# GeoLayer: Towards Low-Latency and Cost-Efficient Geo-Distributed Graph Stores with Layered Graph

Feng Yao, Xiaokang Yang, Shufeng Gong, Song Yu, Yanfeng Zhang, Ge Yu

*Northeastern University*, Shenyang, China

{yaofeng, yangxk, yusong}@stumail.neu.edu.cn, {gongsf, zhangyf, yuge}@mail.neu.edu.cn

*Abstract*—The inherent connectivity and dependency of graph-structured data, combined with its unique topology-driven access patterns, pose fundamental challenges to conventional data replication and request routing strategies in geo-distributed cloud storage systems. In this paper, we propose GeoLayer, a geo-distributed graph storage framework that jointly optimizes graph replica placement and pattern request routing. We first construct a latency-aware layered graph architecture that decomposes the graph topology into multiple layers, aiming to reduce the decision space and computational complexity of the optimization problem, while mitigating the impact of network heterogeneity in geo-distributed environments. Building on the layered graph, we introduce an overlap-centric replica placement scheme to accommodate the diversity of graph pattern accesses, along with a directed heat diffusion model that captures heat conduction and superposition effects to guide data allocation. For request routing, we develop a stepwise layered routing strategy that performs progressive expansion over the layered graph to efficiently retrieve the required data. Experimental results show that, compared to state-of-the-art replica placement and routing schemes, GeoLayer achieves a $1.34\times$–$3.67\times$ improvement in response times for online graph pattern requests and a $1.28\times$–$3.56\times$ speedup in offline graph analysis performance.

*Index Terms*—Geo-distributed cloud storage, graph pattern request, replica placement, request routing

## I. Introduction

Graph-structured data is widely used across various domains such as social networks [2], intelligent transportation systems [3], financial transaction networks [4], and logistics systems [5]. In many practical deployments, such data is often generated at geographically dispersed endpoints. To deliver low-latency access to nearby clients, large organizations strategically deploy their applications across multiple *geo-distributed* data centers (DCs). For example, Meta operates more than 20 DCs globally to store and manage its large-scale social networking services across countries [6].

An example is the infrastructure of global payment platforms [7], [8], which supports a range of regional and global financial business scenarios: (1) Transaction graphs and account relationship networks [9] continuously generated within specific regions are maintained by local DCs to ensure low-latency processing of regional operations such as fund transfers and credit-risk assessment [10], [11]. (2) These platforms also support cross-border transactions and provide access to globally distributed financial services [12], [13], ensuring secure operations by tracking and monitoring transaction paths across regions. (3) At the global scale, these platforms conduct graph analysis to deliver personalized financial services [14].
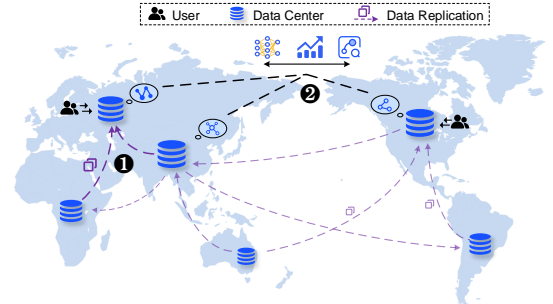


Fig. 1. Geo-distributed graph data storage services.

Geo-distributed storage services serve as the foundational infrastructure for such scenarios, offering two key capabilities: low-latency data access enabled by geo-optimized replica placement, and coordinated routing of data resources across multiple DCs. Specifically, in cross-border transactions, the platform must perform real-time access to remote transaction graph patterns from the initiator to detect potential fraudulent activities [15]. For example, Ant Group scans approximately 400 million potentially risky transaction patterns per day, with each risk control decision constrained to within 500 milliseconds [16]. However, geo-distributed DCs are connected via Wide Area Networks (WANs), where limited bandwidth often leads to significant latency for remote access. To mitigate this, the storage service reduces access latency by either replicating pattern-related data locally or routing data requests through the nearest DCs, as shown in Figure 1 ❶. In addition, Figure 1 ❷ illustrates a global-scale analytical workflow, in which the required graph data is retrieved and routed to three geo-distributed DCs. Enhancing service availability and reliability fundamentally relies on effective replica placement and request routing, yet these problems are non-trivial.

Service providers require that applications satisfy certain latency constraints while minimizing monetary costs, including both data transfer and storage expenses. However, the limited and costly bandwidth of WANs introduces a critical trade-off: replicating data across multiple DCs can significantly reduce remote access latency, but it also introduces additional inter-DC synchronization traffic to maintain consistency among replicas and increased storage costs. Furthermore, when a request spans multiple remote data items or involves global analysis, the presence of multiple replicas across DCs introduces a routing decision problem, as any of the available replicas may serve the request [17]. An effective routing strategy should minimize remote data access latency by selecting the optimal replica set, while simultaneously reducing inter-DC transfer

costs by consolidating data access across fewer locations.

Most existing solutions for replica placement and request routing either focus solely on optimizing the distance between users and data items while overlooking the associations among data units within a request [18]–[24], or rely on job-specific optimizations based on predefined access patterns [17], [25]–[29]. This *job-driven* [29], [30] paradigm inherently assumes fixed and homogeneous data access behaviors, making it feasible to co-locate replicas associated with the same access pattern based on simple statistical analysis of historical access traces. In contrast, associations among data units in graph-structured data are intrinsic to the graph topology. Graph workloads are typically accessed via structural traversals, which are fundamentally *data-driven*, resulting in highly diverse and dynamic associations among data items. This fundamental shift in access characteristics significantly complicates the problem space and renders existing approaches ineffective.

**Challenges.** We present a detailed summary of the unique challenges introduced by graph-structured data.

*Graphs exhibit strong connectivity and dependency relationships.* This implies that a single request may access a large volume of associated data, often triggering substantial cross-DC traffic. As a result, the replica placement strategy must carefully account for the topological co-location of associated data items, substantially increasing the complexity of both replica placement and associated data routing optimization.

*Graph data access exhibits substantial pattern diversity.* Queries often attach variables on vertex and edge attributes to different parts of the query pattern [31]. In real-world financial systems, such as Ant Group's Alipay platform [32], each transaction records dozens of attributes that are directly relevant to queries [33], [34] (*e.g.,* counterparty information, amount, timestamp, payment method, risk score). These attributes collectively define rich graph patterns that closely reflect real query workloads. Furthermore, the variability of source and destination vertices specified by different queries further amplifies the diversity of access patterns and the unpredictability of access behaviors. In addition, data accesses frequently overlap across different patterns, *i.e.,* the same data appears in multiple patterns. Such overlap introduces significant intra-pattern variation in access frequency, further exacerbating the limitations of traditional replica placement strategies that operate at the granularity of entire patterns.

*Geo-distributed DCs typically exhibit heterogeneous network bandwidth* [35], [36], which leads to significant latency variation when accessing associated data items involved in a graph pattern across multiple DCs. This network asymmetry often turns the slowest-responding DC (*i.e., straggler*) into the performance bottleneck [37]. As a result, decision-making becomes more complex, as it must jointly account for both data locality and latency heterogeneity.

**GeoLayer.** To tackle these challenges, we propose GeoLayer, a geo-distributed graph storage framework that optimizes replica placement and request routing from a data-driven perspective, leveraging the structural characteristics of the graph to guide decision-making. GeoLayer focuses on optimizing both latency and cost in geo-distributed graph storage services. To this end, 1) **we define system-level cost metrics and formulate a joint optimization problem** that determines both replica placement and request routing. The problem is inherently a multi-objective, multi-constraint combinatorial challenge, characterized by a vast decision space and strict latency requirements. To reduce complexity, 2) **we construct a layered graph architecture** that organizes the graph topology into multiple layers based on ascending request latency. This design decomposes the global optimization problem into a hierarchical tree of subproblems, significantly shrinking the decision space. Meanwhile, it mitigates the impact of network heterogeneity by confining optimization to regions within each layer that exhibit uniform latency. Built on the layered graph, 3) **we propose an overlap-centric replica placement scheme** that performs layer-by-layer data sinking guided by replication gain. Inspired by the principles of heat conduction and superposition in thermodynamics, we introduce a Directed Heat Diffusion model to support data placement at the granularity of overlapping pattern regions. This model quantifies the affinity between placement regions and the associated data items. For request routing, 4) **we design a stepwise layered routing strategy**. In the online mode, routing follows a bottom-up, expanding retrieval process. In contrast, the offline mode consists of a top-down global data localization phase followed by a bottom-up data assembly phase, which progressively refines the data distribution based on communication cost.

In summary, we make the following key contributions.

- We introduce cost metrics and formulate the problem of minimizing cost while satisfying latency constraints (§III).
- We construct a latency-aware layered graph that decomposes the global optimization problem into a hierarchical tree of subproblems, effectively reducing the decision space (§IV).
- We propose an overlap-centric replica placement scheme that guides progressive data sinking within the layered graph, and further introduce a directed heat diffusion model to determine overlapping pattern regions allocation (§V).
- We design a stepwise layered routing approach for both online and offline workloads (§VI).

## II. BACKGROUND

**Graph pattern.** Consider a connected *graph* $G = (V, E)$, where $V$ is a finite set of vertices and $E \subseteq V \times V$ is a set of edges. A graph pattern $p$ is defined as a tuple of path patterns which, when applied to $G$, yields a set of path bindings [31]. A path is an alternating sequence of vertices and edges, $ph = (v_0, e_0, v_1, e_1, \ldots, e_{n-1}, v_n)$, where each $v_i \in V$ and $e_i \in E$. For ease of presentation, we overload the term *pattern* to refer to the actual set of data items matched by a graph query.

