Location-aware Associated Data Placement for Geo-distributed Data-intensive Applications

Boyang Yu and Jianping Pan
University of Victoria, Victoria, BC, Canada

***Abstract*—Data-intensive applications need to address the problem of how to properly place the set of data items to distributed storage nodes. Traditional techniques use the hashing method to achieve the load balance among nodes such as those in Hadoop and Cassandra, but they do not work efficiently for the requests reading multiple data items in one transaction, especially when the source locations of requests are also distributed. Recent works proposed the managed data placement schemes for online social networks, but have a limited scope of applications due to their focuses. We propose an associated data placement (ADP) scheme, which improves the co-location of associated data and the localized data serving while ensuring the balance between nodes. In ADP, we employ the hypergraph partitioning technique to efficiently partition the set of data items and place them to the distributed nodes, and we also take replicas and incremental adjustment into considerations. Through extensive experiments with both synthesized and trace-based datasets, we evaluate the performance of ADP and demonstrate its effectiveness.**

I. Introduction

Recent Internet developments encourage the emerging of data-intensive applications [1], which leads to a large amount of data stored in the geographically distributed datacenters across regions and a high frequency of data access from the clients or users [2]. Each data request from the users can be fulfilled only at the datacenter holding the requested data or its replicas, so the storage location of the data items really matters to the user-experienced performance. Meanwhile, the optimized data placement can help the service providers by lowering their cost with the improved system efficiency and ensuring the high system availability. Thus we face a general problem of how to efficiently place the data items and replicas to the distributed datacenters, with the requests for them varying across different regions.

Storing the requested data closer to the users is the motivation of most existing work on data placement, which helps to reduce the latency experienced by the users and lower the relaying traffic among datacenters. In the network applications that can utilize geographically distributed datacenters, Content Distribution Network (CDN) [3] is broadly applied to help the access of videos, pictures and texts. Different from traditional CDNs where each content is replicated to every datacenter, people need to constrain the number of replicas allowed for each data item under the increasing scale of data items and data traffic. That leads to the problem of how to choose the proper datacenters to store the replicas with a limited number.

The issue of multi-get hole [4] has not been paid enough attention to until recently, but it can also affect the decision in placing data items to distributed locations. The issue is triggered when multiple items are requested in one transaction. People discovered that using fewer nodes to fulfill such a request is better in terms of the system efficiency, because the request dispatched to each node will introduce a certain overhead to the node regardless of the amount of data requested. To overcome the issue, a favorable paradigm is to place the strongly associated data items, those who are often requested together in the same transaction, at the same location. It has profound applications to Online Transaction Processing systems, where a transaction is fulfilled only after accessing multiple data tables, to Online Social Network (OSN) services, where the polling of news feeds involves not only the data of a user itself but also that of its friends, and even to regular web services, since visiting a webpage actually needs to download multiple files, such as the hypertexts, images and scripts.

The balance of the workload or stored items among distributed datacenters is another factor important to the data placement. The maximum serving capacity in a modern datacenter seems to be unlimited to an individual service provider, especially when designed as a public cloud. However, a poor balance between datacenters would result in the excessive dependence on some critical locations, which increases the worst-case recovery time after a site failure, and therefore should be avoided as much as possible. Here we only consider the balance among the number of data items (or replicas) placed in different datacenters, but it can partially reflect the balance of workload because of the law of large numbers, assuming the randomness of request rates.

It is challenging to solve a data placement problem that combines these different aspects. We will address the corresponding issues progressively from three scenarios. First, for the *scenario without replicas*, we address the placement problem by the hypergraph partitioning formulation, through which the optimized solution can be obtained. Second, we consider the *scenario with replicas* where a certain number of replicas are allowed for each item. The existence of multiple replicas in different locations introduces the decision problem of routing a request, because any one of the corresponding replicas can be used to fulfill the request. A round-robin scheme that iteratively makes the routing decision and replica placement decision is proposed. Third, we consider that the workload may change after a certain time, which makes the earlier solution sub-optimal. Then the problem of *replica migration* is formulated, i.e., how to change the locations of no more than K replicas based on the existing placement to achieve a better performance. To solve the problem, we

978-1-4799-8381-0/15/$31.00 ©2015 IEEE

discuss how to generate the candidates for replica migration and determine the optimal number of selected candidates.

The state-of-the-art implementation in most distributed storage systems today, such as HDFS [5] and Cassandra [6], is mainly hash-based which can be intuitively understood as random placement. Among the related work that discussed the managed data placement, some existing work either just focused on the distance between data and user, such as [7] and [8], or only addressed the co-location of associated data items, such as [9]–[11]. Some others discussed the issues under the scenario of OSN [12]–[14], where the discussed association of stored entities is only based on pairs of users (each pair consists of a user and one of its friends), which overlooks the relationship between concrete data items as well as the association that involves more than two entities. [15] is the work most related to ours in the problem definition, but it relaxed the group association to pairwise in an early stage of the solution. To the best of our knowledge, our data placement scheme makes the advancement of jointly improving the co-location of associated data and localized data serving without a relaxation of the original objective. Besides, the developed methods to support replicas and to address replica migration are also novel and make the scheme more comprehensive.

The rest of this paper is structured as follows. Section II summarizes some related work. Section III presents our modeling framework for the data placement problem. In Section V, the location-aware associated data placement scheme is proposed. In Section VI, the scheme is evaluated through extensive experiments. Section VII concludes the paper.

II. Related Work

Due to the availability of geo-distributed datacenters, there exist multiple choices in selecting the location to store data or the destination to fulfill user requests. In [15], Agarwal et al. presented the automatic data placement across geo-distributed datacenters, which iteratively moves a data item closer to both clients and the other data items that it communicates with. In [16], the replica placement in distributed locations was discussed with the QoS considerations. In [7], Rochman et al. discussed how to place the contents or resources to distributed regions in order to serve more requests locally for a lower cost. In [8], Xu et al. solved the workload management problem, in order to maximize the total utility of serving requests minus the cost, which is achieved through the reasonable request mapping and response routing.

