Scalable Data Placement of Data-intensive Services in Geo-distributed Clouds

Ankita Atrey, Gregory Van Seghbroeck, Bruno Volckaert and Filip De Turck
IDLAB-imec, Technologie Park, UGent, Gent, Belgium

Keywords: Data Placement, Geo-distributed Clouds, Location-based Services, Online Social Networks, Scalability, Spectral Clustering, Hypergraphs, Approximation.

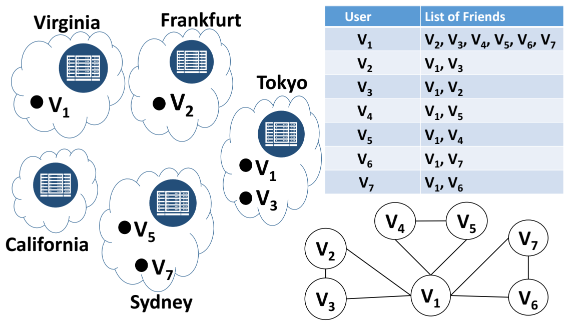
Abstract: The advent of big data analytics and cloud computing technologies has resulted in wide-spread research in finding solutions to the data placement problem, which aims at properly placing the data items into distributed datacenters. Although traditional schemes of uniformly partitioning the data into distributed nodes is the de-facto standard for many popular distributed data stores like HDFS or Cassandra, these methods may cause network congestion for data-intensive services, thereby affecting the system throughput. This is because as opposed to MapReduce style workloads, data-intensive services require access to multiple datasets within each transaction. In this paper, we propose a scalable method for performing data placement of data-intensive services into geographically distributed clouds. The proposed algorithm partitions a set of data-items into geo-distributed clouds using *spectral clustering on hypergraphs*. Additionally, our spectral clustering algorithm leverages randomized techniques for obtaining low-rank approximations of the hypergraph matrix, thereby facilitating superior scalability for computation of the spectra of the hypergraph laplacian. Experiments on a real-world trace-based online social network dataset show that the proposed algorithm is effective, efficient, and scalable. Empirically, it is comparable or even better (in certain scenarios) in efficacy on the evaluated metrics, while being up to 10 times faster in running time when compared to state-of-the-art techniques.

1 MOTIVATION

Since the advent of the Internet, the scale at which data is being generated and processed is increasing at an exponential rate (ins, 2017). Today, we live in a world that is *data-rich* or what is also referred to as the *Information age*. For instance, the amount of data managed by Internet giants like Google and Facebook is of the order of *thousands of petabytes* (sca, 2015). The advancements in big data and cloud computing technologies have definitely enriched the field of scalable data management with state-of-the-art distributed data processing systems like Hadoop and more recently Apache Spark. The main idea employed in these systems is to uniformly distribute data across servers and perform parallel computations on the so-constructed small subsets of data on each server independently. While uniform data partitioning schemes using hashing work well for MapReduce style workloads that can be easily parallelized, they are not suitable for data-intensive workloads that require access to multiple datasets within each transaction (Yu and Pan, 2015; Golab et al., 2014). In these scenarios uniform partitioning may result in a huge volume of data migrations thereby leading to network congestion and eventually reduced system throughput, especially in the case of geographically distributed datacenters where inter-datacenter communication latencies and costs are high. Having said that, there is a need for specialized data placement strategies capable of addressing the previously discussed deficiencies for data-intensive services.

Data-intensive services are becoming increasingly common in a plethora of real-world scenarios, namely – *online social networks (OSNs)*, *content distribution networks (CDNs)* etc. Additionally, ubiquity of the cloud and increased reliance of people across the globe on online services like OSNs and CDNs, requires data to be stored in datacenters that are geographically distributed. The workload generated by these services present two niche challenges that are non-existent in MapReduce style workloads. (1) In the case of CDN services like *YouTube* the hosted content is stored in datacenters situated around the world. It is highly likely that the set of contents retrieved by a user query may be stored in different datacen-

Atrey, A., Seghbroeck, G., Volckaert, B. and Turck, F.
Scalable Data Placement of Data-intensive Services in Geo-distributed Clouds.
In *Proceedings of the 8th International Conference on Cloud Computing and Services Science (CLOSER 2018)*, pages 497-508
ISBN: 978-989-758-295-0



User	List of Friends
v_1	$v_2, v_3, v_4, v_5, v_6, v_7$
v_2	v_1, v_3
v_3	v_1, v_2
v_4	v_1, v_5
v_5	v_1, v_4
v_6	v_1, v_7
v_7	v_1, v_6

Figure 1: Use Case: A Location-based OSN service.

ters across the globe. Similar is the case for OSNs as well. (2) Moving ahead, since the users of OSN services like *Facebook* may register *check-ins* at various locations across the world, not only the data-items but also the source locations of data requests are geographically distributed.

Motivated by the above discussions, the problem of data placement for data-intensive services in datacenters that are distributed geographically across the world is the topic of research detailed in this paper. The use case under investigation is a location-based OSN service as portrayed in Fig 1. A sample Facebook social network is represented using a graph where each vertex corresponds to a user and undirected edges between two vertices represent friendship. In this network, users $\{v_1, v_5\}$ are friends of the user v_4. Similarly $\{v_6, v_1\}$ are friends of v_7. Additionally, the list of all the friends of every user is also portrayed in the table presented right above the social network. Each user can register a check-in, which is denoted by her user-id assigned to a datacenter nearest to her check-in location. In Fig. 1, the user v_1 has registered two check-ins, at datacenters located in Virginia and Tokyo respectively, while user v_2 has checked-in at Frankfurt. Moreover, each user check-in requires data-items corresponding to all her friends, which constitutes the data request pattern triggered by this check-in. For example, a check-in by user v_5 in Sydney would trigger a data request requiring the data-items corresponding to her friends $\{v_1, v_4\}$ to be transferred to that location.

While data placement has been studied extensively (Golab et al., 2014; Yuan et al., 2010; Ebrahimi et al., 2015; Jiao et al., 2014), literature on geo-distributed data-intensive services is relatively scarce (Yu and Pan, 2017). Any successful solution to this problem should provide two capabilities, namely – capturing and improving (1) data-item – data-item associations; and (2) data-item – datacenter associations. State-of-the-art methods proposed in (Nishtala et al., 2013) and (Agarwal et al., 2010) are capable of improving data-item – data-item, and data-item – datacenter associations respectively, however, these methods cannot jointly handle both aspects. To jointly incorporate both aspects, (Yu and Pan, 2015; Yu and Pan, 2017) recently proposed a multi-objective data placement algorithm using *hypergraphs*. Hypergraphs offer a powerful representation by presenting a natural way of capturing multi-way relationships. Similar to graph partitioning however, hypergraph partitioning is NP-Hard. To this end, the authors use heuristic partitioning algorithms available in a publicly available tool – PaToH (Catalyurek, 2011), to efficiently partition large hypergraphs. However, as shown in Sec. 6, PaToH lacks scalability under certain scenarios while also suffering in terms of efficacy. To bridge this gap, in this paper, we propose a novel scalable method for data placement through *Spectral Clustering on Hypergraphs*.

Owing to their strong mathematical properties spectral methods have been shown to be promising in a plethora of areas of machine learning with seminal research by (Shi and Malik, 2000; Spielmat, 1996; Meila and Shi, 2001; Ng et al., 2001). Despite the advantages offered by spectral methods: superior efficacy, strong mathematical properties etc., they are not usually efficient and scalable on large scale data. We mitigate the issue of lack of scalability by proposing an approximate spectral clustering algorithm, thereby bringing the power of spectral methods to perform effective hypergraph clustering. To summarize, the key contributions of this work are as follows.

