Unifying Data and Replica Placement for Data-intensive Services in Geographically Distributed Clouds

Ankita Atrey[1], Gregory Van Seghbroeck[1], Higinio Mora[2], Filip De Turck[1] and Bruno Volckaert[1]
[1]*IDLAB-imec, Technologie Park, Ghent University, Ghent, Belgium*
[2]*University of Alicante, Alicante, Spain*

Keywords: Data Placement, Replica Placement, Geographically Distributed Clouds, Location-Based Services, Online Social Networks, Scalability, Overlapping Clustering.

Abstract: The increased reliance of data management applications on cloud computing technologies has rendered research in identifying solutions to the data placement problem to be of paramount importance. The objective of the classical data placement problem is to optimally partition, while also allowing for replication, the set of data-items into distributed data centers to minimize the overall network communication cost. Despite significant advancement in data placement research, replica placement has seldom been studied in *unison* with data placement. More specifically, most of the existing solutions employ a two-phase approach: 1) data placement, followed by 2) replication. Replication should however be seen as an integral part of data placement, and should be studied as a joint optimization problem with the latter. In this paper, we propose a unified paradigm of data placement, called *CPR*, which *combines data placement and replication* of data-intensive services into geographically distributed clouds as a joint optimization problem. Underneath CPR, lies an overlapping correlation clustering algorithm capable of assigning a data-item to multiple data centers, thereby enabling us to jointly solve data placement and replication. Experiments on a real-world trace-based online social network dataset show that CPR is effective and scalable. Empirically, it is $\approx 35\%$ better in efficacy on the evaluated metrics, while being up to 8 times faster in execution time when compared to state-of-the-art techniques.

1 MOTIVATION

With the emergence of *Cloud computing*, *Big Data*, and *Internet of Things (IoT)*, the rate at which data is being generated is increasing exponentially (ins, 2017; gro, 2018). Although advancements in modern hardware, cloud computing, and big data technologies have enabled development of multiple distributed systems (such as Hadoop (White, 2012) and Apache Spark (Zaharia et al., 2016)) that have significantly enriched the field of scalable data management, effective strategies for data partitioning and placement remain cardinal to the performance of such systems. Consistent with the requirement of *data-intensive services* to *access multiple datasets within each transaction* (Golab et al., 2014; Yu and Pan, 2015; Zhao et al., 2016a; Zhao et al., 2016b; Shabeera et al., 2017), specialized solutions for data placement have recently been proposed based on hypergraph partitioning (Yu and Pan, 2017) and spectral clustering on hypergraphs (Atrey et al., 2018). This is because for data-intensive services, traditional solutions (as employed by Hadoop or Spark) based on uniform partitioning of data-items using hashing may result in a huge volume of data migrations (Golab et al., 2014; Atrey et al., 2018), thereby leading to network congestion and eventually reduced system throughput.

Online social networks (OSNs) are one of the most prevalent instances of data-intensive services in the real-world (Yu and Pan, 2017). Consider a location-based OSN service as presented in Fig. 1. A sample 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 $\{v_3, v_5, v_6, v_7\}$ are friends of the user v_4. Similarly $\{v_2, v_3\}$ are friends of v_1. The list of all the friends of every user is also portrayed in a table in Fig. 1. There exists a notion of a data-item corresponding to each user of the social network, which represents the most recent snapshot (e.g. profile picture, videos, textual posts etc.) of her profile. As shown in Fig. 1, the data-item corresponding to the user v_1 is denoted as $d(v_1)$, that of user v_2 is denoted as $d(v_2)$ and so on. Addition-

Atrey, A., Van Seghbroeck, G., Mora, H., De Turck, F. and Volckaert, B.
Unifying Data and Replica Placement for Data-intensive Services in Geographically Distributed Clouds.
DOI: 10.5220/0007613400250036
In *Proceedings of the 9th International Conference on Cloud Computing and Services Science (CLOSER 2019)*, pages 25-36
ISBN: 978-989-758-365-0

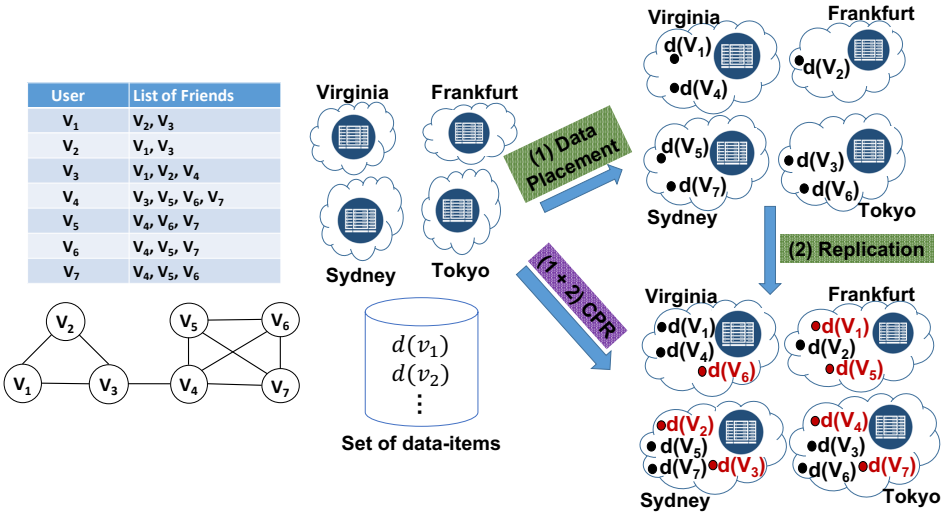

Figure 1: Location Based OSN Service: The standard data placement process (in green): where the data-items are first placed in data centers (black dots) and then replicated (red dots); and the proposed CPR paradigm (in magenta): where both data placement and replication is performed in a single step.

ally, each user can register a check-in, which is assigned to a data center nearest (in geographical distance) to her check-in location. Each user check-in requires retrieval of data from her friends, constituting a data request pattern triggered by this check-in. For example, while registering a check-in in Virginia the user v_7 may want to tag/mention some of her friends. This would require the data-items corresponding to her friends $\{v_4, v_5, v_6\}$ to be available at the Virginia data center, thereby triggering a data request for transferring data-items $\{d(v_4), d(v_6), d(v_7)\}$ to Virginia.

Motivated by the use-case discussed above, the problem of *combined data placement and replication (CPR)* for data-intensive services in data centers that are distributed geographically across the world is the topic of research tackled in this paper.

A careful analysis of the objectives of the generalized data placement problem (Golab et al., 2014) indicates that both data and replica placement are tightly coupled, and should be considered as part of a single optimization problem. In fact, data placement without replication is a specialized instance of the generic data placement problem. Additionally, support for replication is important to ensure fault tolerance. Thus, replica placement or replication is an integral part of the data placement problem. Having said that, despite continued and rigorous advancement of the state-of-the-art in the field of data placement (Golab et al., 2014; Yu and Pan, 2017; Atrey et al., 2018), to the best of our knowledge and as detailed in Sec. 2, none of the existing techniques are capable of jointly performing data and replica placement. More specifically, most of the techniques in the literature employ an *ad hoc two-phase approach* – data placement followed by replication (Fig. 1) – each independently catering to a sub-part of the overall optimization objective, which results in solutions of inferior quality.