Besides fulfilling the request at a location near to the user, there are other factors that affect the system performance, such as the multi-get hole effect, firstly cited in [4]. The effect can be intuitively explained as follows: for a request to obtain multiple items which are stored distributedly, we favor to use fewer nodes to fulfill the request for a higher system efficiency. It incurs the problem of how to co-locate the strongly correlated data items through the managed data placement. [9] proposed to create more replicas of data items to increase the chance of serving more requested items in one node. [10] mentioned that frequent request patterns can be discovered through trace analysis, and each group of items frequently accessed can be treated as a whole in the distributed storage. [17] discussed how to co-locate the related file blocks in the Hadoop file system. [11] improved the hypergraph partitioning by compression to more efficiently partition the data items into sets. Such work did not consider the necessity of fulfilling the request locally and was not aware of the difference among locations, which result in that many requests might need to be served at a remote node inefficiently.

Recent research has studied the data and replica placement in OSN, which favors to place data items from close friends together. In [12], Pujol et al. showed the necessity to co-locate the data of a user and its friends and proposed the dynamic placement scheme. In [13], it was stated that different data items from the same user may be requested with a heterogeneous rate or pattern, and the method to determine the proper number of replicas for each data item was given. In [14], Jiao et al. summarized the relationships of entities in the OSN system and proposed the multi-objective data placement scheme. The friendship-based one-to-one data relationship discussed in these works can be considered as a special case of the multi-data association in our modeling.

III. Modeling Framework

A. Data Items and Nodes

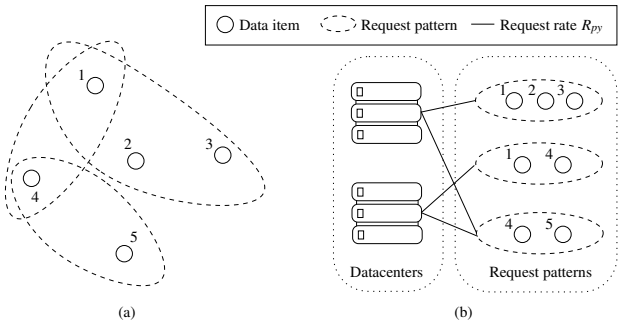

Fig. 1. Problem inputs: (a) request pattern set P; (b) request rate set R

A set X of m *data items* is used to represent the data stored in the system. Depending on the actual type of data storage, the data items can be files, tables or segments in practice. Users of the service may request at most d different items from the set X in each transaction. We denote the space of *request patterns* by $\mathbb{P} = \{X, \varnothing\}^d$. The requests in a practical system actually fall in a subset of the space, denoted by

$$P \subset \mathbb{P} = \{X, \varnothing\}^d \ . \tag{1}$$

As illustrated in Fig. 1a, the example system contains 5 different data items and 3 request patterns, which are $(1, 2, 3)$, $(1, 4)$ and $(4, 5)$. The paradigm of accessing multiple data items in one transaction has many applications. For example, the news feed updating in OSN involves the data of multiple users. In data analysis systems, the output is made through combining or processing multiple data files, each captured from a different data source, possibly remotely distributed.

We consider the scenario that the data items are stored in geographically distributed datacenters, represented by a set Y of n nodes. Compared with the centralized storage, the distributed storage can improve the access latency of most users and achieve a higher level of fault tolerance. We term a *datacenter* as a *node* or a *location* interchangeably in the following. Initially, we consider each data item $x \in X$ is stored at a unique location $y \in Y$. Thus the data-to-location mapping function is defined as

$$\mathcal{D} : x \to y \ , \tag{2}$$

which specifies the storage location y of each item x. Fundamentally, our work focuses on designing the placement scheme that provides a reasonable solution of $\mathcal{D}$. Besides, we use D_y to represent the set of data items stored in node y.

In the state-of-the-art implementation of such a function, the hash-based methods are adopted most broadly, such as in HDFS and Cassandra, since at the time of their design, the main concern was to achieve the load balance between nodes. Although some other policies take effect in their data placement, such as to avoid placing the replicas of a data item in the same datacenter in order to improve the fault tolerance, the schemes can still be understood as random placement. Obviously the hash-based schemes did not pay enough attention to the system performance affected by the data locations and ignored the potential performance improvement through the managed data placement. To attack this deficiency, we start with modeling the metrics of placing data at different locations and the effect of introducing data replicas, and then propose the efficient data placement scheme that fully exploits the benefits of the managed data placement.

B. Data Placement Problem

1) Workload: We assume the request from a client is directed to the datacenter closest to the location of the client. The client can be located outside or inside the datacenters for data storage. When a human user is using the service, they are always from outside locations. Another case is the running jobs or processes inside the datacenter who also can request data items. Without loss of generality, we consider the datacenter that a request is initially accessed to as the source location of the request. So the datacenter or node in our modeling has two roles simultaneously: the *source location of requests* and the *destination location holding the stored data*.

The workload or request rate of each pattern $p \in P$ from the requesting node $y \in Y$ can be measured, denoted by R_{py}. We use the predicted request rates as the input of our scheme to make the data placement decision. There exist mature methods in predicting the rates from a series of rate measurements in the previous time slots, such as EWMA [18], so the details of dealing with prediction are not included in this paper. Although the predicated rates are not the same as the measured ones, we use the same notations in the following for simplicity. First we denote the *workload* or *request rate* set as

$$R = \{R_{py} | p \in P, y \in Y\} \ . \tag{3}$$

In the example of Fig. 1b, the request rate set R is illustrated as a bipartite graph, where datacenters and request patterns are the two sets of vertices, and the edges between these two sets are weighted by the rate R_{py}. Besides, for each data item x, we calculate its total request rate at each node y by

$$R_{xy} = \sum_{p \in P} R_{py} \mathbf{1}(x \in p) \ , \tag{4}$$

where $\mathbf{1}(x \in p)$ indicates whether the data item x is a member of the pattern p, returning 1 if true or 0 otherwise. For each pattern p, we calculate its total request rate by

$$R_p = \sum_{y \in Y} R_{py} \ . \tag{5}$$

2) Metrics: Data placement can affect the system performance in both the system efficiency and user-experienced latency. We investigate the relationships between the placement and system performance and summarize them as two metrics.