- We study the data placement problem in a challenging and close to real-world setting of *data-intensive services in geo-distributed datacenters* (Sec. 3), where traditional methods of hash based partitioning that are prominent in systems like Hadoop and Spark do not perform well.
- We propose a novel scalable algorithm to solve the data placement problem for geo-distributed data-intensive services through *Spectral Clustering on Hypergraphs* (Sec. 4).
- Through extensive experiments on a real-world trace-based social network dataset (Secs. 5 and 6), we show that the proposed spectral clustering algorithm is scalable, and provides a *speed-up* of up to *10 times* over the state-of-the-art hypergraph partitioning method while portraying similar (or in some cases even better) efficacy on several evaluation metrics.

2 RELATED WORK

The data placement problem has been studied extensively in the literature spanning a wide-variety of research areas, both from the perspective of execution environments: ranging from distributed systems (Chervenak et al., 2007; Golab et al., 2014) to cloud computing environments (Yuan et al., 2010; Li et al., 2017; Ebrahimi et al., 2015; Liu and Datta, 2011); application areas: online social network services (Jiao et al., 2014; Han et al., 2017), location aware data placement for geo-distributed cloud services (Yu and Pan, 2017; Zhang et al., 2016; Yu and Pan, 2015; Yu and Pan, 2016; Agarwal et al., 2010), and many more. Here, we provide an overview the existing works that overlap with our problem.

Research on data placement in geo-distributed clouds has increasingly gained popularity over the years. Owing to multiple niche challenges as discussed in Sec. 1, specialized solutions have been devised to solve this problem. The biggest challenge for data placement algorithms in such scenarios is that data migrations from one location to another is significantly more expensive and problematic when compared to other real-world scenarios like grids, clusters, or private clouds where distance between data-centers is not significant.

(Agarwal et al., 2010) proposed a system: Volley, to perform automatic data placement in geographically distributed datacenters. The proposed system possesses the capability to capture data-item – datacenter associations, however, it lacked the capability for handling data-item – data-item associations. (Jiao et al., 2014) formulates a multi-objective social network aware optimization problem that performs data placement by building a model framework, which takes multiple objectives into consideration. (Han et al., 2017) introduce an adaptive data placement algorithm for social network services in a multicloud environment, which adapts to the changing data traffic for performing intelligent data migration decisions. (Rochman et al., 2013) have focused on placing data in a distributed environment to ensure that a large fraction of region specific requests are served at a lower cost. In (Huguenin et al., 2012), a user generated content (UGC) dataset (with more than 650000 YouTube videos) is used to show the correlation between the content locality and geographic locality. (Zhang et al., 2016) propose an integer programming based data placement algorithm capable of minimizing the data communication cost while honoring the storage capacity of geo-distributed datacenters as well.

Researchers have also focused on the design of specialized replication strategies for geo-distributed services. (Kayaaslan et al., 2013) have proposed a document replication framework to deal with the scalability issues. In this replication framework, documents are replicated on datacenters based on region specific user interests. (Shankaranarayanan et al., 2014) propose replication strategies for a class of cloud storage systems denoted as quorum-based systems (viz. Cassandra, Dynamo) capable of solving the data placement problem cognizant of various location-aware metrics like location of geo-distributed datacenters, inter-datacenter communication costs etc.

Table 1: Summary of notations used.

Item	Definition
V	The set of users in the social network $V = \{v_1, v_2, \dots, v_n\}$.
E	The set of edges in the social network $\forall e = (v_x, v_y) \in E$.
$\mathbf{Adj}(v)$	The set of friends of the user $v \mid v \in V$.
$\mathcal{D}$	The set of data-items $\mathcal{D} = \{d_{v_1}, d_{v_2}, \dots, d_{v_n}\}$.
$\mathcal{L}$	The set of datacenters and their locations $\mathcal{L} = \{L_1, L_2, \dots, L_l\}$.
$\mathcal{R}$	The set of request patterns $\mathcal{R} = \{R_1, R_2, \dots, R_r\}$.
$\mathcal{C}$	The set of user check-ins $\mathcal{C} = \{C_k = (R_i, L_j) \mid \exists R_i \in \mathcal{R}, L_j \in \mathcal{L}\}$.
Π	The hypergraph incidence matrix.
$\mathcal{W}_\Pi$	The hyperedge weight matrix.
Φ	The desired datacenter storage distribution.
$\mathcal{P}(\mathcal{D})$	A partition on the set of data-items $\mathcal{D}$.
$\Gamma(L_j)$	Cost (per unit) of outgoing traffic from datacenter L_j.
$\kappa(L_j, L_{j'})$	Inter datacenter latency (directed) between L_j and $L_{j'}$.
$\mathcal{S}(L_j)$	Storage cost (per unit) of datacenter L_j.
$\mathcal{N}(R_i)$	Average number of datacenters accessed by request R_i.

(Golab et al., 2014) prove the reduction of data placement problem to the well known graph partitioning problem and propose an integer linear programming solution. The authors also propose two scalable heuristics to reduce the data communication cost for data-intensive scientific workflows and join-intensive queries in distributed systems. In (Quamar et al., 2013), the authors study the problem of OLTP workloads in cloud computing environments, and propose a scalable workload aware data partitioning and data placement approach called SWORD to reduce the partitioning overhead. SWORD is proposed as a two phase approach: in the first phase a workload is modeled as a hypergraph which is further compressed by using hash partitioning, and later the compressed hypergraph is partitioned to get the placement output. Hypergraph based partitioning solutions (Catalyurek et al., 2007) have also been used in grid and distributed computing environments previously.

Recently, (Yu and Pan, 2015; Yu and Pan, 2016; Yu and Pan, 2017) propose data placement strategies using hypergraph modeling and publicly available partitioning heuristics (Catalyurek, 2011) for data-intensive services, which constitutes the current state-of-the-art for location-aware data placement in geo-distributed clouds. The research presented in this paper proposes a novel hypergraph partitioning scheme using spectral clustering and enjoys strong mathematical properties and improved efficacy when compared to the competing techniques. Moreover, it enjoys superior efficiency and scalability by employing low-rank matrix approximations while retaining the same efficacy as portrayed by the state-of-the-art.

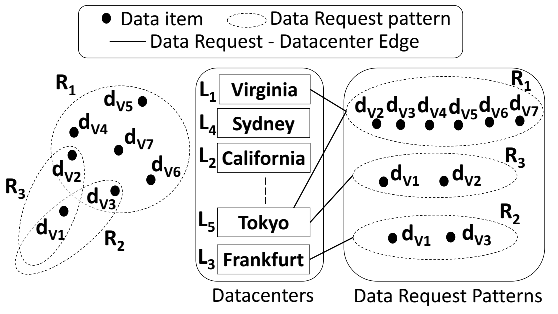


Figure 2: Mapping of different requests to geo-distributed datacenters.

3 PROBLEM STATEMENT

In this section, we formally define the data placement problem for data-intensive services in geo-distributed datacenters. Table 1 summarizes the notations used in the rest of the paper.

A location based online social network (Fig. 1) is represented using a graph $G(V,E)$, where V represents the set of users, and E represents the set of edges (denoting friendship) between any two users of the social network. The set $\mathcal{D}$ contains n data-items corresponding to each user $v \in V$ of the social network. Each user $v \in V$ can register a check-in, where a check-in is a tuple $C_k = (R_i, L_j) \in \mathcal{C}$ composed of a data request pattern $R_i \in \mathcal{R}$ and a datacenter location $L_j \in \mathcal{L}$. Each data request pattern R_i is comprised of a set of data-items $D \subseteq \mathcal{D}$ requested by a user $v \in V$. As discussed in Sec. 1, considering the location based social networks use-case, a data request pattern R_i belonging to a check-in C_k registered by a user v would comprise the data-items corresponding to all the friends of v, i.e. $R_i = \{d_u \mid u \in \mathbf{Adj}(v)\}$. Moving ahead, $L_j \in \mathcal{L}$ represents the location of a datacenter capable of serving user requests. The location of a check-in C_k is decided as the datacenter location L_j closest (in distance) to the actual physical location of the user check-in. In other words, the check-in C_k signifies a request for data-items contained in R_i triggered from the datacenter located at L_j.