**Geo-Distributed Data Center.** We highlight network heterogeneity among geo-distributed DCs. To illustrate this characteristic, we conduct a measurement study across five of Alibaba Cloud's geo-distributed DCs, deploying ecs.c5.xlarge instances in North America (US East–Virginia, US West–

| BW / Lat. | US East | US West | London | Singapore | Beijing |
|---|---|---|---|---|---|
| US East | — | 96Mbps | 92Mbps | 66Mbps | 68Mbps |
| US West | 69ms | — | 93Mbps | 80Mbps | 77Mbps |
| London | 80ms | 136ms | — | 74Mbps | 42Mbps |
| Singapore | 225ms | 178ms | 213ms | — | 96Mbps |
| Beijing | 226ms | 145ms | 256ms | 75ms | — |

| Cloud Provider | Storage ($/GB/Month) | GET/PUT ($/M) | Data Transfer ($/GB) |
|---|---|---|---|
| Amazon [38] | 0.023 | 0.40 / 5.00 | 0.050 |
| Google [39] | 0.020 | 0.40 / 5.00 | 0.080 |
| IBM [40] | 0.023 | 0.42 / 5.20 | 0.050 |
| Azure [41] | 0.021 | 0.50 / 6.50 | 0.040 |
| Alibaba [42] | 0.016 | 0.10 / 1.40 | 0.043 |

Silicon Valley), Europe (London, UK), and the Asia-Pacific region (Singapore; Beijing, China). Specifically, we measure the available bandwidth and round-trip time (RTT) latency between instances across DCs, as summarized in Table I. The measured bandwidth ranges from 96.8 Mbps to 42.4 Mbps, while latency varies from 69.1 ms to 256.3 ms. This variability is primarily attributed to differences in network infrastructure, physical distance, and network topology among the DCs.

**Cloud Storage Cost.** Cloud providers offer *disaggregated* storage as a service in geo-distributed DCs. Data is stored in immutable blocks called *objects*, which are distributed and replicated across multiple storage nodes to ensure availability and durability [43]. The relevant costs primarily include storage, data retrieval and modification, and inter-region data transfer. Taking the US East region as an example, Table II presents the pricing details from various providers under a peak-demand scenario. Cloud storage services allow users to retrieve and modify data through PUT and GET operations, with intra-region API calls billed based on the number of requests. However, these services do not provide intelligent replica placement or request routing strategies.

**System Services Overview.** We outline the system storage services responsible for handling user-requested data items involved in queried graph patterns.

*Graph query mode.* Based on real-world business requirements, graph queries can be categorized into online and offline modes [44]. Accordingly, the storage service exhibits significantly different response characteristics to graph pattern requests when providing data access for these two scenarios:

(1) Online requests typically involve both read and write operations over graph patterns. For read requests, query patterns commonly involve graph pattern matching (often join-heavy on many-to-many edges with group-by, aggregation, and sorting operators) and finding paths (*e.g.,* cheapest/shortest or weighted paths) [45], [46]. Queries attach variables about vertex/edge attributes on different parts of the pattern [31]. For example, in a funds-transfer scenario, a read request

retrieves transfers during a given period from a blocked account to an unblocked account, with variables covering the vertex type Account, vertex attribute isBlocked, edge type Transfer, and edge attribute date. For write requests, the pattern performs insert/update/delete on involved vertices, edges, or attributes. For instance, a scheduled transfer inserts a Transfer edge from the sender to the receiver and updates the corresponding account balance within the specified time window. In online mode, graph pattern requests are primarily used in interactive applications that demand fast responses, imposing strict latency requirements. To meet this, an optimized replica placement strategy can provide low-latency access.

(2) Offline requests primarily focus on graph analytics tasks, such as iterative graph computation [47], [48] and graph learning [49], [50]. It is often necessary to correlate data from multiple DCs to derive global insights. For instance, in a cross-border e-commerce scenario, the system extracts and associates user shopping histories from geo-distributed DCs and conducts a joint analysis across them to generate product recommendations. In offline mode, graph data requests typically have no real-time requirements but require coordination of distributed data resources across DCs to support the efficient execution of analytical algorithms, while minimizing data transmission costs.

*Data replication and request routing.* As discussed in the online mode, requests target graph patterns, such as retrieving all transfers within a given time window from blocked to unblocked accounts. Because the source and destination accounts (and their related transactions) may reside in different DCs, low-latency access is achieved by on-demand replicating the pattern-related items to the DC closest to the user and routing the request locally whenever possible. During workload execution, access logs are continuously collected to capture per-item request frequencies, which are then used to guide adaptive replication decisions [27]. In the offline mode, existing replicas remain reusable. For joint analytics, request routing selects an optimal layout over the available replicas, favoring in-place computation and locality-preserving join, to reduce inter-DC data transfer.

## III. FORMULATION AND MODELING

### A. System Model

We first introduce the definition of request latency, then present the GeoLayer's cost metrics.

**Request Latency.** We define request latency as a measure of pattern request execution efficiency, which comprises the RTT latency incurred by TCP connection establishment (denoted by $l^{RTT}$) and the latency of data item transmission [19]. The request latency $l_{yd}^p$ for a request $p$ originating from DC $y$ that is responded to by the DC $d$ is computed as follows:

$$l_{yd}^p = l_{dy}^{RTT} + S_d^p / BW_{dy}, \qquad (1)$$

where $S_d^p$ denotes the size of the data items involved in $p$ that reside at DC $d$, and $BW_{dy}$ represents the available bandwidth between DC $d$ and DC $y$.

Since data item requests within request $p$ may span multiple DCs, we assume that $p$ involves $n$ DCs. The overall request latency $l_y^p$ is defined as the maximum access latency across all participating DCs, *i.e.*, $l_y^p = \max_{1 \le d \le n}\{l_{yd}^p\}$. The DCs involved in serving request $p$ respond in parallel, after which the partial results are assembled into the final response [51]. However, due to network heterogeneity, some workers may experience substantially higher latency than others. The worker with the highest latency, commonly referred to as the *straggler*, becomes the main bottleneck and ultimately determines the overall latency [37].

**Cost Metrics.** We formulate the major decision factors.

*Storage cost.* The storage cost quantifies the expense of storing data items and their replicas across all DCs. It is defined as:

$$C^{(S)} = \sum_{x \in I} s_x \sum_{d \in D} \delta_{xd} \cdot c_d^{store}, \quad (2)$$

where $I$ is the set of data items, and $s_x$ is the size of item $x$. $\delta_{xd}$ is a binary variable that equals 1 if $x$ is stored in DC $d$, and 0 otherwise; $c_d^{store}$ denotes the unit storage cost in DC $d$.

*Request cost.* The request cost accounts for both read (GET) and write (PUT) operations. We denote $R_{xy}$ (*resp.* $W_{xy}$) as the frequency of read (*resp.* write) requests for data item $x$ originating from DC $y$ over a given time window.

For read requests, beyond the GET cost ($c_d^{read}$), accessing non-local data items incurs additional network costs due to cross-DC transfers. The cost of read requests is defined as:

$$C^{(R)} = \sum_{x \in I} \sum_{y \in D} R_{xy} \sum_{d \in D} \sigma_{xyd}\Big(c_d^{read} + \mathbb{I}(y \neq d) \cdot s_x \cdot c_{dy}^{net}\Big), \quad (3)$$

where $\sigma_{xyd}$ indicates whether DC $d$ provides access to data $x$ requested by DC $y$, and $\mathbb{I}(y \neq d)$ is an indicator function that equals 1 for cross-DC requests ($y \neq d$) and 0 otherwise. $c_{dy}^{net}$ denotes the unit data transmission cost from DC $d$ to DC $y$.

The cost of write requests encompasses the expenses incurred from PUT cost and the network costs associated with synchronizing updates across all replicas, which can be quantified as follows:

$$C^{(W)} = \sum_{x \in I} \sum_{y \in D} W_{xy}\left[c_y^{write} + \sum_{\substack{d \in D \\ d \neq y}} \delta_{xd}\Big(c_d^{write} + s_x \cdot c_{yd}^{net}\Big)\right], \quad (4)$$

where $c_y^{write}$ is the PUT cost of the originating DC and $c_d^{write}$ is the synchronized PUT cost.

*Association penalty cost.* Data items associated with a graph pattern may reside across multiple DCs. In distributed systems, serving such read pattern requests can incur additional system overhead, *i.e.,* coordination costs incurred by distributed routing, which tends to increase with the number of participating DCs [17]. In addition, as previously discussed, the straggler in heterogeneous networks is often the bottleneck for request latency. Considering both factors, GeoLayer incorporates an association penalty cost component to promote the co-location

of associated data items at a minimal number of storage sites, while avoiding routing across highly heterogeneous network paths. The association penalty cost is given by:

$$C^{(A)} = \sum_{p \in P} \sum_{y \in D} R_{py}\left[\lambda_1\Big(\sum_{d \in D} \rho_{pyd} - 1\Big) + \lambda_2 \Delta l_y^p\right],$$

$$\text{where} \quad \Delta l_y^p = \frac{\max\limits_{\{y,d \in D | \rho_{pyd}=1\}} l_{yd}^p - \min\limits_{\{y,d \in D | \rho_{pyd}=1\}} l_{yd}^p}{\min\limits_{\{y,d \in D | \rho_{pyd}=1\}} l_{yd}^p}. \quad (5)$$

Here, $\lambda_1$ and $\lambda_2$ are penalty coefficients, set to 0.5 in our experiments. $\rho_{pyd}$ indicates whether DC $d$ is used to serve at least one data item in the pattern $p$ originated by DC $y$. Ideally, if all data items in a pattern are located within one DC, *i.e.*, $\sum_{d \in D} \rho_{pyd} = 1$, no penalty is incurred. Otherwise, the penalty grows with the number of participating DCs involved in serving the pattern. Additionally, the maximum latency between DCs involved in pattern $p$ (*i.e.,* straggler) should not be excessively disparate. Equation (5) reflects GeoLayer's preference for data locality access and latency uniformity.