Co-location of Associated Data: The system efficiency is characterized by the necessary system time to fulfill the given workload. According to the observation of [4], in distributed systems, the average system time of a request is not only related to amount of information accessed, but also related to the number of distributed nodes involved due to the processing overhead at each node. Denote the span of a request p by S_p, representing the number of items requested by it, and denote the number of items in request p that are fulfilled at node y by S_{py}, since a single node may not be able to provide all the items in p. They have the relationship of $\sum_{y \in Y} S_{py} = S_p$. Note that S_{py} is a variable determined by the mapping function $\mathcal{D}$ and S_p is a constant. We model the necessary system time to partially or fully fulfill a request p at node y as

$$S_{py} + \lambda \cdot \mathbf{1}(S_{py}) \ , \tag{6}$$

where the 0-1 function $\mathbf{1}(S_{py})$ indicates whether $S_{py} \geq 1$. The idea behind it is that the system time consists of the part proportional to S_{py} and the constant overhead of handling the request, denoted by λ. The latter is introduced by the routine operations in handling a request, such as the TCP connection establishment. With the request rates of different patterns, the total system time of fulfilling all the requests is $\sum_{p \in P} R_p \sum_{y \in Y} [S_{py} + \lambda \cdot \mathbf{1}(S_{py})]$, which is equal to

$$\sum_{y \in Y} \sum_{p \in P} R_p [S_{py} + \lambda \cdot \mathbf{1}(S_{py})] \ . \tag{7}$$

Minimizing (7) helps to lower the cost of the service provider, i.e., the amount of resource allocation or the monetary expense on clouds. It can be achieved by the co-location of strongly associated data. For example, in the extreme case, when placing all the items in a pattern p to the same location, the system time on p is at the lower bound $R_p(S_p + \lambda)$.

Because $\sum_{y \in Y} \sum_{p \in P} R_p \cdot S_{py}$ in (7) is a constant for any given workload, we can summarize the metric as

$$C^A = \sum_{y \in Y} \sum_{p \in P} R_p \cdot \lambda \cdot \mathbf{1}(S_{py}) \ . \tag{8}$$

Localized Data Serving: The location difference between the requesting node and the node holding the requested data affects the system performance through incurring the relaying traffic and enlarging the user-experienced latency. To be specific, when the data serving node is not the same as the requesting node, the relaying traffic is introduced, so we use the total relaying traffic as the second metric, denoted by

$$C^L = \sum_{x \in X} \sum_{y \in Y} R_{xy} \cdot [1 - \mathbf{1}(x \in D_y)] \ , \tag{9}$$

where $\mathbf{1}(x \in D_y)$ indicates whether x is stored at node y. Here we use the *relaying traffic* as the metric of localized data serving, but it is still possible to use the *logical distance* between each user and its requested data as the metric instead, which is not included in the paper due to the page limit.

Note that [14] summarized the relationships in the data placement for OSN as two categories: (a) between data and data; (b) between data and node. Our work is different at (a), as we consider each request may involve a group of multiple data. We cannot adapt to their scheme through simply decomposing the multi-data relationship to the pairwise relationships of all the pairs in the group. Due to the generalized hypegraph formulation shown later, our scheme can be extended to incorporate most of the metrics considered in that paper.

3) Optimization Problem: Our objective is to improve the performance of a placement $\mathcal{D}$, represented by

$$C(\mathcal{D}) = C^A + \alpha C^L \ , \tag{10}$$

where α is used to tradeoff the two metrics above. To minimize $C(\mathcal{D})$ helps to improve the system efficiency and the user-experienced latency. Besides, the balance among nodes is also necessary. To ensure the worst-case recovery time upon site failure, the number of data items stored in different datacenters should be balanced. Therefore, we set the *balance constraint* that the number of items stored in each datacenter y, should be in a range $[(1-\epsilon)h_a, (1+\epsilon)h_a]$, where h_a is the average number of items in all nodes and ϵ is the balance parameter. The formulated problem can be generalized as: given $\mathcal{I} = \{P, R\}$, find the optimal placement solution of $\mathcal{D} : x \to y$ that minimizes the value of $C(\mathcal{D})$, subject to the balance constraint.

C. Data Replicas

The data placement problem is more challenging if the replicas of data items are allowed. We would not differentiate the data item itself and its replicas, so both of them are treated as replicas below. The allowed number of replicas for each item is given from a separate process not discussed here, due to the scope of this paper. Below we assume the number of replicas for each data item is k, but the scheme still applies when the number is heterogenous for different items. Because the locations of replicas need to be determined, the data-to-location mapping function is updated to

$$\mathcal{D} : x \to \{y_1, y_2, ..., y_k\} \ . \tag{11}$$

Meanwhile, we have to face the routing decision problem, since the request for an item x can be fulfilled at any location holding a replica of x. We adopt the deterministic routing and represent the routing as a mapping function

$$\mathcal{M} : (x, p, y) \to y_d \ , \tag{12}$$

which can give the routing destination $y_d \in Y$ for each item x in a pattern p requested from node y. Note that the solution for the scenario with replicas should include both $\mathcal{D}$ and $\mathcal{M}$. Besides, we realize that the routing decision should be made based on a given placement of replicas, however after the decision is made, the previously generated placement might be sub-optimal. This makes the problem more complicated and we will present our solution gradually in the following.

