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 geographically 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-
The data placement problem has been studied extensively in the literature spanning a wide-variety of areas of machine learning with semi-supervised learning 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.

(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.

\begin{table}[h]
\centering
\caption{Summary of notations used.}
\begin{tabular}{|l|l|}
\hline
\textbf{Item} & \textbf{Definition} \\
\hline
\hline
$V$ & The set of users in the social network $V = \{v_1, v_2, \ldots, v_n\}$. \\
$E$ & The set of edges in the social network $E = \{(v_i, v_j)\} \subseteq E$. \\
\hline
\textbf{Adj}(v) & The set of friends of the user $v \in V$. \\
\hline
$\mathcal{D}$ & The set of data-items $\mathcal{D} = \{d_1, d_2, \ldots, d_s\}$. \\
\hline
$\mathcal{L}$ & The set of datacenters and their locations $\mathcal{L} = \{l_1, l_2, \ldots, l_t\}$. \\
\hline
$\mathcal{R}$ & The set of request patterns $\mathcal{R} = \{R_1, R_2, \ldots, R_h\}$. \\
\hline
$\mathcal{C}$ & The set of user check-ins $\mathcal{C} = \{C_{i,j} = (v_i, l_j) | (v_i) \in \mathcal{C}, (l_j) \in \mathcal{L}\}$. \\
\hline
$\mathcal{P}$ & The hypergraph incidence matrix. \\
\hline
$\mathcal{P}_0$ & The hyperedge weight matrix. \\
\hline
$\Phi$ & The desired datacenter storage distribution. \\
\hline
$\mathcal{P}^*(\mathcal{D})$ & A partition on the set of data-items $\mathcal{D}$. \\
\hline
$\mathcal{F}(l_i)$ & Cost (per unit) of outgoing traffic from datacenter $l_i$. \\
\hline
$\mathcal{w}(l_i, l_j)$ & Inter datacenter latency (directed) between $l_i$ and $l_j$. \\
\hline
$\mathcal{S}(l_i)$ & Storage cost (per unit) of datacenter $l_i$. \\
\hline
$\mathcal{N}(R)$ & Average number of datacenters accessed by request $R$. \\
\hline
\end{tabular}
\end{table}
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 \(E\) denotes the set of edges (denoting friendship) between any two users of the social network. The set \(D\) 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 \(C_v = (R_v, L_v) \subseteq C\) composed of a data request pattern \(R_v \in D\) and a datacenter location \(L_v \in L\). Each data request pattern \(R_v\) is comprised of a set of data-items \(D \subseteq D\) requested by a user \(v \in V\). As discussed in Sec. 1, considering the location based social networks use-case, a data request pattern \(R_v\) belonging to a check-in \(C_v\) registered by a user \(v\) would correspond to the data-items corresponding to all the friends of \(v\), i.e. \(R_v = \{d_u \mid u \in \text{Adj}(v)\}\). Moving ahead, \(L_v \in L\) represents the location of a datacenter capable of serving user requests. The location of a check-in \(C_v\) is decided as the datacenter location \(L_v\) closest (in distance) to the actual physical location of the user check-in. In other words, the check-in \(C_v\) signifies a request for data-items contained in \(R_v\) triggered from the datacenter located at \(L_v\).

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 \(R_1 = \{d_1, d_3, d_4, d_5\}\), \(R_2 = \{d_1, d_6\}\), and \(R_3 = \{d_1, d_6\}\). Let us also label the datacenter locations as \(L_1 = \text{Virginia}, L_2 = \text{California}, L_3 = \text{Frankfurt}, L_4 = \text{Sydney},\) and \(L_5 = \text{Tokyo}\). 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 \(C_4)\) 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 \(D\), \(m\) user check-ins \(C_k = (R_k, L_k) \subseteq C\), each comprising a data request pattern \(R_k\) being originated from a datacenter located at \(L_k\), a set of \(l\) datacenters with locations in \(L\), with the per unit cost of outgoing traffic from each datacenter \(\forall L_j \in L, \Gamma(L_j)\), the per unit storage cost of each datacenter \(\forall L_j \in L, \kappa(L_j, L_j')\), the inter datacenter latency (directed) for each pair of datacenters \(\forall L_j, L_j' \in L, \xi(L_j, L_j')\), and the average number of datacenters accessed by the data-items requested in each request pattern \(R_k\) being \(N(R_k)\), perform data placement to minimize the optimization objective \(O\), which is defined as the weighted average of \(\Gamma(\cdot), \kappa(\cdot, \cdot), \xi(\cdot, \cdot), \) and \(N(\cdot)\).