To bridge this gap, a unified paradigm of combining data and replica placement, *CPR*, for data-intensive services in geo-distributed clouds is proposed in this paper. We propose an overlapping correlation clustering algorithm to solve the joint optimization problem of data and replica placement. Specifically, overlapping clustering enables a data-item to be assigned to multiple data centers, thereby facilitating both *data placement and replication in a single step*. Key contributions of this work are as follows:

- We study a novel variant of the data placement problem, *CPR*, for data-intensive services in geo-distributed data centers, which aims at *unifying data and replica placement* as a *joint optimization* problem (Sec. 3).
- We propose a novel algorithm using *overlapping correlation clustering* on hypergraphs, which can assign the same data-item to different data centers (Sec. 4). This facilitates jointly solving the data placement and replication problem, as opposed to the existing two-phase approach employed by the state-of-the-art. The proposed algorithm solves a multi-objective optimization, where among others, it facilitates optimization of latency, storage cost, inter data center traffic, and data center span.
- Through extensive experiments on a real-world trace-based social network dataset (Sec. 5), we show that the proposed overlapping clustering algorithm is scalable and effective.

2 RELATED WORK

The data placement problem, being a fundamental question in computer science, has witnessed a huge amount of research over the years, with techniques being developed for different execution environments such as: grid (Kosar and Livny, 2004; Kosar and Livny, 2005; Ding and Lu, 2009), distributed (Chervenak et al., 2007; Golab et al., 2014), and cloud computing (Yu et al., 2012; Guo and Wang, 2013; Li et al., 2017; Ferdaus et al., 2017) environments. From the perspective of the type of workloads, traditionally the focus was on scientific workflows (Yuan et al., 2010; Liu and Datta, 2011; Ebrahimi et al., 2015) and relational workloads such as database joins (Golab et al., 2014), however, of late the focus is moving towards workloads arising from niche applications such as OSN services (Jiao et al., 2014; Han et al., 2017) and data intensive services in geo-distributed clouds (Agarwal et al., 2010; Yu and Pan, 2015; Yu and Pan, 2016; Zhang et al., 2016; Yu and Pan, 2017). Since the focus of this paper is on combining data placement and replication for data-intensive services in geo-distributed data centers, we present a review of existing research that overlap with our work.

Any successful solution to the data placement problem in geo-distributed data centers should provide two capabilities, namely – capturing and improving (1) data-item – data-item associations (i.e., the number of times two data-items were requested together); and (2) data-item – data center associations (i.e., the number of times a data-item was requested at a given data center). On the one hand, a frequent pattern mining based technique proposed by (Nishtala et al., 2013), and a hierarchical clustering algorithm on correlations between data-items proposed by (Zhao et al., 2016b; Zhao et al., 2016a), facilitate management of data-item – data-item associations. On the other hand, methods proposed by (Agarwal et al., 2010; Huguenin et al., 2012; Rochman et al., 2013; Zhang et al., 2016) facilitate capturing of data-item – data center associations. Specifically, (Agarwal et al., 2010) proposed a system Volley that analyzes logs of data center requests to perform automatic data placement in geographically distributed data centers. (Rochman et al., 2013) design robust data placement algorithms to ensure that a large fraction of region specific requests is served at a lower cost, while managing the highly dynamic nature of user requests. (Zhang et al., 2016) propose an integer programming based algorithm for minimizing the data communication cost while honoring the data center storage capacities. However, these methods do not possess both the aforementioned capabilities.

Of late, literature has seen an increased use of hypergraph-based techniques for data placement in geo-distributed clouds, which also constitute the current state-of-the-art. Yu et al. (Yu and Pan, 2015; Yu and Pan, 2017) propose data placement strategies using hypergraph modeling and publicly available partitioning heuristics (Catalyurek, 2011) for data-intensive services. While hypergraph-based modeling facilitates capturing of both data-item – data-item and data-item – data center associations, the hypergraph partitioning heuristics available in (Catalyurek, 2011) facilitate these techniques to scale to large datasets. Recently, Atrey et al. (Atrey et al., 2018) presented a spectral clustering algorithm that employed the use of low-rank approximations of the hypergraph laplacian to obtain superior efficiency and scalability while retaining the same efficacy as portrayed by (Yu and Pan, 2017). Hypergraph based partitioning solutions (Catalyurek et al., 2007) have also been used in grid and distributed computing environments.

There also exists research pertaining to other aspects of geo-distributed data placement, such as placement in multi-clouds and the design of specialized replication strategies. The technique proposed by (Jiao et al., 2014) facilitates data placement in a multi-cloud environment and solves a multi-objective optimization to minimize the carbon footprint and inter cloud traffic. (Han et al., 2017) introduce an algorithm to perform data migration decisions for OSN services in a multicloud environment, which is capable of adapting to the changing data traffic. Location-aware replication strategies[1] capable of optimizing on metrics such as location of geo-distributed data centers, and inter data center communication costs were proposed by (Shankaranarayanan et al., 2014). However, none of these techniques capture both data-item – data-item and data-item – data center associations. Thus, we choose the techniques presented by (Yu and Pan, 2017) and (Atrey et al., 2018) as the representative state-of-the-art methods for comparison.

Having said that, to the best of our knowledge, none of the existing state-of-the-art methods described above are capable of unifying data and replica placement as a joint optimization problem. Specifically, as discussed in Sec. 1 the techniques existing in the literature employ a *two-phase approach*, where replication is performed as an independent step after obtaining the data-item assignments from a data placement algorithm. This ad hoc two-phase approach has several disadvantages such as *sub-optimal replica placement quality*, and decreased efficiency owing to solving two independent optimization prob-

[1]The reader is referred to (Grace and Manimegalai, 2014) for a survey of replica placement algorithms.

lems instead of one. To this end, the research presented in this paper proposes a unified data placement paradigm – *CPR*, capable of jointly performing data and replica placement of data-intensive services into geographically distributed clouds, through a novel approach of data partitioning using overlapping correlation clustering on Hypergraphs. More fundamentally, overlapping clustering allows a data-item to be assigned to multiple data centers (clusters) at the same time, thereby facilitating both data and replica placement in a single step. In other words, the proposed overlapping clustering based algorithm provides a unified solution to the combined data and replica placement problem for data-intensive services.