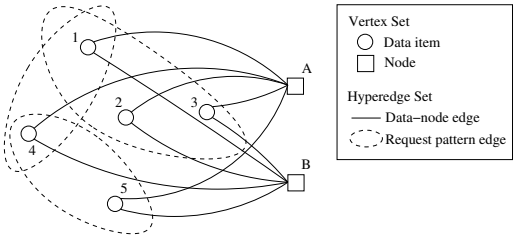

Fig. 2. Hypergraph formulation

IV. Associated Data Placement

A. Placement without Replicas

We start with showing that without data replicas, the optimization problem can be formulated as an n-way hypergraph partitioning problem. Note that in the existing work that also used hypergraph to model to the relationship among multiple items, e.g., [11], the exact data location was not addressed because of their unawareness of the difference between locations. A hypergraph $G(V, E)$ is a further generalization of a graph, i.e., the hypergraph allows each of its hyperedges to involve multiples vertices while the edge of an ordinary graph can only involve two vertices at most. In our scheme, we set up the vertex set V with all the data items and all the nodes in the considered system, such as

$$V = \{X, Y\} \ . \tag{13}$$

The hyperedge set E contains all the request patterns and the pairs between each node and each data item. For each request pattern hyperedge, it involves multiple data items and that is the main reason of introducing hypergraph. Formally,

$$E = \Big\{ \{e_p | p \in P\}, \{e_{xy} | x \in X, y \in Y\} \Big\} \ . \tag{14}$$

Each hyperedge $e \in E$ is assigned a weight. Due to the objective (10), the weights are set according to

$$w_e = \begin{cases} \lambda R_p, & \text{for the request pattern hyperedge } e_p \\ \alpha R_{xy}, & \text{for the data-node hyperedge } e_{xy} \end{cases} \ . \tag{15}$$

An example of formulating the problem into a weighted hypergraph is illustrated in Fig. 2. In the hypergraph, there are two types of vertices: storage node (square) and data item (circle), and two types of edges, the request pattern hyperedge

(dashed circle) and the data-node hyperedge (solid line). We may term hyperedge as edge for short below.

An n-way partitioning of the hypergraph is to partition its vertices into n output sets, such that each vertex only belongs to one of the n sets. The *cut weight of the partitioning* is counted as the sum of the cut weights of its hyperedges. A hyperedge e is cut if its vertices fall to more than one sets; the *cut weight of edge* e is counted as $(t-1)w_e$ if its vertices fall to t sets. Meanwhile we can pre-assign some vertices to the n output sets before applying the hypergraph partitioning algorithm, i.e., they are fixed-location vertices. In our scheme, each of the n nodes is pre-assigned to a different set before the partitioning. By that, after the partitioning, we obtain where to place a data directly from the n output sets, because each node and the data items stored in it would fall to the same set. Next we show the equivalence between the cut weight of an n-way hypergraph partitioning and the objective value $C(\mathcal{D})$ of the data placement based on the partitioning result.

Theorem 1: For any input $\mathcal{I}$, we can formulate it into a hypergraph through the method above. Partition the hypergraph into n sets of vertices, from which, we can obtain the data placement $\mathcal{D}$. Denoting the cut weight of the partitioning by H, it satisfies that $H = C(\mathcal{D}) - B$, where B is a constant.

Proof: First, we discuss the cut weight of a request pattern edge e_p, denoted by H_p. According to the definition of the cut of hyperedges, $H_p = [\sum_{y\in Y} \mathbf{1}(S_{py}) - 1]w_{e_p} = [\sum_{y\in Y} \mathbf{1}(S_{py}) - 1]\lambda R_p$. With (8), We can obtain $C^A - \sum_{p\in P} H_p = \sum_{p\in P} \lambda R_p = B$, which is a constant. Second, we discuss the cut weight of a data-node edge e_{xy}, denoted by H_{xy}. For any data item x, it was connected to all the nodes in the formulated hypergraph. After the partitioning, it can be connected to only one node, because otherwise some sets in the partitioning result would be connected by x, considering that we have assigned each node to a different set. Assuming the item x is finally connected with node f_x, the sum of the cut weights of data-node edges related to x is $\sum_{y\in Y} H_{xy} = \sum_{y\in Y/f_x} \alpha R_{xy}$. After placing the data items according to the partitioning, with (9), we obtain the metric $C^L = \sum_{x\in X}\sum_{y\in Y/f_x} R_{xy}$. Therefore, $H = \sum_{p\in P} H_p + \sum_{x\in X}\sum_{y\in Y} H_{xy} = C^A - B + \alpha C^L = C(\mathcal{D}) - B$. ∎

From the thereom, to minimize the cut weight of the hypergraph partitioning is the same as to minimize the objective function (10). The minimum n-way hypergraph partitioning has been shown to be NP-Hard, but different heuristics have been developed to solve the problem approximately, because of the wide applications of the hypergraph partitioning, such as in VLSI, data mining and bioinformatics. The PaToH tool [19] is what we use to partition the formulated hypergraph.

The general steps of the algorithm in PaToH [19] are as follows: 1) coarsen the initial hypergraph into smaller and smaller scales gradually; 2) solve the partitioning problem on the smallest scale graph; 3) gradually uncoarsen the partitions into the larger scale graph with refinements. The coarsening process may eliminate the chance of placing some vertices in the same set, but with a reasonable coarsening and later refinement, its performance can be still quite well.

B. Placement with Replicas

To overcome the issues introduced by the replicas, we design the round-robin scheme as shown in Fig. 3. In phase (1), we address the initial placement of data replicas by a simple greedy method. In phase (2), the local routing decision is made for each request pattern from each node, considering the existence of replicas. Then the request pattern p attached with each request rate R_{py} is refined towards specific replicas. In phase (3), based on the refined request rates towards replicas, we can make the replica placement decision. Phase (2) and (3) are applied repeatedly until the improvement is smaller than a threshold. The general steps of the whole scheme are shown in Alg. 1. Details of each phase are given as follows.

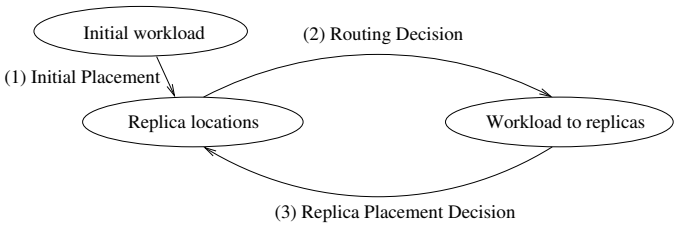

Fig. 3. Logic flow of the scheme with replicas

1) Initial Placement: First we present a greedy method of generating the initial replica placement, which is illustrated as phase (1) in Fig. 3. For each data $x \in X$, we obtain the set $W_x = \{R_{xy} | y \in Y\}$, representing the request rates to data x from different locations, and sort it to the descending order. Based on the allowed number of replicas for data x, which is k in our model, we choose k nodes that are with the highest rate in W_x to store the replicas of item x. This initial placement at least ensures that the resultant total relaying traffic of the system is minimized, which is better than a random initial placement. Although the metric of data co-location is not considered in this phase, the final performance of the scheme is not affected, since that metric takes effect in the later steps.

