edge of the network based on magnitude and location of user demand as well as storage pricing in attempt to reduce data access latency. We present fully decentralized dynamic replica placement algorithm, D-ReP that is based on FLP and requires only local topology information. The algorithm is complemented with a replica discovery method where concerned nodes are notified of nearby replicas.

Experimental results demonstrate effectiveness of D-ReP against non-replicated data source and client side caching in terms of both latency and cost. Decentralized replica placement can either yield the same latency improvement with 14% less additional cost than caching or improve latency by 26% more with the same additional cost, depending on the chosen value of trade-off control parameter. It is also shown that communication overhead and miscommunication errors caused by replica placement and discovery are negligible. Additional results with synthetic usage patterns that are generated with several random distributions generalize our findings and indicate a correlation between algorithm performance and level of geographical locality.

We anticipate that proposed approach will make strongest impact for software services in which data requests or users are clustered in multiple relatively small geographical areas (e.g. within a city). This is due to the fact that moving replicas over long distances in wide area network incurs significant transmission latency and cost. Our experiments with various request distributions support this claim. In the same sense, an additional factor to consider is the size of the replicated data. Some applications that we deem fit in terms of request locality and size include: traffic monitoring and navigation, mobile (live) video streaming, and smart buildings

Further research is needed for replication at the edge of the network, which would fully serve its purpose when the unification of geographically distributed providers becomes prevalent. The most important issue regarding the unification is the standardization of the application programming interfaces by edge providers. On another matter, D-ReP provides best-effort latency improvement and cost reduction. Although it outperforms existing alternatives, certain critical services in areas such as e-health, industrial control, or self-driving cars may require real-time performance guarantees. Hence, it is an open issue to extend D-ReP with such capability.

Acknowledgment

This research was partially supported by İTU-BAP (Grant No: 38450), NETAŞ PhD Project Incentive Award, TÜBİTAK 2211 Graduate Scholarship, and Rucon project (Runtime Control in Multi Clouds), FWF Y 904 START-Programm 2015. Support for CAIDA's Internet Traces is provided by the National Science Foundation, the US Department of Homeland Security, and CAIDA Members. The manuscript was improved thanks to insightful comments by anonymous reviewers.

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[2] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez, "Greening the internet with nano data centers," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 37–48.

[3] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[4] T. Banditwattanawong, M. Masdisornchote, and P. Uthayopas, "Multi-provider cloud computing network infrastructure optimization," *Future Generation Computer Systems*, vol. 55, pp. 116–128, 2016.

[5] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[6] T. Amjad, M. Sher, and A. Daud, "A survey of dynamic replication strategies for improving data availability in data grids," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 337–349, 2012.

[7] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha, "Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 292–308.

[8] F. Chen, K. Guo, J. Lin, and T. La Porta, "Intra-cloud lightning: Building CDNs in the cloud," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 2012, pp. 433–441.

[9] K. Ranganathan and I. Foster, "Identifying Dynamic Replication Strategies for a High-Performance Data Grid," in *Proceedings of the 2nd International Workshop on Grid Computing*, 2001, pp. 75–86.

[10] S. Jin and A. Bestavros, "GreedyDual* Web caching algorithm: exploiting the two sources of temporal locality in Web request streams," *Computer Communications*, vol. 24, no. 2, pp. 174–183, 2001.

[11] J. Liao, F. Trahay, G. Xiao, L. Li, and Y. Ishikawa, "Performing Initiative Data Prefetching in Distributed File Systems for Cloud Computing," *IEEE Transactions on Cloud Computing*, 2015.

[12] "The CAIDA UCSD Anonymized Internet Traces 2015 - [2015-02-19]," URL: http://www.caida.org/data/passive/passive_2015_dataset.xml.

[13] G. Liu, H. Shen, and H. Chandler, "Selective data replication for online social networks with distributed datacenters," in *21st IEEE International Conference on Network Protocols (ICNP)*. IEEE, 2013, pp. 1–10.

[14] K. Kalpakis, K. Dasgupta, and O. Wolfson, "Optimal placement of replicas in trees with read, write, and storage costs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 628–637, 2001.

[15] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, "On the placement of web server replicas," in *Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 3. IEEE, 2001, pp. 1587–1596.

[16] M. Szymaniak, G. Pierre, and M. Van Steen, "Latency-driven replica placement," in *The 2005 Symposium on Applications and the Internet*. IEEE, 2005, pp. 399–405.