3 PROBLEM STATEMENT

Given a set of data-items, data centers, data request patterns, and the replication factor, the objective of the combined (or generalized) data and replica placement problem (CPR) is to intelligently place the data-items, allowing for replication wherever applicable, across data centers so as to minimize the overall communication cost resulting from migration/replication[2] of data-items corresponding to different data requests. Note that analogous to most of the techniques in data placement literature (Golab et al., 2014; Ferdaus et al., 2017; Zhao et al., 2016b; Yu and Pan, 2017; Atrey et al., 2018), we consider the system workload represented by data request patterns to be static. Thus, the proposed algorithm is offline, and has to be re-executed from scratch to accommodate changes in the system workload. Designing algorithms that can accommodate changes in the workload in an online manner would constitute as future work.

Next, we introduce some basic concepts of data placement in the context of OSN services, followed by a formal description of the CPR problem for data-intensive services in geo-distributed data centers.

A location based online social network (Fig. 1) possesses two aspects: (1) a social network connecting users with their friends, and (2) a capability for the users to register *check-ins* at potentially different locations across the globe.

Definition 1 (Social Network. $(G(V,E))$)**.** *A social network with n individuals and m social ties can be denoted as a graph $G(V,E)$, where V is the set of nodes representing the users of the social network, $|V|=n$, and E is the set of edges (representing friend relationships) between any two nodes, $E \subseteq V \times V$, $|E|=m$.*

In the context of data placement, a data-item is an atomic unit of data storage and transfer. Thus, for the OSN use-case:

Definition 2 (Data-items ($\mathcal{D}$).)**.** *A data-item is defined as the most recent snapshot of a user's profile (e.g. profile picture, posts, comments etc.). The set $\mathcal{D}$ contains n data-items corresponding to each user $v \in V$ of the social network, where the data-item for a user v is denoted as $d(v)$.*

Moving ahead, a check-in depicts a social network user visiting any location in the world. Each user check-in is composed of two parts: (1) a location where the check-in was recorded, and (2) a data request pattern triggered by the check-in. As discussed in Sec. 1, the location of a user check-in is decided as the location of a data center closest (in distance) to the actual physical location of the user check-in.

Definition 3 (Data-centers ($\mathcal{L}$).)**.** *A data center constitutes a set of resources to store the data-items and perform different computational tasks on the stored data-items. Each data center is hosted at a location $L_j \in \mathcal{L}$, where $|\mathcal{L}|=l$ denotes the set of data center locations.*

Further, the data request corresponding to a user check-in requires retrieval of the data-items of her friends (Sec. 1). Usually for large scale systems such as OSNs, the data-items (profiles of OSN users) are distributed across data centers and might require migration/replication from one data center to another. The data-items that are potential candidates for migrations constitute a data request, which is formally defined as follows.

Definition 4 (Data-request Patterns ($\mathcal{R}$).)**.** *A data request pattern $R(v) \in \mathcal{R}$ corresponding to a check-in by a user v at the data center location L_j is comprised of the set of data-items corresponding to all the friends of v. Mathematically, $R(v) = \{d(u) \mid u \in \boldsymbol{Adj}(v)\}$. Further, $R(v)$ denotes the set of data-items that are required to be collocated in the same data center L_j, and those that are not stored in L_j are communicated (either migrated or replicated) from the data centers in which they are stored to L_j. The set of data request patterns denoted as $\mathcal{R}$ represents the system workload.*

As an example, the data request pattern for a check-in by the user v_1 (Fig. 1) is denoted as $R(v_1) = \{d(v_2), d(v_3)\}$. Given this information, a check-in is formally defined as follows:

[2]Migration or replication of data-items may involve additional overheads such as data-item retrieval delays, packet loss etc. For the sake of brevity, the focus of this paper is on minimizing the communication cost alone, however, the proposed data-placement algorithm is generic, and not restricted in its scope based on this assumption.

Definition 5 (Check-ins. ($\mathcal{C}$))**.** *A check-in is a tuple* $\forall k_{1\leq k\leq \rho}$, $C_k = (R(v), L_j) \in \mathcal{C}$ *consisting of a data request pattern* $R(v) \in \mathcal{R}$ *triggered by* v *and a location* $L_j \in \mathcal{L}$ *of a data center capable of serving user requests. The set* $\mathcal{C}$ *contains* ρ *user check-ins.*

In other words, the check-in C_k by a user v at a location L_j signifies a request for the data-items contained in $R(v)$ triggered from the data center located at L_j. For example, if the user v_3 in Fig. 1 was the first to register a check-in among all other users, which was recorded at the L_4=Tokyo data center, then $C_1 = (R(v_3), L_4)$, where $R(v_3) = \{d(v_1), d(v_2), d(v_4)\}$.

Note that a user can register multiple check-ins at the same location, and to better capture data-item – data-item and data-item – data center associations, each individual check-in is treated as different from the other. For example, if two data-items $d(v_2)$ and $d(v_3)$ are requested together seven times they would possess a stronger data-item – data-item association than data-items $d(v_1)$ and $d(v_2)$ that coexist in data request patterns just twice. Similarly, if a user v_5 visited Sydney five times the data-items in $R(v_5)$ would possess a stronger data-item – data center association with Sydney when compared to that of any other data center which was visited less often. To capture this, for each check-in by the user v_5 at L_3=Sydney there would be 5 different check-ins denoted as $C_k, \ldots, C_{k+5}$, each composed of the data request pattern $R(v_5)$ and the location L_3. Moreover, this also substantiates the reason behind not indexing each user check-in uniquely using data request patterns R and locations L_j.

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

Problem. *Given a set of n data-items* $\mathcal{D}$ *corresponding to the set of social network users* V, ρ *user check-ins* $C_k = (R(v), L_j) \in \mathcal{C} \mid v \in V, L_j \in \mathcal{L}$ *representing the system workload, each comprising a data request pattern* $R(v)$ *being originated from a data center located at* L_j*, a set of* l *data centers with locations in* $\mathcal{L}$*, with the per unit cost of outgoing traffic from each data center* $\Gamma(L_j) \mid L_j \in \mathcal{L}$*, the per unit storage cost of each data center* $\mathcal{S}(L_j) \mid L_j \in \mathcal{L}$*, the inter data center latency (directed) for each pair of data centers* $\kappa(L_j, L_{j'}) \mid L_j, L_{j'} \in \mathcal{L}$*, the average number of data centers spanned by the data-items corresponding to each request pattern* $R(v)$ *being* $\mathcal{N}(R(v))$*, and the replication factor r, perform combined data and replica placement to minimize the optimization objective O, which is defined as the weighted average*[3] *of* $\Gamma(\cdot), \kappa(\cdot,\cdot), \mathcal{S}(\cdot)$*, and* $\mathcal{N}(\cdot)$.

[3]The weights determine the relative importance of these metrics towards the overall optimization objective, and would be discussed in Sec. 4.1.