2) Routing Decision: A main problem introduced by allowing replicas is to make the optimal routing decision based on the current status of the replica placement, which is shown as phase (2) in Fig. 3. We formulate the optimization problem for the routing of pattern p at node j as follows.

$$\begin{aligned} \min\ & \lambda \sum_{y\in Y} A_y + \alpha \sum_{x\in p}\sum_{y\in Y/j} X_{xy} \\ \text{s.t.}\ & \sum_{y\in Y} X_{xy} \cdot \mathbf{1}(x \in D_y) = 1, \forall x \in p \\ & X_{xy} \le A_y, X_{xy} \in \{0,1\}, A_y \in \{0,1\}, \forall x \in p \end{aligned} \tag{16}$$

The optimal solution of (16) ensures the minimized value of (10) under any given replica placement. The binary variable X_{xy} is used to represent whether an item $x \in p$ will be routed to the node y. And the binary variable A_y indicates whether the node y is utilized (or active) in the routing of p. The constraints ensure that each item $x \in p$ is actually routed to one node holding the replica of x and being active. The objective is to minimize the cost incurred by the fulfillment

of request p from node j. Its first part is about the number of nodes involved. The second part is the relaying traffic in fulfilling p. Indeed they would contribute to the objective (10).

We can generalize a feasible solution to (16) as two cases: the local node j *is* or *is not* included in the chosen active data serving nodes. For the former, we prefer to choose any item $x \in p$ to be served locally if possible, because otherwise, moving the serving to the other locations would introduce the extra relaying traffic. The latter is still necessary, because if we assume that all the items are not served at the local node j, it saves the cost of λA_j compared with the former, which may be able to compensate for the extra relaying traffic.

No matter either case is chosen, the amount of relaying traffic would be determined, and then the objective becomes $\lambda \sum_{y \in Y} A_y$, which makes the problem to the classical set-cover problem. The set-cover problem is proved to be NP-hard, however we can use any Integer Linear Programming solver or heuristics to obtain the approximate solution. In our method, we obtain the solution of two cases first and then choose the one with a lower objective value for (16). For the former case, we choose data items to be served locally as much as possible and then apply the set-cover heuristic to the remaining items. For the latter, we directly apply the set-cover heuristic to all the data items. The heuristic is to iteratively choose the node that covers the highest number of data items not covered yet by the nodes chosen in the earlier iterations.

3) Placement Decision: The placement decision is obtained through extending the solution for the case without replicas. We denote the set of replicas by Z and denote a replica by z.

After phase (2), because the routing decision $\mathcal{M}$ is obtained, we can determine the request rate to each replica. We refine the workload set from $R = \{R_{py}\}$ to $R' = \{R_{p'y}\}$, which is shown as $FuncR$ in Alg. 1. The difference between p and p' is that p' is based on the replica space. Formally, $p' \in \{Z, \varnothing\}^d$. Specifically, p can only indicate whether a data item x is in the pattern p, but p' indicates which specific replica of each item $x \in p$ is actually involved in fulfilling the request.

Then in phase (3), with the obtained workload on the replica space, we make the replica placement decision through extending the hypergraph formulation. The vertices in the hypergraph become the union of the replica set and node set. In the edge set, the data-node edges are substituted by the replica-node edges. The weights of edges are set according to

$$w_e = \begin{cases} \lambda R_{p'}, & \text{for the request pattern edge } e_{p'} \\ \alpha R_{zy}, & \text{for the replica-node edge } e_{zy} \end{cases} . \quad (17)$$

Based on the above formulation, we can still apply the hypergraph partitioning algorithm as in the scheme without replica. The time complexity of the n-way hypergraph partitioning algorithm is $O\big((|V| + |E|) \log n\big)$, so the time complexity of our scheme is no more than $O\big((|P| + km)n \log n\big)$.

After applying the hypergraph partitioning, we obtain the replica placement result, which in fact is the input of making routing decision in the next round. After rounds of iterations of the routing decision and placement decision, the obtained performance tends to keep stable, and we would stop the iteration after the improvement is less than a threshold γ. Finally, the placement and routing decision in the last round are informed to the nodes in the system. With the deterministic routing decision $\mathcal{M}$, we can obtain a hash mapping function for each node, whose input is a request pattern and output is the routing destination of each item in the pattern. Such a function ensures the dispatch of any requests can be made in a very short time which is important for a practical system.

Algorithm 1 *Associated Data Placement with Replicas*

$\mathcal{D} \leftarrow Phase1(\mathcal{I})$ ▷ Initial replica placement
$C \leftarrow 0$
repeat
  $\mathcal{M} \leftarrow Phase2(\mathcal{D})$ ▷ Routing decision
  $R' \leftarrow FuncR(\mathcal{M}, \mathcal{I})$ ▷ Obtain workload to replicas
  $\mathcal{I}' \leftarrow \{R', P\}$ ▷ Inputs on the replica space
  $\mathcal{D} \leftarrow Phase3(\mathcal{I}')$ ▷ Hypergraph partitioning
  $C_{last} \leftarrow C$
  $C \leftarrow C(\mathcal{M}, \mathcal{D})$
until $C - C_{last} < \gamma$
return $\mathcal{M}, \mathcal{D}$

C. Replica Migration

When the workload changes, it is not necessary to adjust the system through completely overriding the existing data placement. Therefore we formulate the replica migration problem and propose the incremental adjustment method. In a practical system, we are given a budget of changing the locations of data replicas. So in the formulated *replica migration problem*, we try to minimize the objective (10) through changing the locations of no more than K replicas, based on the exiting placement solution $\mathcal{D}$, under the changed workload.