### B. Optimization Problem

GeoLayer's objective is to minimize the overall cost by determining 1) the placement of data items involved in patterns, *i.e.*, $\delta_{xd}$, and 2) the routing of pattern read requests, *i.e.*, $\sigma_{xyd}$ and $\rho_{pyd}$, while satisfying all constraints:

$$\min_{\delta_{xd}, \ \sigma_{xyd}, \ \rho_{pyd}} \Big(C^{(S)} + C^{(R)} + C^{(W)} + C^{(A)}\Big), \quad (6)$$

s.t. (a) $\sigma_{xyd} \le \delta_{xd}, \ \sum_{d \in D} \sigma_{xyd} = 1,$

(b) $\rho_{pyd} \le \delta_{x^* d}, \ \forall x^* \in I_p,$

(c) $\dfrac{1}{|I|} \sum_{x \in I} \sum_{y \in D} R_{xy} \sum_{d \in D} \sigma_{xyd} \cdot l_{yd}^x \le \Gamma_{\max},$

(d) $\max\limits_{\{y,d \in D | \rho_{pyd}=1\}} l_{yd}^p \le \eta_p \cdot \Gamma_{\max},$

(e) $\delta_{xd}, \ \sigma_{xyd}, \ \rho_{pyd} \in \{0,1\}, \ \forall x \in I, \forall y, d \in D, \forall p \in P.$

Here, constraint (a) ensures that each data item is routed exclusively to a DC that holds its replica, and that each request for a data item is served by a single DC. Constraint (b) ensures that read requests for data items in pattern $p$ are routed only to the DCs storing the corresponding replicas, where $I_p \subseteq I$ denotes the set of data items associated with pattern $p$. Moreover, GeoLayer emphasizes adherence to latency Service Level Objectives (SLOs). Let $\Gamma_{\max}$ denote the upper bound on acceptable latency. Constraint (c) enforces that the average response latency of read requests does not exceed $\Gamma_{\max}$. Constraint (d) further stipulates that the maximum latency for each request pattern $p$ must remain below $\eta_p \cdot \Gamma_{\max}$, where $\eta_p \in (0,1]$ is a proportionality coefficient that reflects the varying latency requirements of different query patterns. For instance, fraud detection in financial transactions often demands rapid responses (*e.g.,* $\le$ 500ms), whereas logistics delivery path queries can tolerate more relaxed thresholds [52].

The optimization problem can be formulated as a binary integer programming (BIP) model, which has been proven to be NP-hard [53]. The decision variables in this model are tightly interdependent. Replica placement directly constrains the feasible space for routing decisions, requiring the co-location of associated data items involved in the graph pattern and necessitating dynamic trade-offs between request latency and cross-DC synchronization overhead. Simultaneously, routing decisions must jointly consider data locality, based on current replica distribution, and end-to-end request latency.

## IV. LATENCY-AWARE LAYERED GRAPH

**Overview.** The optimization problem is a multi-objective, multi-constraint combinatorial task for which direct global optimization is computationally intractable. We address this by constructing a latency-aware layered graph that decomposes the global problem into a hierarchical tree of subproblems. Patterns involving a large number of *cross-partition edges* inherently trigger inter-DC data accesses [54], with request latencies varying significantly due to network heterogeneity. We organize these cross-partition edges into multiple layers based on ascending request latency. By construction, inter-layer latencies progress monotonically. Within each layer, a set of edges with similar response latencies groups the connected lower-layer graph regions into latency-homogeneous clusters. The connections among these layer-wise edge sets and clusters form a hierarchical tree. Restricting placement and routing decisions to the branches of this tree substantially reduces the decision space, mitigates variable coupling, and provides a unified, stable structure for subsequent overlap-centric replica placement (§V) and stepwise layered routing (§VI).

**Layered Graph Construction.** Given a *connected graph* $G = (V, E)$ and its *geo-distributed partitioning*, the layered graph is constructed by combining three components: (1) the base layer, denoted as $Layer_0$, which consists of per-DC subgraphs $\{G_d = (V_d, E_d)\}_{d \in D}$; (2) upper-layer bridge graphs ($Layer_i, 1 \leq i \leq h$), denoted as $Layer_i : G_i^B = (V_i^B, E_i^B)$ which capture inter-partition connectivity at different latency levels; and (3) the inter-layer links that connect bridge graphs across adjacent layers.

*Underlying graph ($Layer_0$).* In geo-distributed environments, each DC independently stores and maintains a local subgraph. This design isolates locally available data without requiring WAN access and constitutes the base layer ($Layer_0$) of the layered graph, representing the local substructures of $G$ maintained by individual DCs. Formally, the base layer $Layer_0$ consists of $|D|$ disjoint subgraphs $\{G_d = (V_d, E_d)\}_{d \in D}$, obtained by geo-distributed partitioning of $G$. Each $G_d$ is stored at DC $d$ and satisfy: (i) $\bigcup_{d \in D} V_d = V$; (ii) $V_d \cap V_{d'} = \emptyset$ for $\forall d \neq d'$; (iii) $E_d = E \cap (V_d \times V_d)$.

*Upper-layer bridge graph ($Layer_i, 1 \leq i \leq h$).* The *bridge graph* is constructed from cross-partition edges that connect boundary vertices across different DC subgraphs $G_d$, enabling data access across DCs. Considering the latency heterogeneity

of WAN links between DCs, these cross-partition edges are assigned to different latency layers. This organization forms locally uniform-latency regions within each layer.

**Definition 1** (Bridge Graph). Let each subgraph $G_d = (V_d, E_d)$ have a boundary vertex set $B_d = \{ u \in V_d \mid \exists\, v \in V \setminus V_d, (u, v) \in E \}$. Define the global boundary vertex set $V^B = \bigcup_{d \in D} B_d$, and the set of cross-partition edges $E^B = \{(u, v) \in E \mid u \in B_d, v \in B_{d'}, d \neq d'\}$. Then, the bridge graph is defined as $G^B = (V^B, E^B)$.

Considering the network heterogeneity among DCs, we introduce a latency assignment function $\delta : E^B \to \mathbb{R}^+$ that maps each edge in the bridge graph to inter-DC request latency. For any $e = (u, v) \in E^B$, we set $\delta(e) = l_{dd'}^e$, where $l_{dd'}^e$ denotes the observed latency between DCs $d$ and DC $d'$.

We then partition the bridge graph into $h$ latency-based layers to cope with network heterogeneity. First, define a sequence of latency thresholds $0 = t_0 < t_1 < \cdots < t_{h-1} < t_h = +\infty$. Next, introduce the latency-partition function $f : E^B \longrightarrow \{1, 2, \ldots, h\}$, where $e \in E^B$, $f(e) = i \Leftrightarrow \delta(e) \in [t_{i-1}, t_i)$. In other words, each edge whose measured latency falls within $[t_{i-1}, t_i)$ is assigned to $Layer_i$. We then project the global bridge graph $G^B$ onto each layer: $E_i^B = \{e \in E^B \mid f(e) = i\}$, $V_i^B = \bigcup_{e \in E_i^B} \{u, v \mid (u, v) \in e\} \subseteq V^B$. Thus, the bridge graph at latency level $i$ is $G_i^B = (V_i^B, E_i^B)$.

To link layers, we examine how edges in the bridge graph affect connectivity in lower-layer regions and introduce the concept of a *bridge subgraph* to group intra-layer edge subsets.

**Definition 2** (Bridge Subgraph). For a given layer $Layer_i$ with $i > 0$, let $\mathcal{G}_{i-1} = (V, E_{prev})$, where $E_{prev} = \bigcup_{d \in D} E_d \cup \bigcup_{j=1}^{i-1} E_j^B$ represents the aggregated graph composed of the underlying graph ($Layer_0$) and all bridge graphs from $Layer_1$ to $Layer_{i-1}$, and let the weakly connected components of $\mathcal{G}_{i-1}$ be denoted as $WCC = \{cc_1, \ldots, cc_n\}$. A subgraph $G^* = (V^*, E^*)$ with $E^* \subseteq E_i^B$, $V^* = \bigcup_{(u,v) \in E^*} \{u, v\}$, is called a *bridge subgraph* if adding $G^*$ to $\mathcal{G}_{i-1}$ merges a subset of components $\{cc_1', \ldots, cc_m'\} \subseteq WCC$ into a single connected component $\hat{cc} = \left( \bigcup_{j=1}^m cc_j' \right) \cup G^*$. The bridge subgraphs from $Layer_{i-1}$ that are contained within $\hat{cc}$ collectively form a *cluster* linked by $G^*$.

Notably, the DCs in $Layer_0$ are grouped into clusters by the bridge subgraphs in $Layer_1$. Moreover, within each cluster, network resources are more evenly distributed than across disjoint regions at the same layer.

**Example 1:** The right side of Figure 2 depicts a geo-distributed example graph with 18 vertices spread across six DCs, $\{DC_1, \ldots, DC_6\}$. For clarity, inter-DC link latencies are abstracted into three levels, *i.e.*, low, medium, and high, and each cross-partition edge is assigned a latency level via the function $f(e)$. The left side of Figure 2 shows the corresponding layered graph. Local subgraphs maintained by each DC form $Layer_0$, while cross-partition edges are placed in $Layer_1$, $Layer_2$, and $Layer_3$ according to their latency levels. These edges are then grouped into *bridge subgraphs*,
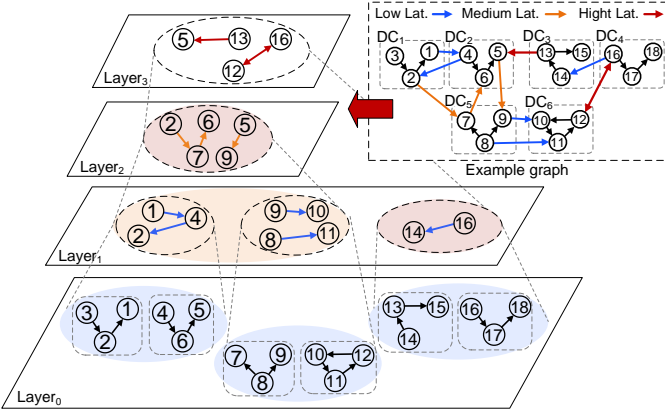
Fig. 2. An example of constructing a layered graph based on latency levels.