Next, we discuss the basic concepts described above in the context of the location based online social networks use-case presented in Sec. 1. Fig. 2 builds upon the example portrayed in Fig. 1 to showcase the data request patterns corresponding to the check-ins registered by users v_1, v_2, and v_3. Let us denote the data request patterns as R_1, R_2, and R_3 respectively, where $R_1 = \{d_{v_2}, d_{v_3}, d_{v_4}, d_{v_5}, d_{v_6}, d_{v_7}\}$, $R_2 = \{d_{v_1}, d_{v_3}\}$, and $R_3 = \{d_{v_1}, d_{v_2}\}$. Let us also label the datacenter locations as $L_1 = Virginia, L_2 = California, L_3 = Frankfurt, L_4 = Sydney$, and $L_5 = Tokyo$. Recall that the user v_1 registered two check-ins: one in Virginia and the other in Tokyo. Similarly, v_2 registered a check-in in Frankfurt, while v_3 checked-in in Tokyo. Thus, in total there are four check-ins: two (C_1 and C_2) for the user v_1, and one each (C_3 and C_4 respectively) for users v_2 and v_3, where $C_1 = (R_1, L_1), C_2 = (R_1, L_5), C_3 = (R_2, L_3)$, and $C_4 = (R_3, L_5)$.

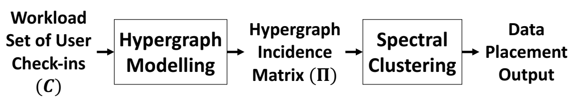
Having defined the basic concepts and their notations, we formally define the problem as:

Problem. *Given a set of n data-items $\mathcal{D}$, m user check-ins $C_k = (R_i, L_j) \in \mathcal{C}$, each comprising a data request pattern R_i being originated from a datacenter located at L_j, a set of l datacenters with locations in $\mathcal{L}$, with the per unit cost of outgoing traffic from each datacenter $\forall L_j \in \mathcal{L}, \Gamma(L_j)$, the per unit storage cost of each datacenter $\forall L_j \in \mathcal{L}, \mathcal{S}(L_j)$, the inter datacenter latency (directed) for each pair of datacenters $\forall L_j, L_{j'} \in \mathcal{L}, \kappa(L_j, L_{j'})$, and the average number of datacenters accessed by the data-items requested in each request pattern R_i being $\mathcal{N}(R_i)$, perform data placement to minimize the optimization objective O, which is defined as the weighted average of $\Gamma(\cdot), \kappa(\cdot,\cdot), \mathcal{S}(\cdot)$, and $\mathcal{N}(\cdot)$.*

4 SPECTRAL CLUSTERING ON HYPERGRAPHS

To effectively solve the data placement problem, we propose a technique as outlined in Algorithm 1. Given the set of data items $\mathcal{D}$, and the set of user check-ins $\mathcal{C}$ comprising the set of data request patterns $\mathcal{R}$ and their locations $\mathcal{L}$, we first construct a hypergraph. As discussed in Sec. 1, hypergraphs provide the capability to capture *multi-way* relationships, thereby facilitating modeling of data-item – data-item and data-item – datacenter location associations. With the hypergraph incidence matrix Π constructed, next, we partition the set of data-items $\mathcal{D}$ into l parts corresponding to the $\mathcal{L}$ datacenters according to the desired storage distribution Φ, using the proposed scalable spectral clustering algorithm. Fig. 3 portrays the overall scheme of our method.

We next explain each of the fundamental steps in detail.



Workload Set of User Check-ins (C)
Hypergraph Modelling
Hypergraph Incidence Matrix (Π)
Spectral Clustering
Data Placement Output

Figure 3: Overview of our Approach.

4.1 Hypergraph Modeling

Continuing our discussion in Sec. 1, hypergraphs provide the ideal representation to solve the data placement problem in geo-distributed datacenters owing to their capability of capturing multi-way relationships. A hypergraph $H(V_H, E_H)$ is a more sophisticated graph construct and a generalization over a graph $G(V,E)$, where (hyper)edges are capable of capturing relationships between several vertices as opposed to just a pair of vertices in graphs. This ability of hyperedges to capture higher order relationships between data points facilitates the hypergraph model to manage both data-item – data-item and data-item – datacenter locations associations.

Algorithm 1: Data Placement Algorithm.

```
Input: $\mathcal{D}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{L}$, $G(V,E)$, $\Phi$
Output: Partitioning of the set of data-items $\mathcal{P}(\mathcal{D})$ into $l$
    datacenters
1: $(\Pi, \mathcal{W}_\Pi) \leftarrow$ ConstructHypergraph$(\mathcal{D}, \mathcal{C}, G(V,E))$
2: $\mathcal{P}(\mathcal{D}) \leftarrow$ SpectralClustering$(\Pi, \mathcal{W}_\Pi, l, \Phi)$
3: return $\mathcal{P}(\mathcal{D})$
```

In the context of our problem statement, every user check-in $C_k \in \mathcal{C}$ consists of a datacenter location $L_j \in \mathcal{L}$ and a data request pattern $R_i \in \mathcal{R}$. Note that for a check-in by a user v, R_i is a set of data-items corresponding to all the friends of v, i.e. $\mathbf{Adj}(v)$, as portrayed by the social network $G(V,E)$. Since a data request pattern involves data-items corresponding to multiple vertices of $G(V,E)$, hyperedges provide a better way to model data-item – data-item associations by linking/connecting multiple data-items via the same hyperedge. Additionally, hyperedges also facilitate modeling of data-item – datacenter location associations by connecting a data-item with a datacenter location when a data-item d_{V_i} is requested from a datacenter location L_j.

The hypergraph vertex set V_H comprises of all the data-items $\mathcal{D}$ and the datacenter locations $\mathcal{L}$. Thus, the number of vertices in the hypergraph are $|V_H| = n' = n + l$. Formally,

$$V_H = \mathcal{D} \cup \mathcal{L} \qquad (1)$$

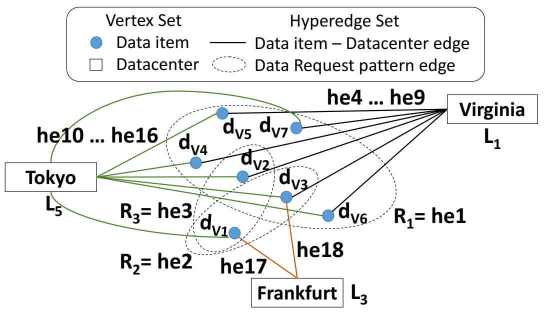
Let $\mathcal{R}_\mathcal{L} = \{\forall d \in R_i, \forall L_j \in \mathcal{L} | \exists C_k = (R_i, L_j) \in \mathcal{C}, R_{dj}\}$ denote the set of edges connecting data-items with datacenter locations corresponding to all the data request patterns $R_i \in \mathcal{R}$ triggered by user check-ins $C_k \in \mathcal{C}$ at datacenter locations $L_j \in \mathcal{L}$. The hypergraph edge set E_H consists of hyperedges corresponding to all the data request patterns $\mathcal{R}$ and all the data-item – datacenter location edges $\mathcal{R}_\mathcal{L}$. Thus, the number of hyperedges in the hypergraph are $|E_H| = m' = r + nl$. Formally,