Consider that there are N replicas in total, we can set $N - K$ of them as fixed-location vertices in the hypergraph formulation, just as we pre-assigned each node to a set previously, and then continue the iterative process above to determine the locations of the remaining K replicas, who are chosen as candidates of replica migration. Then the problem is how to choose the K candidates. Our idea is to select those replicas whose location change can potentially provide a higher gain to the system performance. The strategy in the choosing the K candidates is given as follows. For each replica z of data item x that previously located at node j, we define the gain of its migration as

$$g_z = \max_{j' \in Y/j} \left\{ g^A_{z,j \to j'} + \alpha \cdot g^L_{z,j \to j'} \right\} , \quad (18)$$

which is the maximum benefit after moving it to any other location j'. In (18), $g^A_{z,j \to j'}$ and $g^L_{z,j \to j'}$ correspond to the two metrics in the modeling. For the former, we calculate the benefit of moving replica z from j to j', from the perspective of co-locating associated data. Note that the location change of z would affect related request patterns. Representing the benefit in terms of the number of nodes used to fulfill a request

since the moving by the function $\Delta_{py}^{j\to j'}$, we define

$$g^A_{z,j\to j'} = -\lambda \sum_{p\in P} \mathbf{1}(x \in p) \sum_{y\in Y} R_{py}\Delta_{py}^{j\to j'} \ . \quad (19)$$

For the latter, we define it as the maximum relaying traffic decrease at the new location j' deducting the traffic increase at the old location j, such as

$$g^L_{z,j\to j'} = (1 - \mathbf{1}(x \in D_{j'}))R_{xj'} - R_{zj} \ . \quad (20)$$

After sorting the migration gain g_z of all the replicas in the system, the replica with a higher gain is preferred to be moved away from the original location. Meanwhile, we realize that more than K candidates can be chosen at the beginning, but it may result in more than K replicas with location changed after the hypergraph partitioning. So a binary search is applied to find the optimal candidate number that yields a solution with at most K replicas being migrated finally.

Algorithm 2 *Replica Migration*

1: $Z_c \leftarrow \emptyset; G \leftarrow \emptyset; N_t \leftarrow N;\ N_b \leftarrow K$
2: **for** $z \in Z$ **do** ▷ Obtain migration gain of replicas
3: $G \leftarrow \{G, g_z\}$
4: **end for**
5: **while** $N_t \neq N_b$ **do** ▷ Binary search in $[K, N]$
6: $N_c \leftarrow (N_t + N_b)/2$
7: $Z_c \leftarrow$ Choose N_c replicas as candidates based on G
8: $\mathcal{M}', \mathcal{D}' \leftarrow$ Apply Alg. 1 with $\{Z - Z_c\}$ being fixed
9: $N_m \leftarrow$ Location changes between $\mathcal{D}'$ and original $\mathcal{D}$
10: If $N_m > K$, $N_t \leftarrow N_c$; otherwise, $N_b \leftarrow N_c$
11: **end while**
12: **return** the best $\mathcal{M}', \mathcal{D}'$ ensuring $N_m \leq K$

The pseudo-code of replica migration is shown in Alg. 2. We obtain the set G containing the migration gain of all replicas through line 2–4. The iteration of binary search is executed in line 5–11. In each iteration, N_c is the number of selected candidates before hypergraph partitioning and the replicas with a higher gain in the sorted G are selected as candidates, denoted by Z_c. The replicas not included in Z_c are marked as fixed in applying the hypergraph partitioning. N_m is the number of replicas actually migrated, obtained by comparing the new placement and the old one. The binary search ensures that N_m in the solution can get close to K.

V. Performance Evaluation

A. Experiment Settings

We have implemented the proposed scheme and conducted several studies. In the experiments, we consider there are $n = 10$ datacenters (or nodes), which are geo-distributed. The total number of data items is $m = 10,000$. The number of request patterns in the system is $|P| = 40,000$. In the simulation, for each request pattern, the data items involved are generated first, by randomly selecting at most $d = 40$ items from the set of m items. Previous studies, e.g., [20], have shown that in a broad range of network services, the request rates to different contents follow the Zipf distribution [21]. So the request rate for each pattern is generated following the Zipf distribution with index $H = 0.8$. After that, a node is uniformly generated, set as the source location of that request pattern. The default values of other parameters are $\lambda = 5.0$, $\alpha = 1.0$ and $\epsilon = 0.1$, considering the general small-size data item storing scenario.

Besides the proposed scheme `ADP`, the other schemes implemented and being compared with include: `Hash`, `Closest`, `Multiget` and their variants to fit the case in which replicas are allowed. `Hash` places the data items to the nodes based on the hashing results, representing the state-of-the-art implementation. `Closest` places a data item to the node that has the largest request rate to that particular item. `Multiget` optimizes to place the associated data in the same location extremely, but does not consider the localized data serving.

The metrics we used in the evaluation include:

Co-location: We calculate the average number of data serving nodes involved to fulfill each request, which is weighted by the request rate, and then show the result of dividing it by $n = 10$. It is directly related to the metric C^A in the modeling.

Localization: We also show the percentage of the data items that *is not* served locally among all the requests. It is directly related the metric C^L in the modeling.

Objective: Our optimization objective is given in (10), which is the weighted sum of C^A and C^L. The objective value shown in the figures is normalized with regard to the obtained objective value of `Hash` under the same settings by default.

Balance: We also evaluate the balance of each scheme. Denoting the number of replicas in nodes by $h_1, ..., h_n$, it is calculated by $\max_{i=1}^{n} |h_i - h_a|/h_a$, where $h_a = \sum_{i=1}^{n} h_i/n$.

B. Experiment Results

1) Scenario without Replicas: In the first experiment, we study the performance of `ADP` and compared schemes in a no-replica scenario. Fig. 4 compares the metrics obtained based on their placement results. We can observe that `ADP` performs the best on the objective and the gap between `Hash` and other three schemes is apparent. In terms of co-location, `Multiget` achieves the best performance, because it focuses on the associated relationship among data items in the placement. `Closest` achieves the best performance on localization, since in its design, each data item is always placed to the node with the largest request rate to it. However, `Multiget` and `Closest` only consider one of the two aspects. Comparatively, by optimizing both aspects in a unified way, `ADP` achieves the lowest objective value even if its performance is not the best on either individual metric. Here the shown improvement through `ADP` is not significant because of the conflict between the two metrics and the limited optimization space without replicas. We also compare the balance result of different schemes. Although it is shown that the balance of `ADP` degrades when compared with `Hash`, the balance is still in the acceptable range. According to the results of `ADP`, the maximum and minimum number of items in each of the 10 nodes is $1,083$ and 937 respectively, while the average number of items in each node is $1,000$.