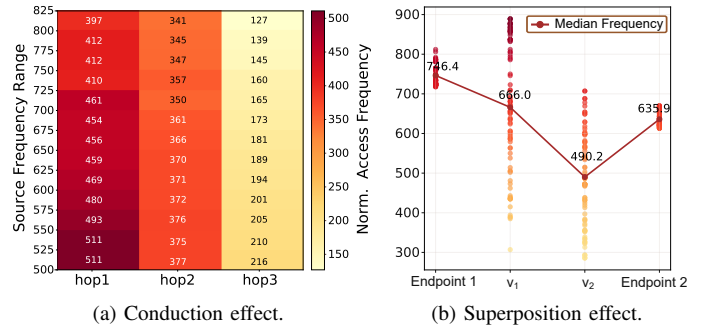(a) Conduction effect.     (b) Superposition effect.

Fig. 3. The conduction and superposition effects of access frequency along paths in pattern access. The color intensity represents the normalized access frequency of vertices, with darker colors indicating higher frequency.

each of which merges a set of previously disjoint connected components. For example, in $Layer_1$, edges $(1, 4)$ and $(4, 2)$ connect the subgraphs at $DC_1$ and $DC_2$, forming a bridge subgraph and thereby clustering $DC_1$ and $DC_2$. Similarly, the bridge subgraphs in $Layer_2$ connect the clusters induced by the yellow-shaded bridge subgraphs in $Layer_1$. □

## V. OVERLAP-CENTRIC REPLICA PLACEMENT

### A. Design Overview

Building on the layered graph, our replica placement scheme localizes decision-making within clusters at each layer, operating at the pattern/overlap-region granularity. The process begins with a sinking step that assigns each pattern to the latency-compliant layer. To address the variation in access frequencies across overlapping regions of patterns, we first apply a gain function to determine whether a pattern should undergo full intra-cluster replication or decomposition. For decomposable patterns, we split them along their overlapping regions and let these region-segments competitively bid for placement across the cluster's bridge subgraphs. To quantify the affinity between each bridge subgraph and these segments, we introduce a directed heat diffusion model, which guides the placement of pattern segments within each cluster. This process cascades recursively through the layers until replicas are placed in individual DCs. By decomposing patterns at the overlap-region granularity, we weaken co-location coupling among pattern-related data and restrict placement decisions to latency-homogeneous clusters. Consequently, routing cost depends only on whether a cluster is covered, rendering the objective cluster-separable and enabling a place-first, then best-response routing (§VI). Due to space constraints, we defer the full implementation details to [1].

### B. Replica Placement Built on Layered Graph

**Sinking of Pattern Replicas.** Graph pattern requests impose varying latency requirements, as reflected in constraints (d) of Equation (6). Because the layered graph is organized by increasing request latency, whenever a pattern's latency bound cannot be met at layer $L_k$, a complete replica of that pattern must be sunk to every bridge subgraph in the corresponding clusters of $L_{k-1}$. In other words, the components of the pattern cannot be split across different bridge subgraph regions in

layer $L_{k-1}$, as such dispersed placement would fail to meet the latency requirements for remote access between components within the pattern. This sinking process continues until the pattern reaches a layer that satisfies its latency requirements. Once a layer does meet the pattern's latency constraint, GeoLayer allows the pattern's data items to be decomposed and distributed across the appropriate regions in all lower layers, enabling finer-grained cost optimization. At that point, the pattern is placed at this layer in preparation for subsequent replica placement optimization. The mechanism of sinking replicas enforces strict latency guarantees while enabling a trade-off between the cost of remote access and the benefits of local caching.

**Directed Heat Diffusion Model.** Before presenting the replica placement optimization, we first introduce the *Directed Heat Diffusion* (DHD) model. This model captures how graph pattern access behaviors manifest over the graph topology, quantifying which regions are most likely to be involved in complete pattern requests and thus warrant co-location. These predictions provide a key foundation for guiding fine-grained replica placement decisions over overlapping pattern regions.

*Insight.* A graph pattern consists of a set of path bindings whose elements exhibit strong connectivity and dependency. To examine how these structural characteristics affect access behavior, we randomly issued 1,000 three-hop path queries on the LDBC dataset [55] and recorded each involved vertex's access frequency. Figure 3a presents a heatmap illustrating the median access frequencies of vertices at hops 1 to 3, grouped by the access frequency range of their corresponding source vertices (y-axis). The results reveal a clear *conduction effect*: median access frequency declines steadily as path length increases. Furthermore, vertices adjacent to high-frequency vertices themselves tend to be accessed more often. Interestingly, certain source vertices (*e.g.,* in $[500, 550]$) exhibit stronger conduction effects; we attribute this to *edge transition probabilities*, whereby a larger neighbor set dilutes the probability of traversing any individual outgoing edge. Figure 3b extends this analysis to paths anchored by high-frequency endpoints and shows that intermediate vertices maintain even higher median access frequencies compared to those in Figure 3a. This finding suggests that the conduction effect accumulates across hops, exhibiting a superposition behavior.

Intuitively, in real-world graphs, core regions (*e.g.,* celebri-

ties in social networks) attract high request rates, and their influence propagates along edges to neighboring regions, elevating neighbors' access frequencies. Overlapping influence from multiple cores superimposes at common neighbors, making their accesses even more frequent. This motivates a heat view of access: vertices act as heat sources, edges conduct heat, and the heat field dissipates unless reheated by new requests.

*Design of the model.* Inspired by thermodynamic principles of heat conduction and superposition, we propose the Directed Heat Diffusion model, which conceptualizes vertices as thermal masses capable of absorbing and diffusing heat.

We first interpret the access frequency $R_{x\cdot}$ ($R_{x\cdot} > 0$) of each data item $x$ (including vertices and edges) as its initial heat in the system, so that $\mathcal{H}_x^0 = R_{x\cdot}$.

At each diffusion step $k$, the heat transferred from vertex $u$ to vertex $v$ is:

$$\Delta\mathcal{H}_{uv}^k = \alpha \cdot \frac{A_{uv}}{|\mathcal{N}_u^{out}|} \cdot \mathsf{ReLU}(\mathcal{H}_u^k - \mathcal{H}_v^k), \qquad (7)$$

where $A_{uv}$ in the adjacency matrix $A$ represents the initial heat value of the *undirected* edge $(u, v)$. The parameter $\alpha$ controls heat diffusivity, and the product $\alpha \cdot A_{uv}$ determines the diffusion intensity. The ReLU operation suppresses negative values, ensuring that heat flows only from higher to lower temperature vertices. Note that heat diffusion does not strictly follow the direction of directed edges, as pattern traversal may also involve moving towards in-neighbors [31]. The term $|\mathcal{N}_u^{out}|$ denotes the number of lower-heat neighbors.

The heat update for a vertex $v$ can be expressed as:

$$\mathcal{H}_v^{k+1} = (1-\gamma)\left[\mathcal{H}_v^k + \sum_{u\in\mathcal{N}_v^{in}}\Delta\mathcal{H}_{uv}^k - \sum_{w\in\mathcal{N}_v^{out}}\Delta\mathcal{H}_{vw}^k\right]. \tag{8}$$

Here, $\gamma$ is the global decay rate that models natural heat dissipation and captures the time-decaying nature of data access frequency due to information obsolescence [24], [29]. The sets $\mathcal{N}_v^{in}$ and $\mathcal{N}_v^{out}$ denote the neighbors from which heat flows into and out of vertex $v$, respectively. Intuitively, heat accumulates from any higher-heat neighbor, while its heat propagates to its lower-heat neighbors.

Additionally, certain regions $\mathcal{O}$ of the graph act as heat sources (*e.g.,* active users who frequently post content in social networks), continually supplying heat to the system. We model their contribution as source dynamics, defined as follows:

$$Q_v^{k+1} = \begin{cases} Q_v^0 e^{-\pi k} + \Delta Q \cdot \sum\sigma_{v\cdot\cdot}, & \text{if } v \in \mathcal{O}, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

Here, $e^{-\pi k}$ implements exponential decay with rate $\pi = \ln(2)/T_{hl}$ determined by the half-life $T_{hl}$. We initialize each source's heat uniformly as $Q_v^0 = 1/|\mathcal{O}|$. The term $\Delta Q \cdot \sum\sigma_{v\cdot\cdot}$ captures additional heat proportional to the frequency of accesses to vertex $v$.

Then the system heat evolves according to

$$\mathcal{H}^{k+1} = (1-\gamma)\left[\mathcal{H}^k + \alpha L_{dir}^k\mathcal{H}^k\right] + \beta Q^k, \tag{10}$$

where $L_{dir}$ is the directional Laplace matrix encoding both the magnitude and direction of heat flow, defined as:

$$(L_{dir})_{vw} = \begin{cases} -\dfrac{A_{vw}}{|\mathcal{N}_v^{out}|}, & \text{if } \mathcal{H}_v > \mathcal{H}_w, \\[2mm] \dfrac{A_{wv}}{|\mathcal{N}_w^{out}|}, & \text{if } \mathcal{H}_w > \mathcal{H}_v, \\[2mm] 0, & \text{otherwise.} \end{cases} \tag{11}$$

Finally, at equilibrium, $\mathcal{H}^{k+1} = \mathcal{H}^k = \mathcal{H}^*$, and substituting into the update Equation (10) yields the steady-state condition

$$\gamma \cdot \mathcal{H}^* - \alpha(1-\gamma)\cdot L_{dir}^*\mathcal{H}^* = \beta Q^*. \tag{12}$$

Regarding model parameter settings, a large $\alpha$ leads to an overly smoothed heat distribution, whereas a smaller $\alpha$ weakens propagation and slows convergence to a stable state. Therefore, a moderate value of $\alpha$ is preferred. For the heat decay rate $\gamma$, selecting a relatively small value preserves vertex heat memory and ensures that the system remains sufficiently active. Finally, since the heat system is influenced by both external injection and internal diffusion, a moderate value of $\beta$ helps prevent external factors from becoming overly dominant. In our implementation, the parameters are fixed to $\alpha = 0.5$, $\gamma = 0.1$, and $\beta = 0.3$.