4 OVERLAPPING CLUSTERING ON HYPERGRAPHS

Given the set of data-items $\mathcal{D}$ and the set of user check-ins $\mathcal{C}$ representing the system workload, the first step is to construct a hypergraph. This results in a higher-order representation of the interaction between the data-items and the data centers in the form of the hypergraph incidence matrix Π, and the hyperedge weight matrix $\mathcal{W}_\Pi$ representing the relative importance of the constructed hyperedges. The next step is to partition the set of data-items $\mathcal{D}$ into l data centers such that each data-item $d(v) \in \mathcal{D}$ is assigned to $r < l$ (allowing for replication) data centers, which is achieved using the proposed overlapping clustering algorithm. An overview of the proposed technique is presented in Fig. 2.

4.1 Hypergraph Construction

Literature on data placement of data intensive services (Yu and Pan, 2017; Atrey et al., 2018) has provided sufficient evidence in support of hypergraphs as the most suitable choice for modeling the interactions between both data-items – data-items and data-items – data centers. Hyperedges allow to model relationships between several vertices as opposed to just a pair of vertices in traditional graphs. Being a more sophisticated construct a hypergraph $H(V_H, E_H)$ serves as a generalization over a graph $G(V, E)$. With this capability to capture *multi-way* relationships, hypergraphs provide a powerful representation to model data-item – data-item and data-item – data center associations.

The system workload represented using user check-ins gives rise to two types of hyperedges E_H: (1) hyperedges corresponding to data request patterns $\mathcal{R}$ that connect all the data-items (data-item – data-item association) in a data request triggered from a user check-in; and (2) hyperedges $\mathcal{R}_\mathcal{L}$ that connect the data-items requested in a check-in with the data center location (data-item – data center association) where the check-in was registered. Thus, the set of vertices V_H in the constructed hypergraph consist of the set of data-items $\mathcal{D}$ and data center locations $\mathcal{L}$, totaling to $|V_H| = n' = n + l$ vertices, and $|E_H| = m' = r + nl$ hyperedges. Eq. 1 formally defines these two sets.

$$\begin{aligned} V_H &= \mathcal{D} \cup \mathcal{L} \\ E_H &= \mathcal{R} \cup \mathcal{R}_\mathcal{L} \end{aligned} \tag{1}$$

Given that there are two different types of hyperedges, there are two types of weights corresponding to either type, with each focused towards optimizing

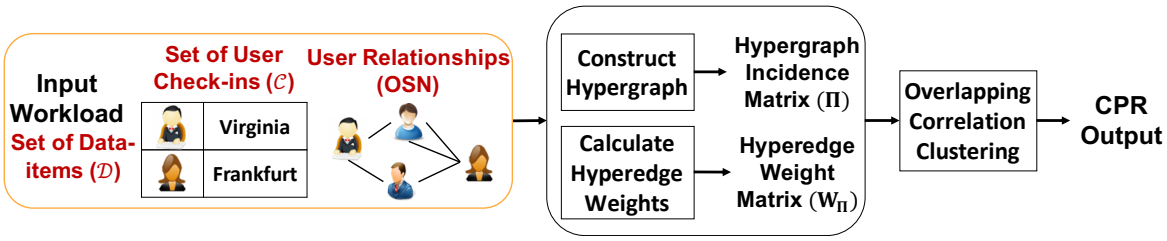

Figure 2: Overview of the proposed overlapping correlation clustering technique for combined data and replica placement.

a different aspect of the problem. More specifically, the weight $W_{\mathcal{R}}$ corresponds to the data request pattern hyperedges, and its aim is to facilitate minimization of $\mathcal{N}(R_i)$: the average number of data centers accessed by a data request pattern R_i, which is achieved by enforcing the data-items that are requested together to be placed together. On the other hand, the weights $(W^{\kappa}_{\mathcal{R}_\mathcal{L}}, W^{\mathcal{S}}_{\mathcal{R}_\mathcal{L}}, W^{\Gamma}_{\mathcal{R}_\mathcal{L}})$ correspond to the data-item – data center hyperedges, and their aim is to minimize inter data center latency $\kappa(L_j, L_{j'})$, storage cost $\mathcal{S}(L_j)$, and cost of outgoing traffic $\Gamma(L_j)$ respectively, by giving higher priority to placing data-items at data center locations from where they have been requested more frequently. The resultant hyperedge weight matrix is then constructed as the weighted sum of the four weights discussed above, which is mathematically defined as:

$$\mathcal{W}_\Pi = \mathbb{W} \cdot (W_{\mathcal{R}}, W^{\kappa}_{\mathcal{R}_\mathcal{L}}, W^{\mathcal{S}}_{\mathcal{R}_\mathcal{L}}, W^{\Gamma}_{\mathcal{R}_\mathcal{L}}). \quad (2)$$

where, $\mathcal{W}_\Pi$ is a diagonal matrix of size $m' \times m'$, and $\mathbb{W}$ is the weight vector for deciding the priorities of the previously discussed hyperedge weighting strategies[4].

The hypergraph $H(V_H, E_H)$ is represented using a $n' \times m'$ dimensional hypergraph incidence matrix Π, which possesses m' hyperedges, and each hyperedge is a n'-dimensional binary column vector. It is formally defined as:

$$\forall he_i \in E_H, he_i^T = [he_{1,i}, he_{2,i}, \ldots, he_{n',i}].$$
$$\Pi = [he_1, he_2, \ldots, he_{m'}]. \quad (3)$$

An 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.

Overlapping correlation clustering requires a similarity matrix denoting similarities between each vertex pair in the (hyper)graph as input. To this end, we construct the normalized hypergraph matrix N_H, which requires two additional operations on the hypergraph incidence matrix Π. We compute two diagonal matrices – the vertex degree matrix ($D_{v\Pi}$) and the hyperedge degree matrix ($D_{he\Pi}$) of dimensionality $n' \times n'$ and $m' \times m'$ respectively. The vertex degree matrix captures the number of hyperedges each vertex of the hypergraph is a part of, while the hyperedge degree matrix measures the number of vertices contained in each hyperedge. Mathematically,

$$D_{v\Pi} = diag(\sum \Pi). \quad (4)$$
$$D_{he\Pi} = diag(\sum \Pi^T). \quad (5)$$

where, $\sum X$ represents the row-wise sum of the input matrix X and X^T represents the transpose of the matrix X.

With this, the normalized hypergraph matrix N_H is mathematically defined as:

$$N_H = \left(D_{v\Pi}^{-1/2} \cdot \Pi \cdot \mathcal{W}_\Pi \cdot D_{he\Pi}^{-1} \cdot \Pi^T \cdot D_{v\Pi}^{-1/2}\right) \quad (6)$$

where, $D_{v\Pi}$ is a $n' \times n'$ diagonal vertex degree matrix, $D_{he\Pi}$ is a $m' \times m'$ diagonal hyperedge degree matrix, and $\mathcal{W}_\Pi$ is a $m' \times m'$ diagonal hyperedge weight matrix. Thus, N_H becomes a $n' \times n'$ matrix.