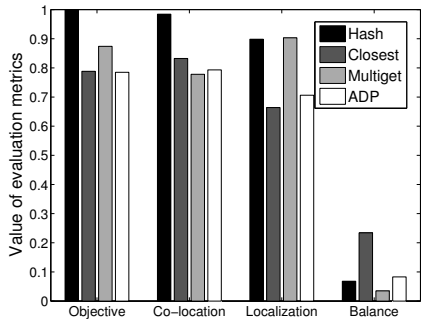

Fig. 4. Performance without replica

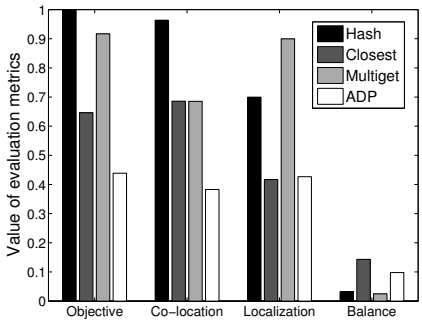

Fig. 5. Performance with 3 replicas

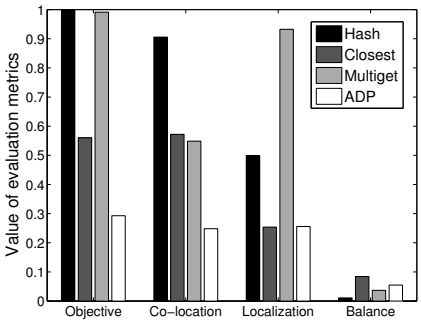

Fig. 6. Performance with 5 replicas

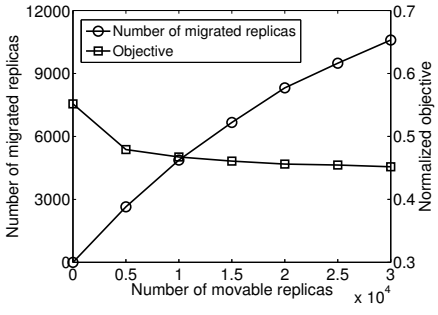

Fig. 7. Replica migration validation

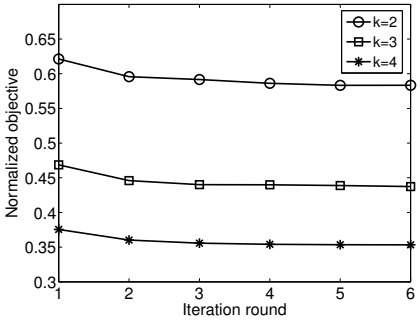

Fig. 8. Iterative process in ADP

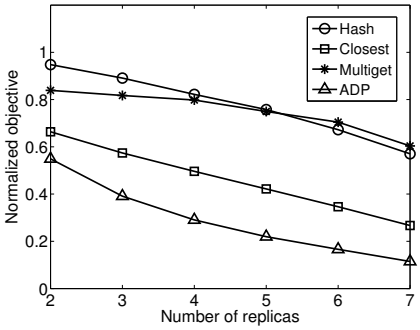

Fig. 9. Effect of replica number

2) Scenario with Replicas: We further evaluate the performance of different schemes under the scenario with replicas. When the replica number is 3, the objective value of ADP outperforms the others by more than 20%, as shown in Fig. 5. It achieves the near-optimal performance on the localized data serving and gives the best performance on the co-location of associated data, by exploiting the existence of replicas. When the replica number is increased to 5, the performance of ADP is even better, as shown in Fig. 6, such as the objective improvement is more than 90% when compared with other schemes. As for the balance, the node difference on the number of stored replicas is still not large in all the schemes. In fact, when compared with Closest, which more relies on the uniform workload from different locations to achieve the balance, in ADP, we can initiatively control the result of balance through adjusting the balance parameter of the hypergraph partitioning algorithm, which will be shown later.

3) Replica Migration: We simulate the workload change by randomly shuffling the request rates of all request patterns. The replica migration method is validated as follows: arbitrarily set the number of movable replicas before applying the placement algorithm, denoted by N_c; then after applying the algorithm, obtain the number of replicas changing location, denoted by N_m, and the objective value. We set the replica number to 3, resulting in $30,000$ replicas in total. Then we change N_c from 0 to $30,000$ in the experiment, where 0 indicates the performance without any migration and $30,000$ indicates the performance of a complete overriding of the exiting placement. As shown in Fig. 7, the increase of N_c enlarges the scope of movable replicas, which yields the decrease of the objective value. When $N_c = 5,000$, more than 50% of the replicas are actually migrated and the objective value shows the largest decrease, which validate the effectiveness of our candidate selection. The monotonically increase of N_m with the increase of N_c validates that the binary search is applicable here.

4) Others: To further investigate the performance of ADP and validate its effectiveness in design, we take extra experiments. Fig. 8 plots the change of the objective value in rounds of iterations in Alg. 1. We expect that the solution is improved through repeatedly making the routing and replica placement decision. As shown in the figure, after about 3–4 rounds, the objective value tends to keep stable. Besides, the method is effective to different numbers of replicas.

To show the effect of changing the number of replicas, we plot Fig. 9. The shown values are all normalized towards the objective value of Hash without replica, to show the trend of performance change under the same scheme. When the number of replicas is increased, the system performance is expected to be improved, such as more requests can be satisfied locally or through fewer nodes. In the figure, ADP shows a better performance in all the cases tested. In practice, the number of replicas cannot be too large as it increases the storage cost of the system and the difficulty of consistency maintenance.

The parameter α in (10) can tradeoff the importance of the data co-location and localized data serving. We change its value, and show the corresponding objective value in Fig. 10. When α is larger, the improvement of ADP over Closest becomes less. It is because that the weight of the localized data serving metric is increased in the objective function and the latter can give the optimal performance on that metric. Meanwhile, the performance of Multiget gets worse because of not paying attention to that metric.