The convergence of the DHD model is formally established, and the detailed proof is deferred to [1] due to the page limit.

**Overlap-Centric Replica Placement.** Once the sinking process is complete, the pattern is assessed within the bridge subgraphs ($BS$s) of layer $L_k$ using a data replication gain metric. This metric guides the decision to either fully replicate the pattern within each $BS$ or to decompose it across the $BS$s in the linked cluster of the layer $L_{k-1}$ for cost minimization.

*Data replication gain.* Given that pattern $p$ sinks into $BS_y$ at layer $L_k$ and data item $x \in I_p$. Suppose the cluster $\mathcal{C}$ in layer $L_{k-1}$ linked by $BS_y$ comprises the $BS$s $\{BS_1, \ldots, BS_n\}$ that request $p$. We define the gain of replicating $p$ in $BS_y$ as:

$$C_{py}^{rep} = \sum_{x\in I_p}\left(\Delta C_{xy}^{(R)} + \Delta\hat{C}_{xy}^{(A)} - \Delta C_{xy}^{(S)} - \Delta C_{xy}^{(W)}\right),$$

$$\text{where} \quad \Delta\hat{C}_{xy}^{(A)} = R_{xy}\cdot\left(\lambda_1\sum_{j=1,BS_j\in\mathcal{C}}^{n}\sigma_{xyj}\right). \tag{13}$$

Intuitively, fully replicating $p$ across the $BS$s within a cluster reduces the cost of remote reads and the association penalty, but increases storage and synchronization costs. The gain is primarily driven by the access frequency $R_{xy}$. For $\Delta\hat{C}^{(A)}$, the elimination of the second term in Equation (5) is due to the latency uniformity guaranteed within the cluster. If $C_{py}^{rep} > 0$, then $p$ is fully replicated; otherwise, it is decomposed across the corresponding $BS$s.

*Overlap-centric regional competition.* We next describe the placement strategy for decomposed regions of patterns. Due to the diversity of access patterns, overlapping regions frequently arise, where vertices and edges are shared across multiple patterns. For example, Figure 4a shows the maximal

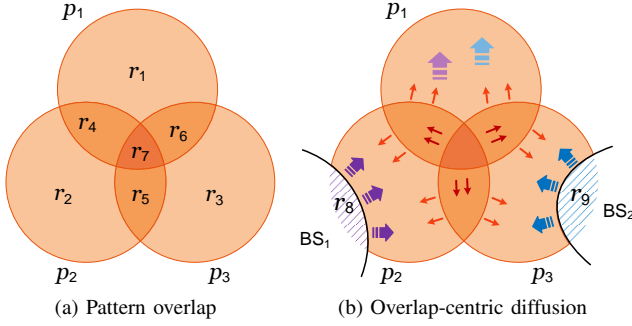(a) Pattern overlap     (b) Overlap-centric diffusion

Fig. 4. Bridge subgraphs in cluster engage in overlap-centric diffusion to compete for regional placement.

overlap scenario among three patterns, forming the 3-Venn diagram [56], from which all other overlap cases can be regarded as subsets. It divides the three patterns into seven disjoint regions $\{r_1, \ldots, r_7\}$, each of which is independently evaluated for placement across the $BS$s.

Overlap regions experience a superposition effect, meaning they incur higher combined access frequencies than non-overlapping areas due to shared requests [57]. This amplification grows with the degree of overlap. Accordingly, we first recompute the replication gain $\Delta C_r^{rep}$ for each overlapping region to decide whether it should be replicated outright or compete for placement among the $BS$s. To arbitrate this competition, we leverage the Directed Heat Diffusion model.

As illustrated in Figure 4b, the shaded regions $r_8$ and $r_9$ represent the overlapping parts between the patterns currently held by bridge subgraphs $BS_1$ and $BS_2$ and the three candidate patterns $p_1$, $p_2$, and $p_3$. We separately combine $r_8$ and $r_9$ with $r_1, \ldots, r_7$ to form two complete heat systems for $BS_1$ and $BS_2$, respectively. Heat updates are then performed according to Equation (8) until the heat originating from $r_8$ and $r_9$ ceases to propagate, which occurs when there are no paths leading to regions with heat levels lower than what $r_8$ and $r_9$ can reach. Finally, the placement of each region is determined by comparing the magnitude of heat diffused from $BS_1$ and $BS_2$. If the patterns co-requested by the $BS$s have no overlap with the patterns they currently hold, or if certain regions are unreachable by the heat diffused from the $BS$s, then placement is determined based on the access frequency of the $BS$s to those patterns or regions.

**Pre-Caching in High-Heat Regions.** As execution proceeds layer by layer, the final placement results are deposited in the underlying DCs. Among these results, certain regions with potentially high influence may not have been captured during the previous historical period.

To proactively guard against sudden surges in access frequency, we use the DHD model to identify and pre-cache these high-heat regions. First, the DC issues a global unobserved edge request to upper layers to acquire complete topological connectivity of externally cached regions. Concurrently, it initializes a high-heat vertex set $\mathcal{O} = \{v \in V \mid \mathcal{H}_v^0 \geq \theta\}$, treating these vertices as external heat sources as defined in Equation (9). The system then iteratively updates heat states using Equation (10) until it reaches steady state (see Equa-

tion (12)). At equilibrium, we extract the high-heat subgraph $G^H = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v \in V \mid \mathcal{H}_v^* \geq \theta\}$ and $\mathcal{E} = \{(u,v) \mid u, v \in \mathcal{V}\}$. This subgraph is subsequently pre-cached in the DC. The guarantee of the lower bound on pre-caching probability is deferred to [1] due to space limitations.

**Online Replica Eviction.** During online query execution, we dynamically evict replicas that become inactive over time, as caching data items that have not been accessed for a long period (*i.e.,* cold data) yields negative returns [24]. To identify cold segments, we model data items as a heat system: each access event creates a heat source at the corresponding item, with intensity proportional to its read frequency. We then iteratively apply the directed heat diffusion update, allowing heat to flow through the cache topology. Items that fail to accumulate sufficient heat are deemed cold and become candidates for eviction. Essentially, a data item's activity level is maintained through both external heat injection and internal heat diffusion, *i.e.,* direct access incidents and indirect correlations with surrounding regions contribute to its activity.

**Update Maintenance.** We adopt a hybrid maintenance strategy that combines incremental updates with periodic refreshes to handle dynamic data changes. The layered graph is a persistent, offline-built structure, with replicas cached within DCs to anticipate future accesses. During periodic maintenance, newly inserted items that have been accessed are captured by access logs, thereby materializing request patterns. These patterns are then fed into the overlap-centric replica placement algorithm, which reacts to these pattern changes and updates replica locations accordingly.

When a vertex or edge is deleted within a DC, maintenance starts at $Layer_0$. We trigger a bottom-up cleanup: the DC handling the delete operation updates its local state, and the removal propagates upward along bridge subgraphs to affected upper-layer clusters until all replicas are removed. In addition, the online replica eviction mechanism gradually evicts replicas as deleted data cools down, assisting the deletion process.

## VI. STEPWISE LAYERED ROUTING

In this section, we present the data request routing strategies for both online and offline query modes.

**Online Query Mode.** Built upon the overlap-centric replica placement scheme over the layered graph, GeoLayer can easily make routing decisions for online query workloads. The core idea is to locate the required data in a layer-by-layer manner: starting from the requesting DC, the search progressively expands to increasingly distant layers until all data matching the requested pattern has been retrieved.

For example, as illustrated in Figure 5, when a pattern request is initiated by a DC in $Layer_0$, the bridge subgraph in $Layer_1$ that links to the requesting DC is responsible for locating the missing data items within its associated cluster in $Layer_0$. The strategy greedily selects the DC that holds the largest subset of requested data items, thereby minimizing the number of DCs involved in the response. If some data
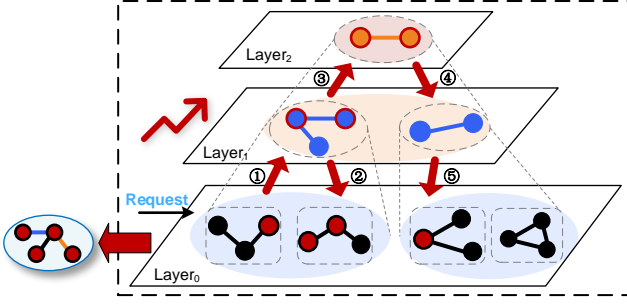
Fig. 5. Request routing under the online query mode.

items remain unresolved, the request is escalated to the bridge subgraph in the next higher layer, $Layer_2$, and the process continues recursively until all required data items are retrieved.

**Offline Query Mode.** The offline query mode typically leverages globally distributed data to support graph analytics. Its objective is to select a cluster of DCs that collectively cover all the required data, enabling cost-efficient execution while ensuring performance guarantees. In contrast to the online routing strategy, the offline mode initiates data requests from the topmost layer to acquire a global view of data distribution. The process comprises two stages: a top-down data localization phase and a bottom-up assembly phase, as illustrated in Figure 6. During the localization phase, bridge subgraphs at each layer identify required data items and progressively map them to candidate DCs in a top-down manner. Redundant cross-DC replicas are not resolved at this stage and are deferred to the assembly phase. Once localization is complete, GeoLayer performs a bottom-up assembly process that progressively selects the most efficient data distribution from replicas at each layer, guided by communication cost.
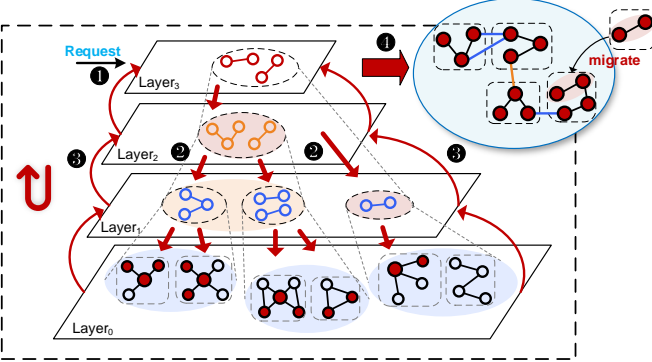


Fig. 6. Request routing under the offline query mode.