[17] P. Liu and J.-J. Wu, "Optimal replica placement strategy for hierarchical data grid systems," in *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, vol. 1. IEEE, 2006, pp. 420–423.

[18] A. Benoit, V. Rehn-Sonigo, and Y. Robert, "Replica placement and access policies in tree networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 12, pp. 1614–1627, 2008.

[19] Y. Rochman, H. Levy, and E. Brosh, "Resource placement and assignment in distributed network topologies," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 2013, pp. 1914–1922.

[20] M. Ali, K. Bilal, S. Khan, B. Veeravalli, K. Li, and A. Zomaya, "DROPS: Division and Replication of Data in the Cloud for Optimal Performance and Security," *IEEE Transactions on Cloud Computing*, 2015.

[21] K. Deng, K. Ren, M. Zhu, and J. Song, "A Data and Task Co-scheduling Algorithm for Scientific Cloud Workflows," *IEEE Transactions on Cloud Computing*, 2015.

[22] L. C. Lizhen, J. Zhang, L. Yue, Y. Shi, H. Li, and D. Yuan, "A Genetic Algorithm Based Data Replica Placement Strategy for Scientific Applications in Clouds," *IEEE Transactions on Services Computing*, 2015.
[23] J. R. Douceur and R. P. Wattenhofer, "Competitive hill-climbing strategies for replica placement in a distributed file system," in *International Symposium on Distributed Computing*. Springer, 2001, pp. 48–62.
[24] M. Tang, B.-S. Lee, C.-K. Yeo, and X. Tang, "Dynamic replication algorithms for the multi-tier data grid," *Future Generation Computer Systems*, vol. 21, no. 5, pp. 775–790, 2005.
[25] X. Tang and J. Xu, "QoS-aware replica placement for content distribution," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 10, pp. 921–932, 2005.
[26] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated Data Placement for Geo-Distributed Cloud Services," in *NSDI*, vol. 10, 2010, pp. 28–0.
[27] L. M. Khanli, A. Isazadeh, and T. N. Shishavan, "PHFS: A dynamic replication method, to decrease access latency in the multi-tier data grid," *Future Generation Computer Systems*, vol. 27, no. 3, pp. 233–244, 2011.
[28] V. Andronikou, K. Mamouras, K. Tserpes, D. Kyriazis, and T. Varvarigou, "Dynamic QoS-aware data replication in grid environments based on data 'importance'," *Future Generation Computer Systems*, vol. 28, no. 3, pp. 544–553, 2012.
[29] M.-C. Lee, F.-Y. Leu, and Y.-p. Chen, "PFRF: An adaptive data replication algorithm based on star-topology data grids," *Future Generation Computer Systems*, vol. 28, no. 7, pp. 1045–1057, 2012.
[30] N. Saadat and A. M. Rahmani, "PDDRA: A new pre-fetching based dynamic data replication algorithm in data grids," *Future Generation Computer Systems*, vol. 28, no. 4, pp. 666–681, 2012.
[31] J.-W. Lin, C.-H. Chen, and J. M. Chang, "QoS-aware data replication for data-intensive applications in cloud computing systems," *IEEE Transactions on Cloud Computing*, vol. 1, no. 1, pp. 101–115, 2013.
[32] X. Fan, X. Ma, J. Liu, and D. Li, "Dependency-aware data locality for MapReduce," in *IEEE 7th International Conference on Cloud Computing*. IEEE, 2014, pp. 408–415.
[33] M. Hu, J. Luo, Y. Wang, and B. Veeravalli, "Practical resource provisioning and caching with dynamic resilience for cloud-based content distribution networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 8, pp. 2169–2179, 2014.
[34] C.-A. Chen, M. Won, R. Stoleru, and G. G. Xie, "Energy-efficient fault-tolerant data storage and processing in mobile cloud," *IEEE Transactions on cloud computing*, vol. 3, no. 1, pp. 28–41, 2015.
[35] C. Papagianni, A. Leivadeas, and S. Papavassiliou, "A cloud-oriented content delivery network paradigm: Modeling and assessment," *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 5, pp. 287–300, 2013.
[36] N. Bonvin, T. G. Papaioannou, and K. Aberer, "A self-organized, fault-tolerant and scalable replication scheme for cloud storage," in *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010, pp. 205–216.
[37] K. Sashi and A. S. Thanamani, "Dynamic replication in a data grid using a Modified BHR Region Based Algorithm," *Future Generation Computer Systems*, vol. 27, no. 2, pp. 202–210, 2011.
[38] X. Liao, H. Jin, and L. Yu, "A novel data replication mechanism in P2P VoD system," *Future Generation Computer Systems*, vol. 28, no. 6, pp. 930–939, 2012.
[39] 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.