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 \(D\) and the set of user check-ins \(C\) comprising the set of data request patterns \(R\) and their locations \(L\), 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 \(\Pi\) constructed, next, we partition the set of data-items \(D\) into \(l\) parts corresponding to the \(L\) datacenters according to the desired storage distribution \(\Phi\), 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.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure3.png}
\caption{Overview of our Approach.}
\end{figure}
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: $\mathcal{D}, \mathcal{C}, \mathcal{R}_C, \mathcal{L}, G(V,E), \Phi$

Output: Partitioning of the set of data-items $\mathcal{P}(\mathcal{D})$ into $l$ datacenters

1: $(\Pi, \mathcal{W}_H) \leftarrow \text{ConstructHypergraph}(\mathcal{D}, \mathcal{C}, G(V,E))$
2: $\mathcal{P}(\mathcal{D}) \leftarrow \text{SpectralClustering}(\Pi, \mathcal{W}_H, l, \Phi)$
3: return $\mathcal{P}(\mathcal{D})$

In the context of our problem statement, every user check-in $C_i \in \mathcal{C}$ consists of a datacenter location $L_j \in \mathcal{L}$ and a data request pattern $R_i \in \mathcal{R}_C$. 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. $\text{Adj}(v)$, as portrayed by the social network $G(V,E)$. Since a data request pattern involves data-item – datacenter location 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 $d_i$ 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,

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

Let $\mathcal{R}_C = \{ \forall d \in R_i, \forall L_j \in \mathcal{L} \mid \exists C_k = (R_i, L_j) \in \mathcal{C}, R_{ij} \}$ denote the set of edges connecting data-items with datacenter locations corresponding to all the data request patterns $R_i \in \mathcal{R}_C$ 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 $\mathcal{R}_C$ and all the data-item – datacenter location edges $\mathcal{R}_L$. Thus, the number of hyperedges in the hypergraph are $|E_H| = m' = r + nl$. Formally,

$$E_H = \mathcal{R}_C \cup \mathcal{R}_L \quad (2)$$

Fig. 4a portrays the hypergraph representation of the data-items and request patterns as presented in Fig. 2. The data-items $\{d_1, \ldots, d_7\}$ 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 $d_2, \ldots, d_7$ 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 $he4 = (d_7, L_1)$ and $he10 = (d_4, L_1)$ represents the relationship between the data-item $d_7$ and the datacenter locations where it was requested, namely – Virginia ($L_1$) and Tokyo ($L_5$). Fig. 4b portrays the incidence matrix $\Pi$ 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 $d_2, \ldots, d_7$ the entries in $\Pi$ 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 $\Pi$, 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 $W_R$ 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 $W_R$ 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 $(W^x_{R_1}, W^S_{R_1}, W^T_{R_1})$ corresponding to data-item – datacenter hyperedges $(\mathcal{R}_C)$ facilitate minimization of inter datacenter latency $\kappa(L_1, L_2)$, storage cost $S(L_1)$, and cost of outgoing traffic $\Gamma(L_1)$ respec-
respectively, by giving priority to placing data-items at data-center locations from where they have been requested more frequently.

The resultant hyperedge weight matrix $W_H$ of $\Pi$ is a diagonal matrix of size $m' \times m'$, which is defined as the weighted sum of $W_{R_{\mathcal{h}}}$, $W_{V_{\mathcal{h}}}^k$, $W_{V_{\mathcal{h}}}^S$, and $W_{V_{\mathcal{h}}}^\Gamma$. Mathematically,

$$W_H = \mathbb{W} \cdot (W_{R_{\mathcal{h}}} W_{V_{\mathcal{h}}}^k W_{V_{\mathcal{h}}}^S W_{V_{\mathcal{h}}}^\Gamma).$$