In the bottom-up assembly phase, the process begins at $Layer_0$, where each DC evaluates its eligibility for inclusion in the analysis computing cluster. If a DC is deemed unsuitable, the requested data it holds will be migrated. The condition for triggering data migration is defined as:

$$\iota \cdot s^{msg} \cdot (|I_v^{rep}| + |BS|) - \sum_{x \in I^{local}} s_x > (1 - \eta_L) \cdot \xi, \quad (14)$$

Here, $\iota$ denotes the predefined number of algorithm iterations, and $s^{msg}$ represents the size of messages exchanged across DCs. The terms $|I_v^{rep}|$ and $|BS|$ represent the number of replica and boundary vertices, both contributing to inter-DC communication volume. The second term captures the size of the local data that would need to be migrated if the DC is excluded. In our implementation, $\xi$ is set to $20\%$ of the inter-DC messages volume. $\eta_L$ denotes the ratio between the average latency of the current layer and that of the topmost layer. To mitigate the impact of high inter-DC transmission latency on algorithm performance, the migration threshold is progressively relaxed in higher layers. After the migration evaluation in $Layer_0$ is completed, each bridged subgraph in $Layer_1$ redistributes the migrated data by hashing it to the retained DCs within the same cluster. If no DC is retained within a cluster, the bridge subgraph at the upper layer completes the data assembly and subsequently forwards the requested data to the designated destination that minimizes communication cost while balancing the load. This process proceeds layer by layer in an upward manner, progressively expanding its scope to perform cross-DC data assembly, until the global result is assembled at the topmost layer.

## VII. EXPERIMENTAL EVALUATION

### A. Experimental Setup

**Environments.** We conduct experiments on a geo-distributed cluster deployed on Alibaba Cloud. The cluster spans eight geo-distributed data centers located in Beijing (China), Seoul (South Korea), Singapore, Sydney (Australia), Frankfurt (Germany), London (UK), Virginia (USA), and Mexico City (Mexico), with each site hosting an `ecs.c5.xlarge`. On this cluster, we construct the layered graph using a fixed latency interval of 100 ms (i.e., $[0, 100), [100, 200), \ldots$). Additionally, we use the publicly advertised prices for storage, GET/PUT APIs, and data traffic on Alibaba Cloud [42].

**Datasets.** Our evaluation employs three distinct datasets: a synthetic social network generated using the LDBC-SNB benchmark with a scale factor (SF) of 10, and two real-world graph datasets, UK-2005 [58] and Twitter-2010 [59]. Table III provides a detailed summary of the statistics for each dataset.

**Workloads.** We evaluate GeoLayer using two types of workloads. (i) Online graph pattern request: We use a subset of the LDBC interactive workload [61], which defines operational queries over a social network. Specifically, we include six interactive queries `IS-1`, `IS-2`, `IS-3`, `IC-1`, `IC-2`, and `IC-3`. In addition, we generate 3-hop random walk patterns on the UK and TW graphs. We extend the graph query capabilities of RAGraph [47] by incorporating parallel Gremlin queries [62], building a geo-distributed testing platform for pattern requests. We randomly select source vertices from different DCs and issue 1,000,000 queries to construct historical patterns with diverse latency characteristics. An
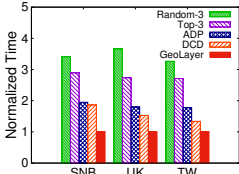
### TABLE III
### DATASET DESCRIPTION

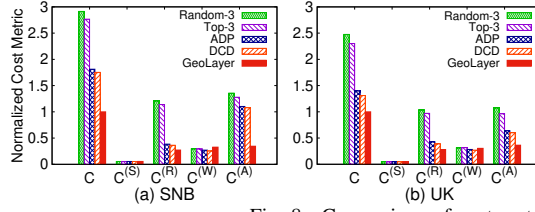| Graph | Vertices | Edges | Abbreviation |
|-------|----------|-------|--------------|
| LDBC-SNB-SF10 [60] | 36,423,181 | 231,371,359 | SNB |
| UK-2005 [58] | 39,459,925 | 936,364,282 | UK |
| Twitter-2010 [59] | 41,652,230 | 1,468,364,884 | TW |

9

Fig. 7. Request latency.
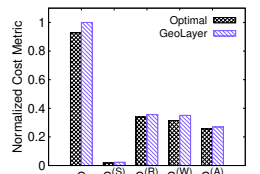

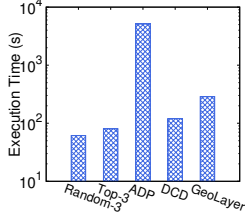Fig. 8. Comparison of cost metrics.
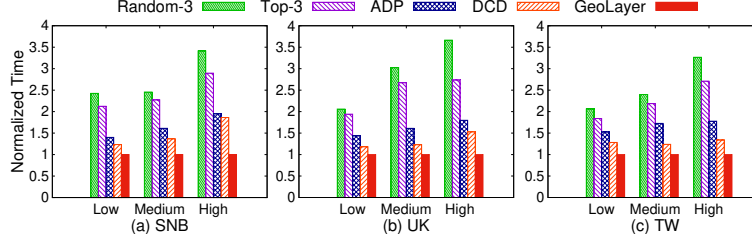

Fig. 9. Optimality gap.


Fig. 10. Execution time.
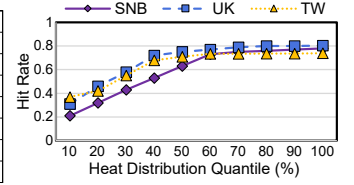

Fig. 11. Sensitivity to network heterogeneity.


Fig. 12. Impact of parameter $\theta$.

additional 100,000 queries are executed as test patterns, with 30% of the data undergoing write operations. To simulate diverse application requirements, each test pattern is assigned a random latency requirement value, which is mapped to the latency interval of one layer in the layered graph. (ii) Offline graph analysis: We run offline graph analytical workloads on RAGraph, with the evaluated algorithms including PageRank [63], Single-Source Shortest Path (SSSP) [64], Core Decomposition [65], Hyperlink-Induced Topic Search (HITS) [66], and Label Propagation Algorithm (LPA) [67]. The inputs are routed subgraphs from three datasets, distributed across four DCs located in Beijing (China), Singapore, London (UK), and Virginia (USA). To reduce communication overhead, RAGraph optimizes the number of iterations. We set the number of iterations to 15 for PageRank, 10 for SSSP, 20 for HITS, and 10 for LPA. The number of iterations for core decomposition is set to the graph's maximum core number.

**Competitors.** To evaluate the effectiveness of GeoLayer in the online mode, we compare it with four representative schemes: Random-3, Top-3, ADP [28], and DCD [27]. Random-3 places replicas in three randomly selected DCs, while Top-3 selects the three DCs with the highest read request frequencies. Both schemes use random request routing. ADP [28] models the replica placement problem as a hypergraph partitioning task, whereas DCD [27] tackles it using an overlapping community detection algorithm. For request routing, both ADP and DCD adopt greedy methods based on the set cover problem.

To assess the effectiveness of GeoLayer for offline request routing, we compare it against three baselines: RAGraph [47], RAGraph+ [68], and GrapH [69]. RAGraph [47] exploits graph monotonicity to design heterogeneity-aware graph computation strategies. We use RAGraph as the underlying execution engine and treat its original data layout as the default baseline. In addition, it introduces a contribution-driven edge migration strategy (denoted as RAGraph+) to further optimize data distribution [68]. GrapH [69] designs an adaptive edge migration strategy that considers vertex traffic and network heterogeneity. All competitors optimize graph analysis performance by strategically selecting data locations.

## B. Effectiveness Analysis in Online Mode

**Overall Performance.** We first evaluate the request latency of test patterns under different replica placement strategies. Figure 7 presents the normalized response time for pattern requests across various datasets, with GeoLayer serving as the baseline (*i.e.,* normalized in unit time). Across all datasets, GeoLayer consistently outperforms all competing approaches, achieving substantially lower response times. Specifically, GeoLayer achieves average speedups of 3.4× over Random-3, 2.8× over Top-3, 1.8× over ADP, and 1.6× over DCD. These improvements stem from GeoLayer's layered graph architecture, which explicitly accounts for network heterogeneity, and its replica placement strategy that is tailored to the characteristics of graph access patterns.

We also report the algorithm execution time, as shown in Figure 10. Random-3 and Top-3 exhibit lower computational complexity due to their simple heuristic strategies. ADP incurs higher overhead as it relies on hypergraph partitioning and multiple rounds of iteration. Although GeoLayer involves additional complexity due to its multi-layered design, it remains efficient by leveraging cluster-level parallelism, producing results within a reasonable execution time.