$$E_H = \mathcal{R} \cup \mathcal{R}_\mathcal{L} \qquad (2)$$

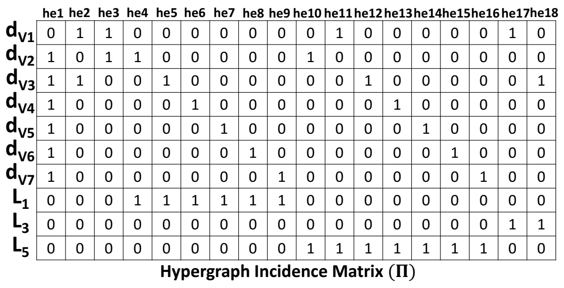
Fig. 4a portrays the hypergraph representation of the data-items and request patterns as presented in Fig. 2. The data-items $\{d_{v1}, \ldots, d_{v7}\}$ and the datacenter locations $\{L_1, L_3, L_5\}$ constitute the hypergraph vertex set. The hyperedges corresponding to the data request patterns $\{R_1, R_2, R_3\}$ are labeled as $he1, he2, he3$ respectively and are denoted using a dashed ellipse, while the hyperedges connecting each data-item – datacenter location pair are labeled as $he4, \ldots, he17$. Since $v_2, v_3, v_4, v_5, v_6, v_7$ are friends of v_1, the data-items $d_{v2}, \ldots, d_{v7}$ belonging to the request pattern R_1 are connected by the hyperedge $he1$. Similarly, since v_7 is a friend of v_1, who registered two check-ins: one at Virginia (L_1) and the other at Tokyo (L_5), the hyperedge $he4 = (d_{v_7}, L_1)$ and $he10 = (d_{v_7}, L_5)$ represents the relationship between the data-item d_{v_7} and the datacenter locations where it was requested, namely – Virginia (L_1) and Tokyo (L_5). Fig. 4b portrays the incidence matrix Π corresponding to the hypergraph H presented in Fig. 4a. As can be seen, the rows of this matrix are the vertex set of the hypergraph, while the columns are the hyperedges. Moreover, if a vertex participates in a hyperedge then the row corresponding to it contains a 1 in the column corresponding to that hyperedge, and 0 otherwise. Since the hyperedge $he1$ (corresponding to the data request pattern R_1) connects the data-items $d_{v2}, \ldots, d_{v7}$ the entries in Π corresponding to them are filled with 1 while all other entries are 0.

4.1.1 Calculating Hyperedge Weights

Having constructed the hypergraph $H(V_H, E_H)$ and discussed its representation using a hypergraph incidence matrix Π, we next discuss ways to assign weights to hyperedges. There are two major types of hyperedges constructed in the representation discussed above: (1) Data request pattern hyperedges, and (2) Data-item – Datacenter hyperedges, and both of them capture different properties required by a data placement algorithm in geo-distributed datacenters. The weight of a data request pattern hyperedge $W_\mathcal{R}$ is set using the request rate of that pattern, which is defined as the number of times a data request pattern is triggered by a user check-in. The weight $W_\mathcal{R}$ facilitates prioritization of data-items that are usually requested together, to be placed together by the data placement algorithm, thereby helping optimize (minimize) $\mathcal{N}(R_i)$: the average number of datacenters accessed by a data request pattern R_i. On the other hand, the weights $(W^{\kappa}_{\mathcal{R}_\mathcal{L}}, W^{\mathcal{S}}_{\mathcal{R}_\mathcal{L}}, W^{\Gamma}_{\mathcal{R}_\mathcal{L}})$ corresponding to data-item – datacenter hyperedges ($\mathcal{R}_\mathcal{L}$) facilitate minimization of inter datacenter latency $\kappa(L_j, L_{j'})$, storage cost $\mathcal{S}(L_j)$, and cost of outgoing traffic $\Gamma(L_j)$ respec-



(a) Hypergraph Representation



	he1	he2	he3	he4	he5	he6	he7	he8	he9	he10	he11	he12	he13	he14	he15	he16	he17	he18
d_{V1}	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
d_{V2}	1	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
d_{V3}	1	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1
d_{V4}	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
d_{V5}	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
d_{V6}	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0
d_{V7}	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
L_1	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
L_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
L_5	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0

Hypergraph Incidence Matrix (Π)

(b) Hypergraph Incidence Matrix

Figure 4: Modeling the data request patterns triggered by user check-ins as a Hypergraph.

tively, by giving priority to placing data-items at datacenter locations from where they have been requested more frequently.

The resultant hyperedge weight matrix $\mathcal{W}_\Pi$ of Π is a diagonal matrix of size $m' \times m'$, which is defined as the weighted sum of $W_{\mathcal{R}}$, $W^{\kappa}_{\mathcal{R}_\mathcal{L}}$, $W^{\mathcal{S}}_{\mathcal{R}_\mathcal{L}}$, and $W^{\Gamma}_{\mathcal{R}_\mathcal{L}}$. Mathematically,

$$\mathcal{W}_\Pi = \mathbb{W} \cdot (W_{\mathcal{R}}, W^{\kappa}_{\mathcal{R}_\mathcal{L}}, W^{\mathcal{S}}_{\mathcal{R}_\mathcal{L}}, W^{\Gamma}_{\mathcal{R}_\mathcal{L}}). \tag{3}$$

where, $\mathbb{W}$ is the weight vector for deciding the priorities of the previously discussed hyperedge weighting strategies.

4.2 Spectral Clustering

Spectral methods have been shown to be promising in a plethora of machine learning research areas: image segmentation (Shi and Malik, 2000; Meila and Shi, 2001), data clustering (Ng et al., 2001), web search (Gibson et al., 1998), and information retrieval (Deerwester et al., 1990). The power of these algorithms is that they possess both sound mathematical properties and strong empirical prowess. They are named spectral algorithms as they use the information manifested within the spectra (both eigen-values and eigen-vectors) of a similarity/affinity matrix. More fundamentally, for a graph G represented using a similarity matrix containing node-node similarities, these methods use the spectra of the graph laplacian to understand the intrinsic data properties like structure, connectivity etc. Interestingly, the laplacian for hypergraph was derived in (Zhou et al., 2006), where it is shown to be analogous to the simple graph laplacian. This result facilitates application of spectral methods on hypergraphs.

Having discussed about the importance of spectral methods, we next describe the proposed algorithm for performing spectral clustering on hypergraphs. The first step in spectral clustering on hypergraphs is to construct the hypergraph laplacian matrix L_H. The output of the hypergraph construction step is a $n' \times m'$ dimensional hypergraph incidence matrix Π and a $m' \times m'$ dimensional diagonal hyperedge weight matrix $\mathcal{W}_\Pi$. As discussed previously, the hypergraph incidence matrix $\Pi = [he_1, he_2, \ldots, he_{m'}]$ possesses m' hyperedges, where each hyperedge $he_i = [he_{1,i}, he_{2,i}, \ldots, he_{n',i}]$ is a n'-dimensional binary vector. The entry $he_{j,i} = 1$ indicates that the j^{th} vertex in the hypergraph vertex set is participating in the i^{th} hyperedge, while $he_{j,i} = 0$ indicates otherwise. With this, the hypergraph laplacian L_H is defined as:

$$L_H = I - (D_v^{-1/2} \Pi \mathcal{W}_\Pi D_{he}^{-1} \Pi^T D_v^{-1/2}) \tag{4}$$

where, I is a $n' \times n'$ identity matrix;
D_v is a $n' \times n'$ diagonal vertex degree matrix defined as $D_v = diag(\sum \Pi)$;
D_{he} is a $m' \times m'$ diagonal hyperedge degree matrix defined as $D_{he} = diag(\sum \Pi^T)$;
$\mathcal{W}_\Pi$ is a $m' \times m'$ diagonal hyperedge weight matrix defined as $\mathcal{W}_\Pi = diag([W_1, W_2, \ldots, W'_m])$, and $W_1, \ldots, W'_m$ are calculated as described in Eq. 3. Thus, L_H becomes a $n' \times n'$ matrix.