4.2 Overlapping Correlation Clustering

We begin with a description of correlation clustering: where given a complete graph with edges labeled as positive or negative, the objective is to identify a partitioning of the graph such that it minimizes the sum of the number of positively labeled edges cut and the number of negatively labeled edges not cut by the partition. In the current scenario, the input is a normalized hypergraph similarity matrix N_H representing the pair-wise similarity between data-items, and a set $\mathcal{L}$ of l labels representing the data center locations. The task of correlation clustering is to find a mapping $\mathcal{F} : V_H \rightarrow \mathcal{L}$ for partitioning the set of data-items into l data centers, that minimizes the following loss function:

$$\mathbb{L}_{Correlate}(V_H, \mathcal{F}) = \sum_{\substack{(u,v)\in V_H \times V_H \\ \mathcal{F}(u)=\mathcal{F}(v)}} (1 - N_H(u,v)) + \sum_{\substack{(u,v)\in V_H \times V_H \\ \mathcal{F}(u)\neq\mathcal{F}(v)}} N_H(u,v). \quad (7)$$

[4]For additional details about the hypergraph construction and the hyperedge weight calculation steps the reader is referred to (Yu and Pan, 2017; Atrey et al., 2018).

Algorithm 1: Overlapping Clustering Algorithm.

Input: Π, $\mathcal{W}_\Pi$, l, r, Φ
Output: Partitioning of the hypergraph vertex set $\mathcal{P}(V_H)$ into l clusters allowing r replicas
1: $D_v(\Pi) \leftarrow diag(\sum \Pi)$; $D_{he}(\Pi) \leftarrow diag(\sum \Pi^T)$
2: Compute normalized hypergraph N_H as described in Eq. 6
3: Randomly initialize the label sets of size r for each data-item $u \in V_H$
4: **while** $\mathbb{L}_{Overlap}(V_H, \mathcal{F})$ decreases **do**
5: **for each** $u \in V_H$ **do**
6: find the label set F that minimizes $\mathbb{L}^u_{Overlap}(F|\mathcal{F})$
7: Update $\mathcal{F}(u) \leftarrow F$
8: **end for**
9: **end while**
10: **return** $\mathcal{P}(V_H)$ defined by $\mathcal{F}$

As discussed in Sec. 1, the goal of overlapping clustering is to partition the set of data-items $\mathcal{D}$ into l data centers, with each data-item being assigned to more than one data center to appropriately allow for replication. To achieve this, instead of mapping each data-item to a single label (corresponding to a data center), it is mapped to a set of labels thereby allowing each data-item to be associated with multiple data centers. Given the label set definition as the set of all subsets of data center locations $\mathcal{L}$ except the empty set: $\mathcal{L}^+ = 2^{\mathcal{L}} \setminus \{\emptyset\}$, and a similarity function over the data-item label sets $\mathbb{S}(\cdot)$, the underlying optimization objective reduces to identifying a mapping $\mathcal{F} : V_H \rightarrow \mathcal{L}^+$ under which the similarity between any pair of data-items $\forall u, v \in V_H$, $N_H(u,v)$ agrees as much as possible with the similarity between their corresponding label sets $\mathbb{S}(\mathcal{F}(u), \mathcal{F}(v))$.

Similar to the loss function for correlation clustering $\mathbb{L}_{Correlate}$, the loss function for overlapping correlation clustering is defined as:

$$\begin{aligned}&\mathbb{L}_{Overlap}(V_H, \mathcal{F}) \\ &= \sum_{(u,v)\in V_H \times V_H} |\mathbb{S}(\mathcal{F}(u), \mathcal{F}(v)) - N_H(u,v)|. \\ &= \sum_{u\in V_H} \sum_{v \in V_H \setminus \{u\}} |\mathbb{S}(\mathcal{F}(u), \mathcal{F}(v)) - N_H(u,v)|. \end{aligned} \quad (8)$$

where $\mathbb{S}(\cdot)$ is defined as the set-intersection indicator function:

$$\mathbb{S}(X,Y) = \begin{cases} 1, & \text{if } X \cap Y \neq \emptyset. \\ 0, & \text{otherwise.} \end{cases} \quad (9)$$

Formally, the goal of overlapping clustering is to find a mapping $\mathcal{F}^*$ in order to minimize $\mathbb{L}_{Overlap}(V_H, \mathcal{F})$, which is mathematically denoted as:

$$\mathcal{F}^* = \underset{\mathcal{F}}{\operatorname{argmin}} \mathbb{L}_{Overlap}(V_H, \mathcal{F}). \quad (10)$$

Overlapping correlation clustering was shown to be NP-Hard in (Bonchi et al., 2013), thus, there does not exist any polynomial time algorithm capable of solving it. To this end, we propose a greedy algorithm that iteratively refines the quality of the label sets of one vertex (data-item) in the hypergraph at a time. More specifically, given the label sets of all the other vertices in the hypergraph, the greedy algorithm applies a local optimization (on one vertex) to improve the cost of the overall solution until convergence. Algorithm 1 presents the pseudo-code of the proposed overlapping clustering algorithm.

We begin by computing the normalized hypergraph matrix N_H as described in Eq. 6 (lines 1–2). Next, each vertex $u \in V_H$ is initialized with a random label set of size r (equal to the replication factor), thereby facilitating each data-item to be assigned to r data centers (line 3). Post the initialization, we apply the greedy local optimization approach described above (lines 4–9). More specifically, the label set of each node $u \in V_H$ is iteratively improved, keeping the label sets of all the other nodes fixed, till the overall loss $\mathbb{L}_{Overlap}(V_H, \mathcal{F})$ converges. To understand the loss with respect to each node u, we rewrite Eq. 8 as:

$$\mathbb{L}(V_H, \mathcal{F}) = \sum_{u\in V_H} \mathbb{L}^u_{Overlap}(\mathcal{F}(v) \mid \mathcal{F}). \quad (11)$$

where

$$\begin{aligned}&\mathbb{L}^u_{Overlap}(\mathcal{F}(v) \mid \mathcal{F}) \\ &\quad = \sum_{v\in V_H \setminus \{u\}} |\mathbb{S}(\mathcal{F}(u), \mathcal{F}(v)) - N_H(u,v)| \end{aligned} \quad (12)$$

5 EXPERIMENTS

In this section, the proposed CPR paradigm and overlapping clustering algorithm is evaluated through experiments on a large scale location-based OSN.

5.1 Dataset

Following the literature on data placement in geo-distributed cloud services (Yu and Pan, 2017; Atrey et al., 2018), we extract a trace from the *Gowalla*[5] social network dataset, available publicly from the SNAP (sna, 2017) repository. The social network consists of 196591 vertices corresponding to the users of the social network, and 950327 edges, each representing friend relationship between two users. Additionally, the dataset contains 6442890 user check-ins registered from February 2009 to October 2010, triggering a total of 102314 data request patterns.