(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 $\Pi$ and a $m' \times m'$ diagonal hyperedge weight matrix $W_H$. As discussed previously, the hypergraph incidence matrix $\Pi = [he_1, he_2, ..., he_m]$ possesses $m'$ hyperedges, where each hyperedge $he_i = [he_{i,1}, he_{i,2}, ..., he_{i,n'}]$ is a $n'$-dimensional binary vector. The entry $he_{ij} = 1$ indicates that the $j^{th}$ vertex in the hypergraph vertex set is participating in the $i^{th}$ hyperedge, while $he_{ij} = 0$ indicates otherwise. With this, the hypergraph laplacian $L_H$ is defined as:

$$L_H = I - (D_v^{-1/2} \Pi W_H D_h^{-1} \Pi^T D_v^{-1/2})$$

(4)

where, $I$ is a $n' \times n'$ identity matrix; $D_v$ is a $n' \times n'$ diagonal vertex degree matrix defined as $D_v = \text{diag}(\sum \Pi)$; $D_h$ is a $m' \times m'$ diagonal hyperedge degree matrix defined as $D_h = \text{diag}(\sum \Pi^T)$; $W_H$ is a $m' \times m'$ diagonal hyperedge weight matrix defined as $W_H = \text{diag}([W_1, W_2, ..., W_m])$, and $W_1, ..., W_m$ 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:

$$L_H = U \Lambda V$$

(5)

where, $U = [u_1, ..., u_{n'}]$ is a $n' \times n'$ matrix formed by the eigen-vectors of $L_H$; $\Lambda = \text{diag}(\lambda_1, ..., \lambda_{n'})$ 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 $\Lambda$ 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 eigenvectors 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 $\mathcal{P}^l(\mathcal{D})$, 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 $\Phi$, 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 $s_j$, to $s_j/r$, 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 – Gowalla1, 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

1http://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.

<table>
<thead>
<tr>
<th>Region</th>
<th>Storage (GB-month)</th>
<th>Outgoing Traffic (GB/s)</th>
<th>Virginia</th>
<th>California</th>
<th>Oregon</th>
<th>Ireland</th>
<th>Frankfurt</th>
<th>Singapore</th>
<th>Tokyo</th>
<th>Sydney</th>
<th>Sao Paulo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virginia</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
</tr>
<tr>
<td>California</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
</tr>
<tr>
<td>Oregon</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
</tr>
<tr>
<td>Ireland</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
</tr>
<tr>
<td>Frankfurt</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
</tr>
<tr>
<td>Singapore</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
</tr>
<tr>
<td>Tokyo</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
</tr>
<tr>
<td>Sao Paulo</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
<td>10.0</td>
</tr>
</tbody>
</table>

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 de-
criptions of the compared techniques:

- **Random**: partitions the set of data-items $D$ randomly into $|L| = l$ datacenters. To distribute the data-items according to the datacenter storage size distribution $\Phi$, we ensure that random partitioning samples data-items based on $\Phi$ 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 $\Phi$. Thus, once the datacenter 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 $\Phi$, thereby ensuring load-balancing.

5.4 Parameters

As discussed in Sec. 4, different hyperedge weights $(W_R, W^k_R, W^S_R, W^T_R)$ facilitate optimization of different objectives. As stated in Eq. 3, a weight vector $W$ facilitates prioritization of these objectives based on the assigned weights. In our study, we incorporate the use of specific weight vector $W$ settings: $W_1 : \{100, 1, 1, 1\}$, $W_2 : \{1, 100, 1, 1\}$, $W_3 : \{1, 1, 100, 1\}$, and $W_4 : \{1, 1, 1, 100\}$, 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 $L$ prescribed by the data placement algorithm.

- **Balance**: is calculated as the pearson’s correlation coefficient between the expected storage size distribution $\Phi$, 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 $W$.

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 non-deterministic 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...
Observe Span Traffic Latency Storage Balance
Normalized Evaluation Metrics

<table>
<thead>
<tr>
<th>Method</th>
<th>Obj</th>
<th>Span</th>
<th>Traffic</th>
<th>Latency</th>
<th>Storage</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nearest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hyper</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectral</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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 \( \Phi(\cdot) \).

Fig. 7 presents the results corresponding to the weight-vector setting \( \mathcal{W}_1 \), 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 \( \mathcal{W}_1 \). Note that Nearest outperforms both Hyper and Spectral on the traffic and latency metrics, as they have lower weights in the optimization objective under \( \mathcal{W}_1 \). 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 \( \mathcal{W}_2, \mathcal{W}_3, \) and \( \mathcal{W}_4 \) 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 \( S(\cdot) \), when a higher preference is given to these metrics under the weight-vector settings \( \mathcal{W}_2, \mathcal{W}_3, \) and \( \mathcal{W}_4 \) 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 \( \mathcal{W}_4 \), 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 (≈3–4 times on average) faster when compared to Hyper, while also
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 $W$, 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.

ACKNOWLEDGEMENTS

This research is partly funded by the IWT SBO De-CoMaD$S$ project.

REFERENCES

Placement for Geo-distributed Cloud Services. In NSDI.