The next step is to perform eigen-decomposition of the hypergraph laplacian matrix L_H, in order to identify its spectra: the eigen-values and eigen-vectors. The eigen-decomposition of L_H is written as:

$$L_H = U \Lambda V \tag{5}$$

where,
$U = [u_1, \ldots, u_{n'}]$ is a $n' \times n'$ matrix formed by the eigen-vectors of L_H;
$\Lambda = diag(\lambda_1, \ldots, \lambda_{n'})$ is a diagonal $n' \times n'$ matrix formed by the eigen-values of L_H and,
$V = U^T$ since L_H is a square symmetric matrix.

The eigen-vectors U and eigen-values Λ collectively define the spectra for the hypergraph laplacian L_H.

As discussed in Sec. 1, despite their strong theoretical and mathematical properties, spectral methods are usually not scalable. This is mainly due to the complex operation of performing a full eigen decomposition of a large matrix, which is cubic $O(n'^3)$ in the dimensionality of L_H in the worst case. Having said that, it is shown in the literature (Zhou et al., 2006) that it is not required to use all the eigenvectors and thus, in practice one may work with just a small fraction of n'. To this end, we work with a low-rank approximation of L_H, thereby restricting the eigen decomposition to just calculate the 100 smallest eigen vectors of L_H. To scale this operation further, we employ the use of randomized methods to perform approximate partial matrix decompositions (Halko et al., 2011), which again works very well in practice without any significant loss in accuracy. As will be explained later in Sec. 6, these randomized methods facilitate the proposed spectral clustering algorithm to achieve superior efficiency and scalability, without any loss in the efficacy.

The last step involves performing k-means clustering on the eigen-vectors U of the hypergraph laplacian matrix L_H. Since we know the number of datacenter locations a priori, which is $l = |\mathcal{L}|$, we partition U into l clusters. This operation partitions the set of data-items $\mathcal{D}$ into l different sets, thereby forming $\mathcal{P}(\mathcal{D})$, which is used as the placement decision recommended by the proposed data placement algorithm. The clustering approach used here is not a vanilla k-means, rather it includes multiple extensions. First, to ensure *load balancing* we modify the objective function of k-means to honor the desired storage distribution Φ, which provides information about the expected capacity of each datacenter location. Second, we employ the use of k-means++ initialization as proposed in (Arthur and Vassilvitskii, 2007) and parallelization to scale-up the clustering algorithm to very large datasets.

The three step process: (1) Hypergraph Laplacian construction, (2) Eigen decomposition of the hypergraph laplacian, and (3) k-means clustering on the eigen-vectors, with added extensions and modifications constitutes the proposed scalable spectral clustering algorithm.

Note that the approach discussed above is capable of solving the data placement problem without considering the possibility of replicas. Since replication may be important in real-world settings for ensuring fault-tolerance and load-balancing, we briefly discuss an extension to allow for the scenarios with replication as well. To this end, we use the proposed data placement algorithm to get a placement without replication, however, we change the capacity of each datacenter from s_{L_j} to s_{L_j}/r, where r is the desired replication factor. We then execute the same algorithm r times on different permutations of the set of datacenters $\mathcal{L}$. This allows the same data-item to be stored (in the expected sense) on r datacenters, thereby meeting the desired replication factor. With this, the proposed algorithm is extended to handle scenarios where replication is allowed as well. For the sake of brevity, we keep this discussion around extensions to scenarios with replication short. A detailed analysis and evaluation of various replication strategies will constitute as future work.

5 EVALUATION SETUP

5.1 Geo-distributed Datacenters

To simulate a real-world geo-distributed cloud environment, we employ the use of $l = 9$ geo-distributed datacenters based on the regions provided by AWS global infrastructure (aws, 2017). Note that the AWS infrastructure evolves continuously and for the sake of standardization and reproducible comparison with previous work (Yu and Pan, 2015), we only chose the 9 oldest and prominent regions, namely: Virginia, California, Oregon, Ireland, Frankfurt, Singapore, Tokyo, Sydney, and Sao Paulo, for our experimental setup. Our experimental setup closely mirrors the actual AWS setup, as the costs involved for storage and outgoing traffic as indicated in Table 2a are as advertised by Amazon. Moreover, the inter-datacenter latencies(aws, 2016) are also measured by the packet transfer latency between the chosen regions using the Linux *ping* command. Table 2b presents the average inter-datacenter latency values (in ms) between the 9 chosen datacenter regions. It is evident from the values portrayed in Table 2 that the properties exhibited by datacenters vary significantly with the region, and hence, any data placement strategy should incorporate this knowledge while performing placement decisions.

5.2 Data

The dataset used in our experiments is a trace of a location-based online social network – *Gowalla*[1], available publicly from the SNAP (sna, 2017) repository. The Gowalla dataset has been used extensively (Yu and Pan, 2015; Yu and Pan, 2017) for data placement research in geo-distributed cloud services. The social network contains 196591 vertices and 950327 edges. The vertices are the users in the social network, while the edges represent friend relationship between

[1]http://snap.stanford.edu/data/loc-gowalla.html

Table 2: (a) Traffic and Storage Costs, and (b) Inter Datacenter Latency (in ms) based on Geo-distributed Amazon Clouds.

(a) Costs (in $)

Region	Storage ($/GB-month)	Outgoing Traffic ($/GB)
Virginia	0.023	0.02
California	0.026	0.02
Oregon	0.023	0.02
Ireland	0.023	0.02
Frankfurt	0.025	0.02
Singapore	0.025	0.02
Tokyo	0.025	0.09
Sydney	0.025	0.14
Sao Paulo	0.041	0.16

(b) Latency (in ms)

Region	Virginia	California	Oregon	Ireland	Frankfurt	Singapore	Tokyo	Sydney	Sao-Paulo
Virginia	0.0	72.738	86.981	80.546	88.657	216.719	145.255	229.972	119.531
California	71.632	0.0	19.464	153.202	166.609	174.010	102.504	157.463	192.670
Oregon	88.683	19.204	0.0	136.979	159.523	161.367	89.095	162.175	182.716
Ireland	80.524	153.220	136.976	0.0	19.560	239.023	212.388	309.562	191.292
Frankfurt	88.624	166.590	159.542	19.533	0.0	325.934	236.537	323.483	194.905
Singapore	216.680	173.946	161.423	238.130	325.918	0.0	73.807	175.328	328.080
Tokyo	145.261	102.523	89.157	212.388	236.558	73.785	0.0	103.907	256.763
Sydney	229.748	157.843	161.932	309.562	323.152	175.355	103.900	0.0	322.494
Sao Paulo	119.542	192.700	181.665	191.559	194.900	327.924	256.665	322.523	0.0

two users. The trace provides 6442890 user check-ins logged over the period of February, 2009 to October, 2010. As described in Sec. 3, each user check-in consists of a data request pattern and a location. In continuation to our discussions in Sections 1 and 3, a request (indicated by a check-in) by a user v would involve pulling the data of all his/her friends. Thus, the data request pattern corresponding to a check-in by a user v is the set of data-items corresponding to all the friends of v, i.e. $\mathbf{Adj}(v)$. The location field of a user check-in are the GPS coordinates of the place from where the check-in was triggered. These GPS coordinates were mapped to the closest (in terms of distance) datacenter region to identify the source location of a data request pattern, and can thus, be one of the 9 datacenter regions as described in Sec. 5.1. With this pre-processing, we obtain a use-case scenario consisting of 196591 data-items and 102314 data request patterns.