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

Table 1: (a) Traffic and Storage costs, and (b) Inter data center latency 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	3.523	72.738	86.981	80.546	88.657	216.719	145.255	229.972	119.531
California	71.632	5.842	19.464	153.202	166.609	174.010	102.504	157.463	192.670
Oregon	88.683	19.204	5.551	136.979	159.523	161.367	89.095	162.175	182.716
Ireland	80.524	153.220	136.976	5.005	19.560	239.023	212.388	309.562	191.292
Frankfurt	88.624	166.590	159.542	19.533	4.425	325.934	236.537	323.483	194.905
Singapore	216.680	173.946	161.423	238.130	325.918	5.870	73.807	175.328	328.080
Tokyo	145.261	102.523	89.157	212.388	236.558	73.785	6.846	103.907	256.763
Sydney	229.748	157.843	161.932	309.562	323.152	175.355	103.900	4.889	322.494
Sao Paulo	119.542	192.700	181.665	191.559	194.900	327.924	256.665	322.523	6.076

5.2 Experimental Setup

All experiments are performed using code 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 corresponding to the random, hypergraph partitioning, and spectral clustering methods are averaged over 10 runs.

We simulate a real-world geo-distributed cloud environment based on the AWS global infrastructure(aws, 2017a). To ensure consistency with previous research (Yu and Pan, 2015; Atrey et al., 2018), we use the $l = 9$ oldest and prominent AWS data center regions, namely: Virginia, California, Oregon, Ireland, Frankfurt, Singapore, Tokyo, Sydney, and Sao Paulo. To closely mirror the actual AWS setup, the costs involved for storage and outgoing traffic are as advertised by Amazon. Moreover, the inter data center latencies between the chosen regions are measured by the packet transfer latency using the Linux *ping* command(aws, 2017b). These data center characteristics are presented in Table 1.

An analysis of user check-ins revealed the existence of disparity in their check-in behavior. Some data centers (ex: Virginia and Frankfurt) register a very high number of check-ins, while others (ex: SaoPaulo and Sydney) receive very few. This effects the amount of storage required at each data center, which is dependent upon both the number of check-ins registered in a region and the size of data request pattern triggered by each check-in. Based on this, the storage size for each data center region $\forall L_j \in \mathcal{L}$ is calculated as $S_j = \sum |R(v)|| \; \exists C_k = (R(v), L_j), L_j \in \mathcal{L}$. Let $S = \sum_{j=1}^{l} S_j$ be the total storage size, then the data center storage size follows a multinomial distribution and is calculated as: $\Phi \sim [\frac{S_1}{S}, \frac{S_2}{S}, \ldots, \frac{S_l}{S}]$. The expected storage size at each data center calculated using Φ serves as the load-balancing factor, and is input to the data placement algorithm to facilitate load-balancing among the 9 data center regions.

To summarize, for the Gowalla dataset the data placement task reduces to partitioning 196591 data-items corresponding to the social network users into 9 data centers based on the 102314 data request patterns triggered from user check-ins.

Baselines: We compare the proposed data placement algorithm for effectiveness, efficiency, and scalability with four baselines.

- **Random:** partitions the set of data-items $\mathcal{D}$ randomly into $|\mathcal{L}|$ data centers.
- **Nearest:** assigns each data-item to the data center from where it has been requested the highest number of times.
- **Hypergraph Partitioning (Hyper):** is the data placement algorithm proposed by (Yu and Pan, 2015; Yu and Pan, 2017), which uses the hypergraph partitioning algorithms available in the PaToH toolkit (Catalyurek, 2011).
- **Spectral Clustering (Spectral):** is the data placement algorithm proposed by (Atrey et al., 2018), which uses fast approximate eigen decomposition methods for efficiently performing spectral clustering on hypergraphs.

As discussed in Sec. 2, Hyper (Yu and Pan, 2017) and Spectral (Atrey et al., 2018) serve as the representative state-of-the-art methods for data placement of data-intensive services in geo-distributed data centers. To ensure load balancing, all the techniques partition the data-items according to the data center storage size distribution Φ. Thus, the results for the balance evaluation metric are close to 1 for all the techniques considered in this study.

Parameters. The weight vector $\mathbb{W}$ (Eq. 2 facilitates optimization of different objectives by prioritizing different hyperedge weights (Sec. 4.1) which represent different preferences or importance towards the considered evaluation metrics. To this end, we experiment with varying settings for $\mathbb{W}$: $\mathbb{W}_1 : \{100,1,1,1\}$ to minimize the data center span $\mathcal{N}(\cdot)$; $\mathbb{W}_2 : \{1,100,1,1\}$ for minimizing the inter data center traffic $\Gamma(\cdot)$; $\mathbb{W}_3 : \{1,1,100,1\}$ to minimize the inter data center latency $\kappa(\cdot)$; and $\mathbb{W}_4 : \{1,1,1,100\}$ for minimizing the storage cost $\mathcal{S}(\cdot)$. Note that in all the weight-vector settings, the value 100 is just used to in-

dicate higher relative importance of the corresponding metric. The portrayed results are not dependent on the specific value of 100, and can be reproduced with any value as long as it is $>> 1$. Further, Spectral uses 100 smallest eigen-vectors of the hypergraph laplacian for spectral clustering. Following best practices in data storage management (rep, 2018), the replication factor r was set to 3.

Evaluation Metrics. We consider two categories of evaluation metrics. The first type is concerned with the efficiency of the studied algorithms, while the other is concerned with their efficacy.

- **Efficiency:** We evaluate the efficiency of the methods using their execution time, i.e., the time required to produce the data placement output.
- **Efficacy:** of the studied methods is measured across the following metrics.
  - **Span ($\mathcal{N}(\cdot)$):** of a data request pattern $R(v)$ is defined as the average number of data centers required to be accessed to fetch the data-items requested in $R(v)$. The span for the entire workload is calculated as the average of the data center spans of each request pattern $R(v) \in \mathcal{R}$.
  - **Traffic ($\Gamma(\cdot)$):** The total traffic cost of a data request pattern $R(v)$ is defined as the sum of outgoing traffic prices of the data centers involved in outgoing requests for the data-items in $R(v)$. The traffic cost of the entire workload is calculated as the sum of traffic costs of each request pattern $R(v) \in \mathcal{R}$.
  - **Latency ($\kappa(\cdot)$):** The inter data center latency of a data request pattern $R(v)$ is calculated as the sum of access latencies required to fetch all the data-items requested in $R(v)$ from the data center where they are placed to the data center from where the request was triggered. The latency of the entire workload is calculated as the sum of the latencies of each request pattern $R(v) \in \mathcal{R}$.
  - **Storage ($\mathcal{S}(\cdot)$):** The sum of the total cost on storing all of the data-items corresponding to every data request pattern $R(v) \in \mathcal{R}$ in data centers $\mathcal{L}$ 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 $\mathbb{W}$.