**Cost Metrics.** We further evaluate system cost metrics to understand the sources of performance improvement. As shown in Figure 8, all objective values are normalized to the total cost ($C$) incurred by GeoLayer. Overall, GeoLayer achieves substantially lower objective values across all datasets. On average, it reduces the cost by 60.8% (up to 65%) over Random-3, 57.5% (up to 63.8%) over Top-3, 31.1% (up to 44.8%) over ADP, and 28.1% (up to 42.9%) over DCD. The marked reduction in association penalty cost ($C^{(A)}$) highlights GeoLayer's effectiveness in addressing network heterogeneity and minimizing inter-DC associated data access. By incorporating the DHD model, which captures the access characteristics of graph patterns, GeoLayer significantly reduces the read cost ($C^{(R)}$). Together, these two enhancements account for the observed reductions in request latency. In contrast, the WAN costs of write operations ($C^{(W)}$) may occasionally rise during replica synchronization, as GeoLayer deploys additional replicas to
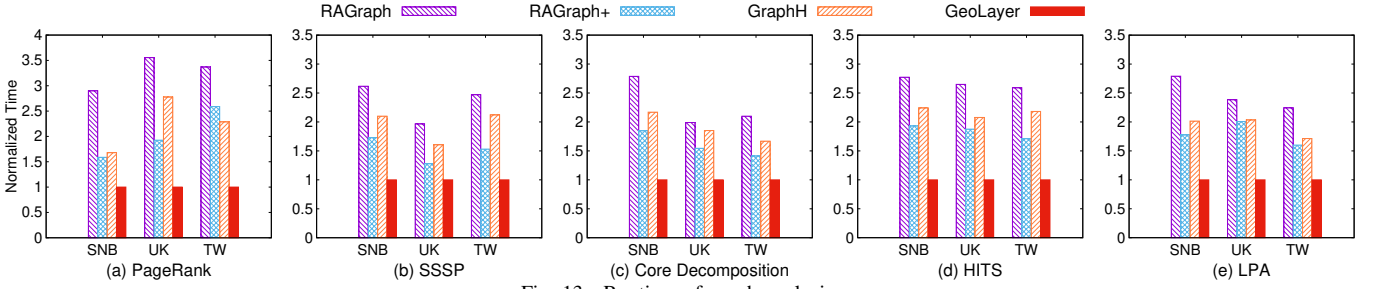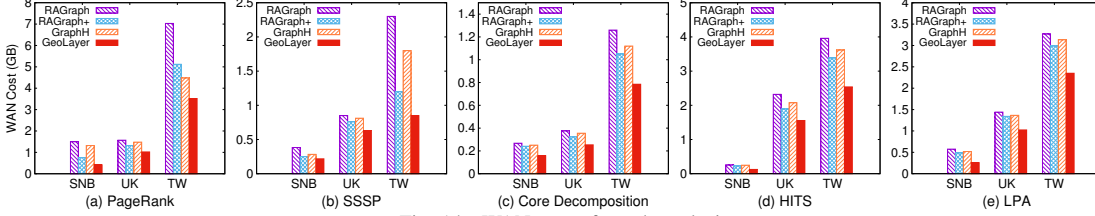
Fig. 13. Runtime of graph analysis.
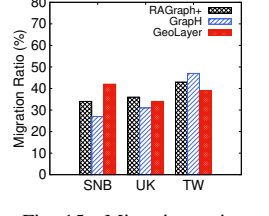


Fig. 14. WAN cost of graph analysis.



Fig. 15. Migration ratio.

further optimize the association penalty cost. Nevertheless, this trade-off is justified, since in practice the frequency of Write updates is typically much lower than that of Reads (*i.e.,* $C^{(R)} + C^{(A)}$), resulting in an overall net benefit.

**Optimality Gap.** To quantify the closeness of GeoLayer to the optimal solution, we employ an MILP solver (PuLP/CBC) [70] on the small-scale WIKI-vote dataset [71] (7,115 vertices and 103,689 edges). The setup involves four DCs, with 1,000 historical query patterns and 100 test patterns. We solve Equation (6) to obtain the global optimum ($C^*$) for comparison. Figure 9 reports the costs of GeoLayer and the global optimum across different metrics, with all values normalized to GeoLayer 's total cost ($C$). We compute the overall optimality gap as: $\text{Gap} = \frac{C-C^*}{C^*} \times 100\% = 7.8\%$. This result shows that GeoLayer achieves small optimality gaps under the same constraints, indicating that our replication and routing schemes are near-optimal on small graphs, while scaling effectively to large deployments where exact optimization is intractable.

**Sensitivity to Network Heterogeneity.** We investigate the impact of network heterogeneity on system performance in geo-distributed environments. To this end, we construct low, medium, and high heterogeneity network settings by selecting DCs from different geographical regions. The low heterogeneity setting consists of 8 DCs located within mainland China. The medium heterogeneity setting includes 8 DCs distributed across Asia. The high heterogeneity setting involves globally distributed DCs, as described in the VII-A setup. Figure 11 shows the normalized response times for graph pattern requests across all datasets. Compared to the competitors, GeoLayer achieves an average speedup of $1.7\times$ (up to $2.4\times$) under the low-heterogeneity network, $1.9\times$ (up to $3.0\times$) under the medium-heterogeneity network, and $2.4\times$ (up to $3.7\times$) under the high-heterogeneity network. These results highlight the robustness of GeoLayer's layered graph architecture in adapting to varying degrees of network diversity, particularly in environments with pronounced network variation.

**Pre-Caching Accurary.** To assess the effectiveness of the

pre-caching strategy, we measure cache hit rates under different thresholds $\theta$, which correspond to increasing quantiles of the vertex heat distribution, as shown in Figure 12. The results indicate that selecting high-heat vertices based on a moderately high quantile (50%–60%) is sufficient to achieve near-optimal hit rates. This supports the observation that graph access patterns exhibit strong skewness, with a small portion of vertices being accessed disproportionately more frequently.

### C. Performance Comparison in Offline Mode

**Execution Time.** Figure 13 shows the normalized execution time of graph analysis algorithms on different datasets under each strategy. The runtime of GeoLayer is reported as the baseline. As can be seen from the results, GeoLayer consistently outperforms others in call cases. Specifically, it achieves an average speedup of $2.6\times$ (up to $3.6\times$) over RAGraph, $1.8\times$ (up to $2.6\times$) over RAGraph+, and $2.0\times$ (up to $2.8\times$) over GrapH. Unlike conventional fixed-partition architectures, GeoLayer employs an adaptive routing mechanism that balances execution site selection and cross-DC communication overhead. By progressively selecting partitions in a layer-wise manner, GeoLayer effectively suppresses the formation of network heterogeneity hotspots, leading to improved efficiency.

**WAN Cost.** We evaluate WAN usage during algorithm execution by measuring the actual volume of inter-DC data transfer. As shown in Figure 14, GeoLayer incurs the lowest WAN cost across all test conditions. Specifically, GeoLayer reduces WAN cost by an average of 42.1% (up to 72.0%) compared to RAGraph, 28.1% (up to 46.8%) compared to RAGraph+, and 34.7% (up to 68.2%) compared to GrapH. The reduction in WAN usage stems from GeoLayer's ability to select fewer execution sites while minimizing the introduced cross–DC communication. GeoLayer constructs 2 execution sites for the SNB graph, and 3 sites for both the UK and TW graphs.

**Migration Ratio.** We also report the ratio of migrated edges under each strategy, as shown in Figure 15. GeoLayer achieves a migration ratio ranging from 34% to 42% across the three
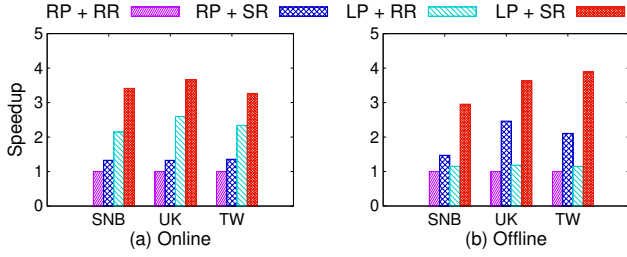
Fig. 16. Performance gain.

datasets. On the SNB graph, GeoLayer exhibits a relatively higher migration ratio, primarily due to the consolidation of data into only two execution sites to enable more efficient processing. All evaluated strategies consider network heterogeneity in geo-distributed environments and aim to improve graph analysis performance by reducing communication overhead through limited data movement.

### D. Ablation Study

In this subsection, we quantify the contributions of the replica placement and request routing components under both online and offline workloads across three datasets. We adopt RP + RR (3-replica random placement + random routing) as the baseline. To isolate component benefits, we replace one or both modules: (i) RP + SR: random placement with stepwise layered routing (routing contribution); (ii) LP + RR: overlap-centric replica placement with random routing (placement contribution); and (iii) LP + SR: both overlap-centric placement and stepwise layered routing (full configuration). For offline analytics, We report PageRank as a representative algorithm.

Figure 16 presents speedups of all configurations relative to the RP + RR baseline. Replacing only the routing with GeoLayer 's stepwise layered routing (RP + SR) yields 1.32–1.36× speedups for online workloads and 1.47–2.50× for offline workloads. Enabling overlap-centric replica placement on the layered graph while keeping routing random (LP + RR) achieves 2.15–2.60× speedups for online workloads and 1.15–1.19× for offline workloads. We observe that LP yields larger gains in the online mode, whereas SR dominates in the offline mode. This is because LP is tuned for online pattern accesses, while offline graph analytics typically involve broader, iterative access patterns that benefit more from SR 's intra-layer locality and stepwise cross-layer escalation. Nevertheless, LP preserves locality over random placement and thus provides consistent gains even for offline workloads. Finally, LP + SR (GeoLayer) improves performance to 3.26–3.66× for online workloads and 2.95–3.88× for offline workloads, indicating that both designs in GeoLayer are complementary and jointly deliver the best performance.

## VIII. RELATED WORKS

**Data Replication and Request Routing.** A large body of work [18]–[24], [72]–[78] focuses on optimizing replica placement and request routing decisions in geo-distributed environments. These efforts address the problem from multiple perspectives, considering factors such as replica placement [22]–[24], [72], caching time [18], data migration [68],

[69], QoS/QoE/SLA guarantees [19], [20], [74], consistency models [21], [75], security policies [76], [77], and load balancing [73], [78]. Several studies further incorporate data association into decision-making processes [17], [25]–[29], [79]. Volley [79] proposes an iterative optimization algorithm based on weighted spherical means to handle inter-data dependencies. Yu et al. [28] model data association as a hypergraph partitioning problem to guide co-location decisions. Liu et al. [27] develop a replica placement strategy based on overlapping community detection to handle associated data. These approaches commonly abstract associated data items as edges and connect them with storage nodes to form a graph structure, thereby enabling graph-based modeling of placement decisions. However, these associations are driven by job-level relationships, whereas GeoLayer focuses on the intrinsic structural properties of the graph.