Moving ahead, we present certain dataset statistics. Fig. 5 portrays the distribution of user check-ins across the 9 datacenter regions discussed above. It is evident that Virginia and Frankfurt register the highest ($\approx$ 40%) and the second highest ($\approx$ 30%) number of user check-ins. On the other hand, Tokyo, Sydney, and SaoPaulo get the fewest ($\approx$ 10% combined) number of user check-ins. This clearly shows a huge disparity in the check-in distribution. Based on the user check-in distribution, we extract the storage size distribution Φ of the datacenter regions, which is dependent upon both the number of check-ins registered in a region and the size of data request pattern triggered by the check-in. As is clear from Fig. 6, the desired storage distribution portrays a similar trend as that of the check-in distribution. Note that this storage distribution Φ also serves as an input to the data placement algorithm, thereby facilitating load-balancing among the 9 datacenter regions. More fundamentally, the load-balancing factor is calculated as the expected storage size at each datacenter region using the storage distribution Φ.

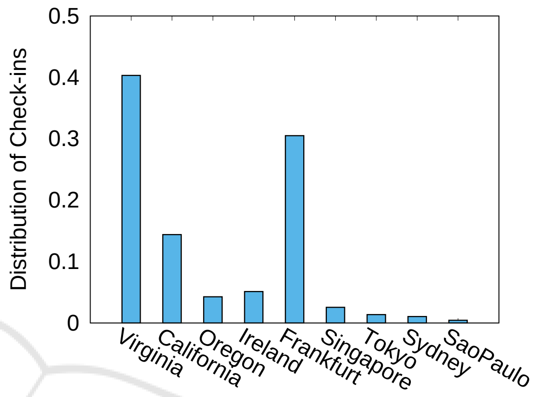


Figure 5: Distribution of user check-ins across geo-distributed datacenters.

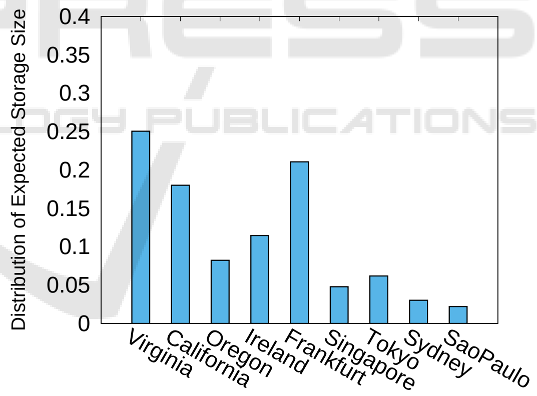


Figure 6: Distribution of expected datacenter storage size.

5.3 Algorithms

We compare the proposed data placement algorithm using spectral clustering on hypergraphs for effectiveness, efficiency, and scalability against a number of baselines – *Random* and *Nearest*, and the state-of-the-art hypergraph partitioning technique (Yu and Pan, 2015; Yu and Pan, 2017). All the algorithms were implemented in C++. To perform hypergraph partitioning for the technique proposed by (Yu and Pan, 2015) and reproduce their results we use the *PaToH* (Catalyurek, 2011) toolkit. Next, we give brief descriptions of the compared techniques:

- **Random:** partitions the set of data-items $\mathcal{D}$ randomly into $|\mathcal{L}| = l$ datacenters. To distribute the data-items according to the datacenter storage size distribution Φ, we ensure that random partitioning samples data-items based on Φ thereby ensuring load-balancing.
- **Nearest:** assigns each data-item to the datacenter from where it has been requested the highest number of times. Similar to random, to ensure load-balancing this technique follows the datacenter storage distribution Φ. Thus, once the datcenter with the highest number of requests for a particular data-item has reached its capacity, we randomly choose a datacenter location capable of serving new requests.
- **Hypergraph Partitioning:** is the data placement algorithm proposed by (Yu and Pan, 2015; Yu and Pan, 2017). After the hypergraph modeling step discussed in Sec. 4, it uses the hypergraph partitioning algorithms available in the PaToH toolkit. The partitioning algorithms maintain the datacenter storage size distribution Φ, thereby ensuring load-balancing.

5.4 Parameters

As discussed in Sec. 4, different hyperedge weights ($W_{\mathcal{R}}, W^{\kappa}_{\mathcal{R}_\mathcal{L}}, W^{\mathcal{S}}_{\mathcal{R}_\mathcal{L}}, W^{\Gamma}_{\mathcal{R}_\mathcal{L}}$) facilitate optimization of different objectives. As stated in Eq. 3, a weight vector $\mathbb{W}$ facilitates prioritization of these objectives based on the assigned weights. In our study, we incorporate the use of specific weight vector $\mathbb{W}$ settings: $\mathbb{W}_1 : \{100, 1, 1, 1\}$, $\mathbb{W}_2 : \{1, 100, 1, 1\}$, $\mathbb{W}_3 : \{1, 1, 100, 1\}$, and $\mathbb{W}_4 : \{1, 1, 1, 100\}$, which represent different preferences or importance of the considered evaluation metrics, such as, higher priority of collocating the associated data-items thereby minimizing the datacenter span $\mathcal{N}(\cdot)$, lower inter-datacenter traffic $\Gamma(\cdot)$, lower inter-datacenter latency $\kappa(\cdot)$, and lower storage cost $\mathcal{S}(\cdot)$ respectively. Note that in all the weight-vector settings, the value 100 is just used to indicate higher relative importance of the corresponding metric. The results portrayed are not dependent on the specific value of 100, rather the weight-vectors can work with any value as long as it is $>> 1$.

5.5 Evaluation Metrics

- **Span** ($\mathcal{N}(\cdot)$)**:** of a data request pattern R_i is defined as the average number of datacenters required to be accessed to fetch the data-items requested in R_i. Further, the span for the entire workload is calculated as the average of the datacenter spans of each request pattern $R_i \in \mathcal{R}$.
- **Traffic** ($\Gamma(\cdot)$)**:** The total traffic cost of a data request pattern R_i is defined as the sum of outgoing traffic prices of the datacenters involved in outgoing requests for the data-items in R_i. Further, the traffic cost of the entire workload is calculated as the sum of traffic costs of each request pattern $R_i \in \mathcal{R}$.
- **Latency** ($\kappa(\cdot)$)**:** The inter-datacenter latency of a data request pattern R_i is calculated as the sum of access latencies required to fetch all the data-items requested in R_i from the datacenter where they are placed to the datacenter from where the request was triggered. Further, the latency of the workload is calculated as the sum of the latencies of each request pattern $R_i \in \mathcal{R}$.
- **Storage** ($\mathcal{S}(\cdot)$)**:** The sum of the total cost on storing all of the data-items corresponding to every data request pattern $R_i \in \mathcal{R}$ in datacenters $\mathcal{L}$ prescribed by the data placement algorithm.
- **Balance:** is calculated as the pearson's correlation coefficient between the expected storage size distribution Φ, and the actual storage size distribution obtained after performing data placement. If the value is close to 1, it means that the two distributions are highly similar, while they are dissimilar if the value is close to -1.
- **Objective. (Obj.):** is defined as the weighted sum of the considered performance metrics, where the weights are described using the weight vector $\mathbb{W}$.