Note that the results portrayed corresponding to each evaluation metric (barring Balance) have been normalized in the scale of $[0,1]$ by dividing each value by the highest observed value in that particular metric. For example, let $nmax = max_{\forall R(v) \in \mathcal{R}}(\mathcal{N}(R(v)))$ be the highest observed span value, then the span for each data request pattern $R(v)$ is normalized as: $\mathcal{N}(R(v))/nmax \mid \exists R(v) \in \mathcal{R}$. A similar operation is performed for other evaluation metrics as well. Normalization ensures that all the values lie in a common range, thereby ensuring equal and fair contribution of each evaluation metric towards *Obj*. Additionally, note that the optimization problem underneath CPR is concerned with the minimization of the evaluation metrics (barring Balance), hence, the smaller the portrayed values the better the performance is.

5.3 Evaluation Results: Quality Metrics

Figs. 3– 6 present the results on the considered evaluation metrics corresponding to different weight vector settings: $\mathbb{W}_1$–$\mathbb{W}_4$. It is evident that the proposed overlapping clustering algorithm (Overlap) performs the best (achieving the least value) on the overall optimization objective (Obj) across all the weight vector settings, while being significantly better than the random and nearest methods. Additionally, Overlap also outperforms Hyper and Spectral by being up to 30–40% and 20–30% better respectively.

Redirecting our focus to other evaluation metrics, it can be noticed that Nearest outperforms Hyper, Spectral, and Overlap in some cases, however, the latter are still significantly better than the Random method. For instance consider Fig. 3, it can be observed that Nearest is better on the inter data center traffic and latency metrics. This is because according to the weight vector setting $\mathbb{W}_1$, minimizing the data center span holds the highest priority while traffic and latency metrics have lower weights in the optimization objective. A similar behavior is observed for the other three weight vector settings: $\mathbb{W}_2$, $\mathbb{W}_3$, and $\mathbb{W}_4$ as well (Figs. 4– 6). To understand this observed behavior better, let us analyze the results presented in Fig. 6. It is not hard to infer that storage cost might be inversely related to other parameters such as inter data center latency and traffic. Therefore, preferentially optimizing to achieve lower storage costs ($\mathbb{W}_4$) thereby also obtaining better performance on Obj, might lead a technique to suffer on other metrics, i.e., a lower storage cost might lead to higher latencies or traffic cost. Despite this behavior, most importantly Overlap significantly outperforms all the

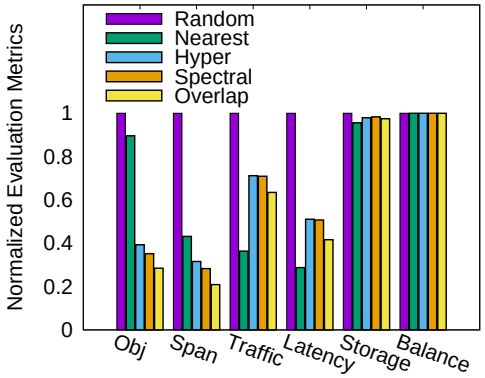

Figure 3: Overlap results in reducing the data center span $\mathcal{N}(\cdot)$ by $\approx 35\%$ when compared to Spectral with $\mathbb{W}_1 = \{100,1,1,1\}$.

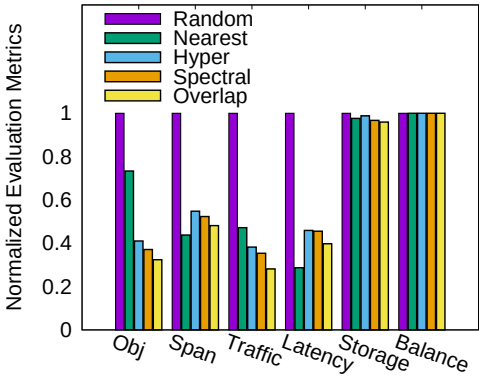

Figure 4: Overlap results in reducing the inter data center traffic $\Gamma(\cdot)$ by $\approx 26\%$ when compared to Spectral with $\mathbb{W}_2 = \{1,100,1,1\}$.

considered baselines on the corresponding evaluation metric that the weight-vector setting is tuned to optimize. More fundamentally, in addition to being better on Obj., Overlap outperforms the other methods in minimizing the data center span $\mathcal{N}(\cdot)$, inter data center traffic cost $\Gamma(\cdot)$, inter data center latency $\kappa(\cdot)$, and storage cost $\mathcal{S}(\cdot)$, when a higher preference is given to these metrics under the weight-vector settings $\mathbb{W}_1$, $\mathbb{W}_2$, $\mathbb{W}_3$, and $\mathbb{W}_4$ respectively.

Moving ahead, we analyze the reason behind the sub-optimal performance of the Nearest method. The main limitation is that Nearest is inclined to assign each data-item to a data center that receives the highest number of access requests for that data-item, which consequently results in minimizing (on an average) the geographical distance between the data-item and the source location of the data request. Note that this optimization strategy is oblivious to the fact that the storage or traffic costs might not be correlated with the distance, thereby leading to sub-optimal performance in real-world settings that require multi-objective optimization. We also refer the reader to Table 2, which presents a quantitative summary of the performance of all the considered baselines indicating how worse each baseline is relative to Overlap.

Based on the above analysis, it is clear that Hyper, Spectral, and Overlap possess the capability to adapt the optimization based on the input weight vector setting. This is because of their higher-order modeling capabilities courtesy hypergraphs, which renders them better suited for performing multi-objective optimizations. Further, since Overlap models data placement and replication as a joint optimization problem (CPR), it achieves better performance on the evaluation metrics when compared to both Hyper and Spectral that solve each problem independently.

5.4 Evaluation Results: Efficiency and Scalability

As analyzed in Sec. 5.3, Hyper, Spectral, and Overlap stand out as techniques possessing good quality on the evaluation metrics, with Overlap being the technique possessing the best quality ($\approx$ 30–40% better) among them. In this section, we study their execution time performance on the Gowalla dataset. Fig. 7 clearly portrays the *superior efficiency* of Overlap when compared to Hyper and Spectral, where on average it is $\approx$ 4–5 *faster* when compared to Hyper, and $\approx$ 2–3 *faster* when compared to Spectral across different weight vector settings. Given that the scale of real-world social networks of today is humongous, the ability to scale to large datasets is a paramount property for any data placement algorithm. Thus, this capability to efficiently and gracefully scale to large datasets serves as one of the major advantages of Overlap over Hyper and Spectral.

In summary, through extensive experiments we verify that the proposed overlapping clustering algo-

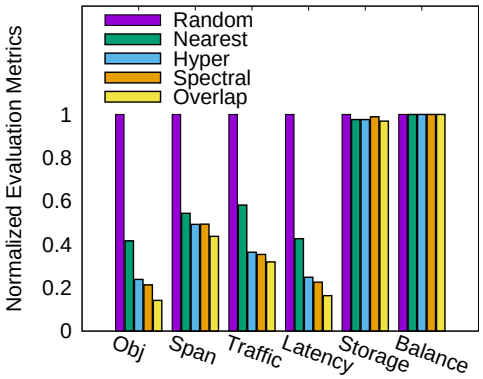

Figure 5: Overlap results in reducing the inter data center latency $\kappa(\cdot)$ by $\approx 38\%$ when compared to Spectral with $\mathbb{W}_3 = \{1,1,100,1\}$.