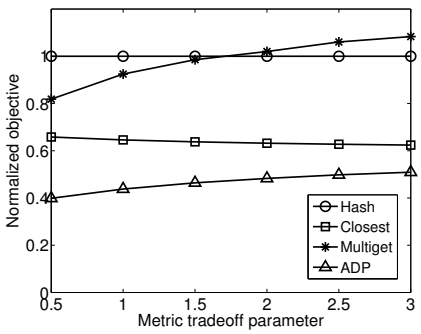

Fig. 10. Effect of metric tradeoff parameter α

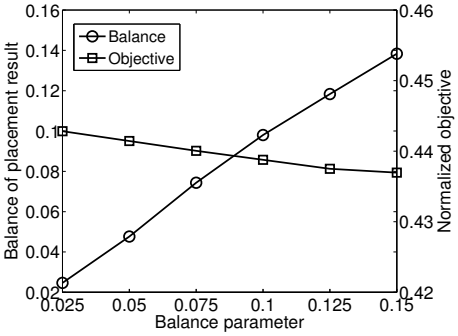

Fig. 11. Effect of balance parameter ϵ

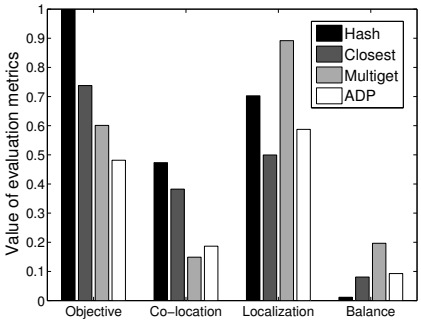

Fig. 12. Performance with OSN trace

The hypergraph partitioning algorithm can take the balance parameter ϵ as input, which allows us to flexibly control the balance in ADP. We plot Fig. 11 to compare the results with different ϵ as the input of the algorithm. The increase of ϵ makes the nodes less balanced but results in a lower objective value. Besides, it is shown that the resultant balance metric is always lower than the input ϵ. So in ADP, given the acceptable level of balance, we can tune the parameter ϵ to improve the system performance while satisfying the balance constraint.

We further make an experiment with an OSN dataset [22]. We extract the friend relationships from the dataset and simulate the scenario that a user polls the updates of all its friends when using the news feed updating function. We consider each user as a data item, which results in 45,092 data items and 63,102 request patterns in the experiment. The request rates are randomly generated following the Zipf distribution and the source location of each request pattern is also random. On weighted average, each request pattern contains 11.22 items. The number of replicas is set to $k = 2$. The comparison results are shown in Fig. 12, which also validate the better performance of our proposed scheme. Here the running of ADP lasts about 292.8 seconds on the Xeon E5645 CPU, and we will further improve the running efficiency in the future.

VI. Conclusions

We studied the balanced data placement problem for geographically distributed datacenters, with joint considerations of the localized data serving and the co-location of associated data. By formulating the scenario without replicas using a hypergraph, we presented the framework to obtain the optimized solution. We further addressed the placement problem under the replica scenario, by introducing the iterative process of routing and replica placement. Besides, the method that migrates replicas based on the existing placement was proposed to deal with the workload change. Finally, we evaluated the proposed scheme through extensive experiments.

Acknowledgment

This work is supported in part by NSERC, CFI and BCKDF.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM computer communication review*, vol. 39, no. 1, pp. 68–73, 2008.

[3] M. Rabinovich and O. Spatscheck, *Web Caching and Replication*. Addison-Wesley, 2002.

[4] "Memcached Multiget Hole," http://highscalability.com/blog/2009/10/26/facebooks-memcached-multiget-hole-more-machines-more-capacit.html.

[5] "HDFS Architecture Guide," http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[6] "About Replication in Cassandra," http://www.datastax.com/docs/1.0/cluster_architecture/replication.

[7] Y. Rochman, H. Levy, and E. Brosh, "Resource placement and assignment in distributed network topologies," in *Proc. of IEEE INFOCOM*, 2013.

[8] H. Xu and B. Li, "Joint request mapping and response routing for geo-distributed cloud services," in *Proc. of IEEE INFOCOM*, 2013.

[9] S. Raindel and Y. Birk, "Replicate and bundle (RnB)–a mechanism for relieving bottlenecks in data centers," in *Proc. of IEEE IPDPS*, 2013.

[10] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab *et al.*, "Scaling memcache at Facebook," in *Proc. of USENIX NSDI*, 2013, pp. 385–398.

[11] A. Quamar, K. A. Kumar, and A. Deshpande, "Sword: scalable workload-aware data placement for transactional workloads," in *Proc. of EDBT*, 2013, pp. 430–441.

[12] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez, "The little engine (s) that could: scaling online social networks," in *Proc. of ACM SIGCOMM*, 2010.

[13] G. Liu, H. Shen, and H. Chandler, "Selective data replication for online social networks with distributed datacenters," in *Proc. of IEEE ICNP*, 2013.

[14] L. Jiao, J. Li, W. Du, and X. Fu, "Multi-objective data placement for multi-cloud socially aware services," in *Proc. of IEEE INFOCOM*, 2014.

[15] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated data placement for geo-distributed cloud services," in *Proc. of USENIX NSDI*, 2010.

[16] X. Tang and J. Xu, "On replica placement for QoS-aware content distribution," in *Proc. of IEEE INFOCOM*, 2004.

[17] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "Map task scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality," in *Proc. of IEEE INFOCOM*, 2013.

[18] J. S. Hunter, "The exponentially weighted moving average." *Journal of Quality Technology*, vol. 18, no. 4, pp. 203–210, 1986.

[19] U. V. Catalyurek and C. Aykanat, "PaToH: Partitioning tool for hypergraphs," *http://bmi.osu.edu/umit/PaToH/manual.pdf*.

[20] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "YouTube traffic characterization: a view from the edge," in *Proc. of ACM IMC*, 2007, pp. 15–28.

[21] L. A. Adamic and B. A. Huberman, "Zipf's law and the internet," *Glottometrics*, vol. 3, no. 1, pp. 143–150, 2002.

[22] J. Kunegis, "Konect - the koblenz network collection," in *Proc. of Int. Web Observatory Workshop*, 2013. http://konect.uni-koblenz.de/networks/facebook-wosn-links.