6 EVALUATION RESULTS

All the experiments were done using codes written in C++ on an Intel(R) Xeon(R) E5-2698 28-core machine with 2.3 GHz CPU and 256 GB RAM running Linux Ubuntu 16.04. Owing to their nondeterministic nature, results for the random and hypergraph partitioning methods are averaged over 10 runs. Note that the results portrayed corresponding to each evaluation metric (barring Balance) are normalized in the scale of $[0, 1]$ by dividing each value by the highest observed value in that particular metric. Normalization ensures that all the values lie in a common range ($[0, 1]$), thereby facilitating joint analysis of all the considered evaluation metrics for all the algorithms. Since the problem formulation in this study is concerned with the minimization of the evaluation metrics, the smaller the portrayed values the better the performance is. Additionally, note that the results for

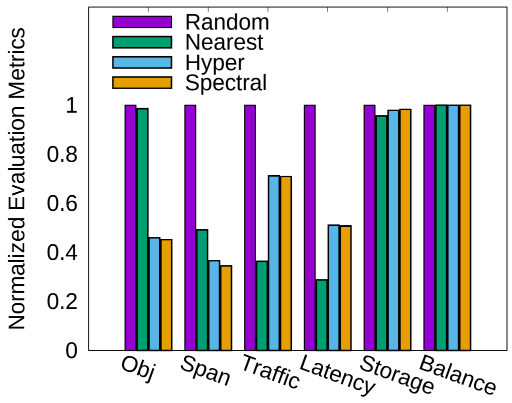


Figure 7: $\mathbb{W}_1 = \{100, 1, 1, 1\}$: Higher priority towards minimizing the datacenter span $\mathcal{N}(\cdot)$.

the balance evaluation metric are close to 1 for all the techniques considered in this study. This is because every technique possesses the capability to honor the desired storage size distribution Φ.

Fig. 7 presents the results corresponding to the weight-vector setting $\mathbb{W}_1$, where minimizing the datacenter span holds the highest priority in the optimization objective. It is evident that both, the proposed spectral clustering algorithm (Spectral), and the state-of-the-art hypergraph partitioning algorithm (Hyper) achieve a low value on the overall optimization objective (Obj), while being significantly better than the random and nearest methods. This is because of their capability to preferentially minimize the datacenter span, which possesses the highest priority in the optimization under $\mathbb{W}_1$. Note that Nearest outperforms both Hyper and Spectral on the traffic and latency metrics, as they have lower weights in the optimization objective under $\mathbb{W}_1$. However, both Spectral and Hyper are still significantly better than the Random method.

A similar behavior is observed in Figs. 8, 9, and 10 corresponding to the other three weight vector settings $\mathbb{W}_2$, $\mathbb{W}_3$, and $\mathbb{W}_4$ respectively. Both Spectral and Hyper outperform the other methods significantly on the overall optimization (Obj), while also being significantly better on the corresponding evaluation metric that the weight-vector setting is tuned to optimize. More fundamentally, in addition to being better on Obj., both Spectral and Hyper outperform the other methods in minimizing inter-datacenter traffic cost $\Gamma(\cdot)$, inter-datacenter latency $\kappa(\cdot)$, and storage cost $\mathcal{S}(\cdot)$, when a higher preference is given to these metrics under the weight-vector settings $\mathbb{W}_2$, $\mathbb{W}_3$, and $\mathbb{W}_4$ respectively.

The main limitation of Nearest is that it tries to assign each data-item to a datacenter with the highest number of accesses to that data-item, thereby aiming to minimize (on an average) the geographical distance between the data-item and the source location of the data request oblivious to the fact that the actual traffic or storage costs might not be proportional to the distance. The main advantage of both Spectral and Hyper over Nearest is that owing to their higher-order modeling capabilities they are capable of better addressing multi-objective optimizations, and possess the capability to adapt their performance based on different weight-vector settings. This is evident from the results portrayed in Figs. 7– 10. To further emphasize on the capability to adapt the optimization based on different weight-vector settings, we discuss the results portrayed in Fig. 10. It is clear that according to $\mathbb{W}_4$, the optimization objective gives more preference towards minimizing the storage cost. Note that storage cost and other parameters like inter-datacenter traffic and latency might be inversely related to each other, i.e., a lower storage cost might lead to higher latencies or traffic cost. This behavior is also evident from Fig. 10, where both Spectral and Hyper achieve lower storage costs, thereby also achieving better performance on Obj, however, suffer slightly on other metrics. Thus, methods like Nearest would find difficulty in handling such cases, while both Spectral and Hyper possess the capability to adapt the optimization based on the weight-vector.

Figs. 7– 10 show that the performance of both Spectral and Hyper are quite similar on the evaluated metrics. While Spectral is always marginally better in efficacy when compared to Hyper, the major advantage of Spectral over Hyper comes from its capability to scale gracefully and efficiently to large datasets. It is intuitive that scalability is a paramount property for any data placement algorithm, since the scale of real-world social networks or for that matter any real-world data-intensive services is humongous. Fig. 11 shows that Spectral is up to *10 times* ($\approx$ 3–4 times on average) *faster* when compared to Hyper, while also

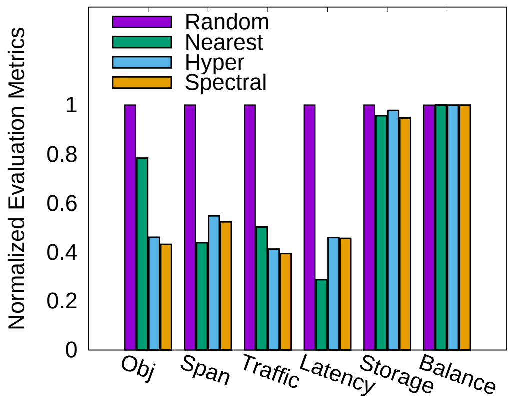


Figure 8: $\mathbb{W}_2 = \{1, 100, 1, 1\}$: Higher priority towards minimizing the inter-datacenter traffic $\Gamma(\cdot)$.

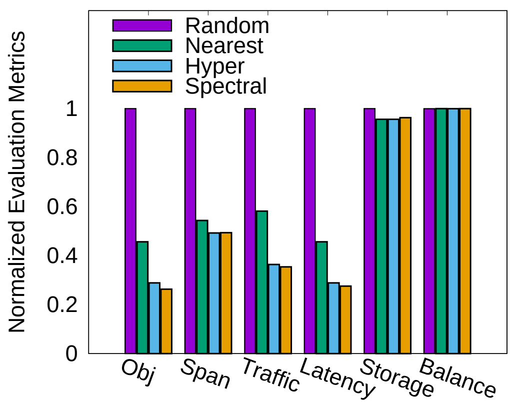


Figure 9: $\mathbb{W}_3 = \{1,1,100,1\}$: Higher priority towards minimizing the inter-datacenter latency $\kappa(\cdot)$.

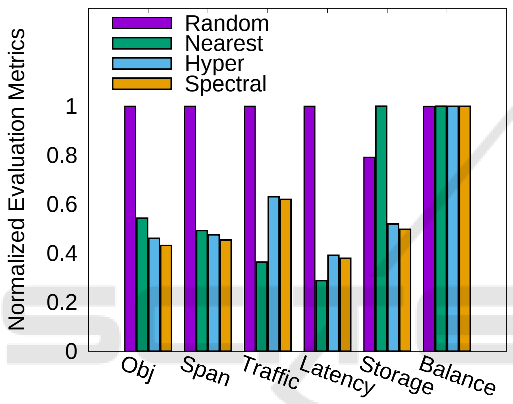


Figure 10: $\mathbb{W}_4 = \{1,1,1,100\}$: Higher priority towards minimizing the storage cost $\mathcal{S}(\cdot)$.

being slightly better in all evaluated metrics for the majority of the considered weight-vector settings.

In summary, through extensive experiments we verify that the proposed spectral clustering algorithm is *efficient, scalable, and effective*. Although Spectral is not always the best on every evaluated metric, it serves to be the most effective technique in terms of improving Obj, which is the main target of our multi-objective optimization. Additionally, it possesses the capability to adapt to the change in weight vector settings $\mathbb{W}$, which facilitates handling of a variety of real-world scenarios as described by different weight vectors.