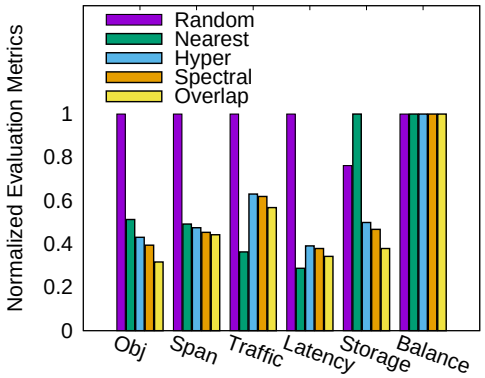

Figure 6: Overlap results in reducing the storage cost $\mathcal{S}(\cdot)$ by $\approx 24\%$ when compared to Spectral with $\mathbb{W}_4 = \{1, 1, 1, 100\}$.

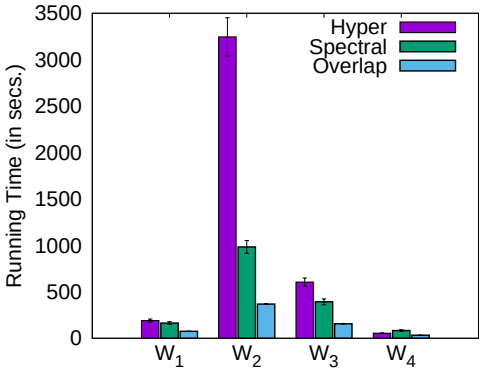

Figure 7: Comparing the execution times of the proposed overlapping clustering algorithm with spectral clustering(Atrey et al., 2018) and hypergraph partitioning algorithm(Yu and Pan, 2017).

rithm is *efficient, scalable, and effective*. Additionally, since there is just one algorithm (unlike previous works) that can jointly solve the data and replica placement problem, it offers a better and unified system design. Further, the capability to adapt to the change in weight vector settings $\mathbb{W}$ facilitates handling of a variety of real-world scenarios as described by different weight vectors.

6 CONCLUSIONS

In this paper, we addressed the problem of *combined data and replica placement* of data-intensive services into geo-distributed clouds. Although replication is an integral part of data placement, we identified that instead of posing it as a joint optimization problem, most of the techniques in the literature have treated them as independent problems, and have employed a two-phase approach: performing data placement followed by replication. Consequently, a unified paradigm, CPR, capable of *combining data placement and replication* was devised, thereby enabling the two problems to be studied in unison. Specifically, the proposed overlapping correlation clustering algorithm on hypergraphs with its ability to partition the set of data-items by assigning a data-item to multiple data centers facilitated this joint optimization. Experiments on a real-world trace-based social network dataset portrayed the effectiveness, efficiency, and scalability of the proposed algorithm.

Table 2: Quantifying the performance of the considered baselines relative to the proposed overlapping correlation clustering algorithm on the evaluation metrics.

Algorithm	Degradation in performance of Baselines relative to Overlap				
	Span	Traffic	Latency	Storage	Obj
Random	377.78%	255.24%	510.87%	100.87%	274.68%
Nearest	106.29%	67.70%	160.50%	163.64%	139.69%
Hyper	50.91%	35.78%	51.85%	31.67%	37.98%
Spectral	35.26%	25.65%	37.69%	23.28%	24.66%

Currently, the proposed algorithm learns a data and replica placement strategy from a historical snapshot of the social network trace. In the future, the focus would be to make CPR adaptive for managing updates in the data (including changes to the data request patterns) in an online manner, and dynamically updating the placement output. Additionally, the aim is to generalize the notion of CPR to broader and generic classes of the data placement problem.

ACKNOWLEDGMENTS

This research is partly funded by VLAIO, under grant number 140055 (SBO Decomads).

REFERENCES

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

(2017b). Latency Between AWS Global Regions. http://zhiguang.me/2016/05/10/latency-between-aws-global-regions/.

(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/.

(2018). Cisco Visual Networking Index: Forecast and Trends (2017–2022). https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.

(2018). HDFS Architecture Guide. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#References.

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*.

Atrey, A., van Seghbroeck, G., Volckaert, B., and Turck, F. D. (2018). Scalable data placement of data-intensive services in geo-distributed clouds. In *CLOSER*, pages 497–508.

Bonchi, F., Gionis, A., and Ukkonen, A. (2013). Overlapping correlation clustering. *Knowl. Inf. Syst.*, 35(1):1–32.

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*.

Ding, Y. and Lu, Y. (2009). Automatic data placement and replication in grids. In *HiPC*, pages 30–39.

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.

Ferdaus, M. H., Murshed, M., Calheiros, R. N., and Buyya, R. (2017). An algorithm for network and data-aware placement of multi-tier applications in cloud data centers. *JNCA*, 98:65 – 83.

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.

Grace, R. K. and Manimegalai, R. (2014). Dynamic replica placement and selection strategies in data grids— A comprehensive survey. *JPDC*, 74(2):2099 – 2108.

Guo, W. and Wang, X. (2013). A data placement strategy based on genetic algorithm in cloud computing platform. In *WISA*, pages 369–372.

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.

Kosar, T. and Livny, M. (2004). Stork: making data placement a first class citizen in the grid. In *ICDCS*, pages 342–349.

Kosar, T. and Livny, M. (2005). A framework for reliable and efficient data placement in distributed computing systems. *JPDC*, 65(10):1146–1157.

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.

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.

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

Shabeera, T., Kumar, S. M., Salam, S. M., and Krishnan, K. M. (2017). Optimizing vm allocation and data placement for data-intensive applications in cloud using aco metaheuristic algorithm. *IJEST*, 20(2):616 – 628.

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

White, T. (2012). *Hadoop: The Definitive Guide*. O'Reilly Media, Inc.

Yu, B. and Pan, J. (2015). Location-aware associated data placement for geo-distributed data-intensive applications. In *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.

Yu, T., Qiu, J., Reinwald, B., Zhi, L., Wang, Q., and Wang, N. (2012). Intelligent database placement in cloud environment. In *ICWS*, pages 544–551.

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

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., and Stoica, I. (2016). Apache spark: A unified engine for big data processing. *CACM*, 59(11):56–65.

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.

Zhao, Q., Xiong, C., and Wang, P. (2016a). Heuristic data placement for data-intensive applications in heterogeneous cloud. *Journal of Electrical and Computer Engineering*, 2016:1–8.

Zhao, Q., Xiong, C., Zhang, K., Yue, Y., and Yang, J. (2016b). A data placement algorithm for data intensive applications in cloud. *JGDC*, 9(2):145–156.