[40] Y. Chen, R. H. Katz, and J. D. Kubiatowicz, "Dynamic replica placement for scalable content delivery," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 306–318.
[41] H. Shen, "An efficient and adaptive decentralized file replication algorithm in P2P file sharing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 6, pp. 827–840, 2010.
[42] P. Pantazopoulos, M. Karaliopoulos, and I. Stavrakakis, "Distributed placement of autonomic internet services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1702–1712, 2014.
[43] G. Smaragdakis, N. Laoutaris, K. Oikonomou, I. Stavrakakis, and A. Bestavros, "Distributed server migration for scalable internet service deployment," *IEEE/ACM Transactions on Networking*, vol. 22, no. 3, pp. 917–930, 2014.
[44] M. Karlsson and C. Karamanolis, "Choosing replica placement heuristics for wide-area systems," in *Proceedings of the 24th International Conference on Distributed Computing Systems*. IEEE, 2004, pp. 350–359.
[45] J. Ma, W. Liu, and T. Glatard, "A classification of file placement and replication methods on grids," *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1395–1406, 2013.
[46] R. K. Grace and R. Manimegalai, "Dynamic replica placement and selection strategies in data gridsâĂŤa comprehensive survey," *Journal of Parallel and Distributed Computing*, vol. 74, no. 2, pp. 2099–2108, 2014.
[47] Q. Zhang, S. Li, Z. Li, Y. Xing, Z. Yang, and Y. Dai, "CHARM: A Cost-Efficient Multi-Cloud Data Hosting Scheme with High Availability," *IEEE Transactions on Cloud Computing*, vol. 3, no. 3, pp. 372–386, 2015.
[48] S. Wu, K.-C. Li, B. Mao, and M. Liao, "DAC: Improving storage availability with Deduplication-Assisted Cloud-of-Clouds," *Future Generation Computer Systems*, 2016.
[49] M. T. Melo, S. Nickel, and F. Saldanha-Da-Gama, "Facility Location and Supply Chain Management – A Review," *European Journal of Operational Research*, vol. 196, no. 2, pp. 401–412, 2009.
[50] S. H. Owen and M. S. Daskin, "Strategic Facility Location: A Review," *European Journal of Operational Research*, vol. 111, no. 3, pp. 423–447, 1998.
[51] S. Guha and S. Khuller, "Greedy Strikes Back: Improved Facility Location Algorithms," *Journal of Algorithms*, vol. 31, no. 1, pp. 228–248, 1999.
[52] A. Aral and T. Ovatman, "Network-Aware Embedding of Virtual Machine Clusters onto Federated Cloud Infrastructure," *Journal of Systems and Software*, vol. 120, pp. 89–104, 2016.
[53] T. Moscibroda and R. Wattenhofer, "Facility location: distributed approximation," in *Proceedings of the 24th ACM symposium on Principles of distributed computing*. ACM, 2005, pp. 108–117.
[54] W. Zhao, X. Xu, N. Xiong, and Z. Wang, "A weight-based dynamic replica replacement strategy in data grids," in *3rd IEEE Asia Pacific Services Computing Conference*. IEEE, 2008, pp. 1544–1549.
[55] M. Teng and L. Junzhou, "A prediction-based and cost-based replica replacement algorithm research and simulation," in *19th International Conference on Advanced Information Networking and Applications*, vol. 1. IEEE, 2005, pp. 935–940.
[56] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.
[57] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.
[58] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An approach to universal topology generation," in *Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 2001, pp. 346–353.
[59] L. B. Lave, *Benefit-Cost Analysis*. AEI Press, 1996.

Atakan Aral is a Postdoctoral Research Fellow at the Institute of Information Systems Engineering, Vienna University of Technology. He received a dual MSc degree in Computer Science and Engineering from Politecnico di Milano in 2011 and Istanbul Technical University (ITU) in 2012, and a PhD degree in Computer Engineering from ITU in 2016. His research interests center around resource management for distributed and virtualized computing systems such as intercloud and edge.

Tolga Ovatman is working as an Associate Professor in the Computer Engineering Department of Istanbul Technical University (ITU), Turkey. He received his B.Sc. degree in computer engineering from Hacettepe University, Turkey, in 1999 and his M.Sc. and Ph.D. degrees in computer engineering from ITU in 2005 and 2011, respectively. His research interests include cloud computing, model checking, parallel programming and object-oriented design.