7 CONCLUSIONS

In this paper, we have addressed the problem of data placement of data-intensive services into geo-distributed clouds. We identified the need for specialized methods to perform data placement for data-intensive services, as contrary to MapReduce style workloads, these workloads require access to multiple datasets within a single transaction, thus, rendering traditional methods of hash based partitioning inadequate. Consequently, we devised a scalable spectral clustering algorithm for hypergraph partitioning, thereby facilitating data placement. Experiments on a real-world trace-based social network dataset portrayed the effectiveness, efficiency, and scalability of the proposed algorithm. In the future, we would like to design and develop a distributed version of our algorithms and implement the entire system on state-of-the-art big data technologies like Apache Spark. Additionally, we would also like to include the notion of replicas directly in the data placement problem formulation.

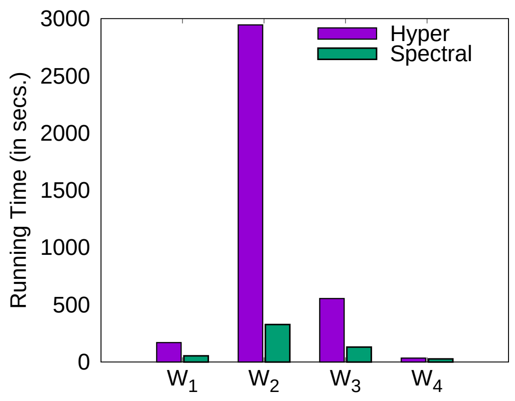


Figure 11: Efficiency of spectral clustering over hypergraph partitioning.

ACKNOWLEDGEMENTS

This research is partly funded by the IWT SBO DeCoMAdS project.

REFERENCES

(2015). How much data does X store? https://techexpectations.org/2015/03/28/how-much-data-does-x-store/.

(2016). Latency Between AWS Global Regions. http://zhiguang.me/2016/05/10/latency-between-aws-global-regions/.

(2017). AWS Global Infrastructure. https://aws.amazon.com/about-aws/global-infrastructure/.

(2017). SNAP Datasets. https://snap.stanford.edu/data/.

(2017). The Exponential Growth of Data. https://insidebigdata.com/2017/02/16/the-exponential-growth-of-data/.

Agarwal, S., Dunagan, J., Jain, N., Saroiu, S., Wolman, A., and Bhogan, H. (2010). Volley: Automated Data

Placement for Geo-distributed Cloud Services. In *NSDI*.

Arthur, D. and Vassilvitskii, S. (2007). k-means++: The Advantages of Careful Seeding. In *SODA*, pages 1027–1035.

Catalyurek, U. V. (2011). PaToH (Partitioning Tool for Hypergraphs). http://bmi.osu.edu/umit/PaToH/ manual.pdf.

Catalyurek, U. V., Boman, E. G., Devine, K. D., Bozdag, D., Heaphy, R., and Riesen, L. A. (2007). Hypergraph-based Dynamic Load Balancing for Adaptive Scientific Computations. In *IPDPS*, pages 1–11.

Chervenak, A., Deelman, E., Livny, M., Su, M., Schuler, R., Bharathi, S., Mehta, G., and Vahi, K. (2007). Data Placement for Scientific Applications in Distributed Environments. In *GRID*.

Deerwester, S. C., Ziff, D. A., and Waclena, K. (1990). An Architecture for Full Text Retrieval Systems. In *DEXA*, pages 22–29.

Ebrahimi, M., Mohan, A., Kashlev, A., and Lu, S. (2015). BDAP: A Big Data Placement Strategy for Cloud-Based Scientific Workflows. In *BigDataService*, pages 105–114.

Gibson, D., Kleinberg, J., and Raghavan, P. (1998). Inferring Web Communities from Link Topology. In *HYPERTEXT*, pages 225–234.

Golab, L., Hadjieleftheriou, M., Karloff, H., and Saha, B. (2014). Distributed Data Placement to Minimize Communication Costs via Graph Partitioning. In *SSDBM*, pages 1–12.

Halko, N., Martinsson, P., and Tropp, J. A. (2011). Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Review*, 53(2):217–288.

Han, S., Kim, B., Han, J., K.Kim, and Song, J. (2017). Adaptive Data Placement for Improving Performance of Online Social Network Services in a Multicloud Environment. In *Scientific Programming*, pages 1–17.

Huguenin, K., Kermarrec, A. M., Kloudas, K., and Taïani, F. (2012). Content and Geographical Locality in User-generated Content Sharing Systems. In *NOSSDAV*, pages 77–82.

Jiao, L., Li, J., Du, W., and Fu, X. (2014). Multi-objective data placement for multi-cloud socially aware services. In *INFOCOM*, pages 28–36.

Kayaaslan, E., Cambazoglu, B. B., and Aykanat, C. (2013). Document Replication Strategies for Geographically Distributed Web Search Engines. *Inf. Process. Manage.*, 49(1):51–66.

Li, X., Zhang, L., Wu, Y., Liu, X., Zhu, E., Yi, H., Wang, F., Zhang, C., and Yang, Y. (2017). A Novel Workflow-Level Data Placement Strategy for Data-Sharing Scientific Cloud Workflows. *IEEE TSC*, PP(99):1–14.

Liu, X. and Datta, A. (2011). Towards Intelligent Data Placement for Scientific Workflows in Collaborative Cloud Environment. In *IPDPSW*, pages 1052–1061.

Meila, M. and Shi, J. (2001). Learning Segmentation by Random Walks. In *NIPS*, pages 873–879.

Ng, A. Y., Jordan, M. I., and Weiss, Y. (2001). On Spectral Clustering: Analysis and an Algorithm. In *NIPS*, pages 849–856.

Nishtala, R., Fugal, H., Grimm, S., Kwiatkowski, M., Lee, H., Li, H. C., McElroy, R., Paleczny, M., Peek, D., Saab, P., Stafford, D., Tung, T., and Venkataramani, V. (2013). Scaling Memcache at Facebook. In *NSDI*, pages 385–398.

Quamar, A., Kumar, K. A., and Deshpande, A. (2013). SWORD: Scalable Workload-aware Data Placement for Transactional Workloads. In *EDBT*, pages 430–441.

Rochman, Y., Levy, H., and Brosh, E. (2013). Resource placement and assignment in distributed network topologies. In *INFOCOM*, pages 1914–1922.

Shankaranarayanan, P. N., Sivakumar, A., Rao, S., and Tawarmalani, M. (2014). Performance Sensitive Replication in Geo-distributed Cloud Datastores. In *DSN*, pages 240–251.

Shi, J. and Malik, J. (2000). Normalized Cuts and Image Segmentation. *PAMI*, 22(8):888–905.

Spielmat, D. A. (1996). Spectral Partitioning Works: Planar Graphs and Finite Element Meshes. In *FOCS*, pages 96–105.

Yu, B. and Pan, J. (2015). Location-aware associated data placement for geo-distributed data-intensive applications. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 603–611.

Yu, B. and Pan, J. (2016). Sketch-based data placement among geo-distributed datacenters for cloud storages. In *INFOCOM*, pages 1–9.

Yu, B. and Pan, J. (2017). A Framework of Hypergraph-based Data Placement among Geo-distributed Datacenters. *IEEE TSC*, PP(99):1–14.

Yuan, D., Yang, Y., Liu, X., and Chen, J. (2010). A data placement strategy in scientific cloud workflows. *FGCS*, 26(8):1200 – 1214.

Zhang, J., Chen, J., Luo, J., and Song, A. (2016). Efficient location-aware data placement for data-intensive applications in geo-distributed scientific data centers. *Tsinghua Science and Technology*, 21(5):471–481.

Zhou, D., Huang, J., and Schölkopf, B. (2006). Learning with Hypergraphs: Clustering, Classification, and Embedding. In *NIPS*, pages 1601–1608.