**Geo-Distributed Graph Data Processing.** Many existing efforts [47], [48], [80]–[85] aim to optimize the performance of graph-structured data processing in geo-distributed settings. GeoGraph [80] leverages clustered DCs and a metagraph to support graph query processing. PGPregel [48] adopts sampling and combiner techniques to enable privacy-preserving graph processing. Hosseinalipour et al. [82] propose an energy-efficient graph job assignment method for geo-distributed cloud networks. RAGraph [47] exploits graph monotonicity to design heterogeneity-aware geo-distributed graph computation strategies. These works focus on optimizing the performance of upstream computing systems, and are thus orthogonal to GeoLayer's design.

**Layered Graph Structures.** Layered designs are widely used in graph computation and analysis. CompressGraph [86] builds a two-layer traversal model via graph compression. Layph [64] partitions a graph into an upper-layer skeleton and lower-layer disjoint subgraphs to constrain the scope of incremental computation in dynamic graph processing. In road networks, hierarchical structures with shortcut edges are standard for speeding up path queries [87], [88]. These layered approaches are tailored to the computational characteristics of graph data, but do not consider heterogeneous latency on edges. In contrast, GeoLayer adopts a layered structure specifically designed for the latency characteristics of cross-partition edges, marking a fundamental distinction from prior work.

## IX. CONCLUSION

This work contributes to optimizing replica placement and request routing for geo-distributed graph-structured data stores. The highlights of GeoLayer include: (1) We design a latency-aware layered graph structure that captures network heterogeneity and simplifies the optimization process. (2) We uncover distinctive pattern access features embedded in the graph topology and leverage them to develop a directed heat diffusion model that guides data placement. (3) We propose a stepwise layered routing approach built on the layered graph, which incrementally traverses layers to locate relevant data. Our experiment validates the effectiveness of GeoLayer.

REFERENCES

[1] F. Yao, X. Yang, S. Gong, S. Yu, Y. Zhang, and G. Yu, "Geolayer: Towards low-latency and cost-efficient geo-distributed graph stores with layered graph (full version)," *arXiv preprint arXiv:2509.02106*, 2025, https://arxiv.org/abs/2509.02106.

[2] L. Jiao, J. Li, T. Xu, and X. Fu, "Cost optimization for online social networks on geo-distributed clouds," in *ICNP*, 2012, pp. 1–10.

[3] P. Li, S. Guo, T. Miyazaki, X. Liao, H. Jin, A. Y. Zomaya, and K. Wang, "Traffic-aware geo-distributed big data analytics with predictable job completion time," *TPDS*, vol. 28, no. 6, pp. 1785–1796, 2016.

[4] A. Elliott, M. Cucuringu, M. M. Luaces, P. Reidy, and G. Reinert, "Anomaly detection in networks with application to financial transaction networks," *arXiv preprint arXiv:1901.00402*, 2019.

[5] L. Gu, J. Hu, D. Zeng, S. Guo, and H. Jin, "Service function chain deployment and network flow scheduling in geo-distributed data centers," *TNSE*, vol. 7, no. 4, pp. 2587–2597, 2020.

[6] "Facebook daily active users (daus)," https://investor.fb.com/investor-events/event-details/2021/Facebook-Q2-2021-Earnings/default.aspx, 2021.

[7] A. group, "Alipay official website," www.alipay.com, 2025, accessed: May 2025.

[8] PayPal, "Paypal official website," https://www.paypal.com, 2025, accessed: May 2025.

[9] X. Zhu, Z. Fu, Z. Pan, J. Jiang, C. Hong, Y. Liu, Y. Fang, W. Chen, and C. He, "Taking the pulse of financial activities with online graph processing," *SIGOPS*, vol. 55, no. 1, pp. 84–87, 2021.

[10] R. Mitra, A. Dongre, P. Dangare, A. Goswami, and M. K. Tiwari, "Knowledge graph driven credit risk assessment for micro, small and medium-sized enterprises," *International Journal of Production Research*, vol. 62, no. 12, pp. 4273–4289, 2024.

[11] Q. Yuan, Y. Liu, Y. Tang, X. Chen, X. Zheng, Q. He, and X. Ao, "Dynamic graph learning with static relations for credit risk assessment," in *AAAI*, vol. 39, no. 12, 2025, pp. 13 133–13 141.

[12] S. V. Subramanyam, "Cloud computing and business process re-engineering in financial systems: The future of digital transformation," *IJITMIS*, vol. 12, no. 1, pp. 126–143, 2021.

[13] F. Moshirian, "Financial services in an increasingly integrated global financial market," *Journal of Banking & Finance*, vol. 32, no. 11, pp. 2288–2292, 2008.

[14] Y.-C. Chou, C.-T. Chen, and S.-H. Huang, "Modeling behavior sequence for personalized fund recommendation with graphical deep collaborative filtering," *Expert Systems with Applications*, vol. 192, p. 116311, 2022.

[15] Z. Pan, T. Wu, Q. Zhao, Q. Zhou, Z. Peng, J. Li, Q. Zhang, G. Feng, and X. Zhu, "Geaflow: a graph extended and accelerated dataflow system," *SIGMOD*, vol. 1, no. 2, pp. 1–27, 2023.

[16] A. Docs, "Fraud protection," https://docs.antom.com/ac/risk_control/overview, accessed: March 2025.

[17] B. Yu and J. Pan, "Location-aware associated data placement for geo-distributed data-intensive applications," in *INFOCOM*, 2015, pp. 603–611.

[18] S. Liu, X. Mo, M. Hershcovitch, H. Zhang, A. Cheng, G. Girmonsky, G. Vernik, M. Factor, T. Bang, S. Ponnapalli *et al.*, "Skystore: Cost-optimized object storage across regions and clouds," *arXiv preprint arXiv:2502.20818*, 2025.

[19] H. Wang, H. Shen, Z. Li, and S. Tian, "Geocol: A geo-distributed cloud storage system with low cost and latency using reinforcement learning," in *ICDCS*, 2021, pp. 149–159.

[20] Z. Wu, C. Yu, and H. V. Madhyastha, "{CosTLO}:{Cost-Effective} redundancy for lower latency variance on cloud storage services," in *NSDI*, 2015, pp. 543–557.

[21] W.-C. Chang and P.-C. Wang, "Write-aware replica placement for cloud computing," *JSAC*, vol. 37, no. 3, pp. 656–667, 2019.

[22] C. Li, J. Liu, W. Li, and Y. Luo, "Adaptive priority-based data placement and multi-task scheduling in geo-distributed cloud systems," *KBS*, vol. 224, p. 107050, 2021.

[23] T. Bang, C. Douglas, N. Crooks, and J. M. Hellerstein, "Skypie: A fast & accurate oracle for object placement," *SIGMOD*, vol. 2, no. 1, pp. 1–27, 2024.

[24] Y. Mansouri, A. N. Toosi, and R. Buyya, "Cost optimization for dynamic replication and migration of data in cloud data centers," *TCC*, vol. 7, no. 3, pp. 705–718, 2017.

[25] L. Jiao, J. Lit, W. Du, and X. Fu, "Multi-objective data placement for multi-cloud socially aware services," in *INFOCOM*, 2014, pp. 28–36.

[26] B. Yu and J. Pan, "Sketch-based data placement among geo-distributed datacenters for cloud storages," in *INFOCOM*, 2016, pp. 1–9.

[27] K. Liu, J. Peng, J. Wang, W. Liu, Z. Huang, and J. Pan, "Scalable and adaptive data replica placement for geo-distributed cloud storages," *TPDS*, vol. 31, no. 7, pp. 1575–1587, 2020.

[28] B. Yu and J. Pan, "A framework of hypergraph-based data placement among geo-distributed datacenters," *IEEE Transactions on Services Computing*, vol. 13, no. 3, pp. 395–409, 2020.

[29] Y. Huang, Y. Shi, Z. Zhong, Y. Feng, J. Cheng, J. Li, H. Fan, C. Li, T. Guan, and J. Zhou, "Yugong: Geo-distributed data and job placement at scale," *PVLDB*, vol. 12, no. 12, pp. 2155–2169, 2019.

[30] X. Zhang, A. Kashaf, Y. Zou, W. Zhang, W. Liao, H. Song, J. Ye, Y. Li, R. Shi, Y. Tian *et al.*, "Reslake: Towards minimum job latency and balanced resource utilization in geo-distributed job scheduling," *PVLDB*, vol. 17, no. 12, pp. 3934–3946, 2024.

[31] A. Deutsch, N. Francis, A. Green, K. Hare, B. Li, L. Libkin, T. Lindaaker, V. Marsault, W. Martens, J. Michels *et al.*, "Graph pattern matching in gql and sql/pgq," in *SIGMOD*, 2022, pp. 2246–2258.

[32] Alipay, "Alipay," https://https://global.alipay.com, 2025, accessed: August 2025.

[33] Q. Zhong, Y. Liu, X. Ao, B. Hu, J. Feng, J. Tang, and Q. He, "Financial defaulter detection on online credit payment via multi-view attributed heterogeneous information network," in *WWW*, 2020, pp. 785–795.

[34] Y.-L. Zhang, J. Zhou, W. Zheng, J. Feng, L. Li, Z. Liu, M. Li, Z. Zhang, C. Chen, X. Li *et al.*, "Distributed deep forest and its application to automatic detection of cash-out fraud," *TIST*, vol. 10, no. 5, pp. 1–19, 2019.

[35] B. Fatemipour, Z. Zhang, and M. St-Hilaire, "A survey on replica transfer optimization schemes in geographically distributed data centers," *IEEE TNSM*, 2024.

[36] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," *SIGCOMM*, vol. 45, no. 4, pp. 421–434, 2015.

[37] W. Fan, P. Lu, W. Yu, J. Xu, Q. Yin, X. Luo, J. Zhou, and R. Jin, "Adaptive asynchronous parallelization of graph algorithms," *=TODS*, vol. 45, no. 2, pp. 1–45, 2020.

[38] A. S3, "Amazon s3 pricing," https://aws.amazon.com/cn/s3/glacier/pricing/, 2025, accessed: March 2025.

[39] G. C. Storage, "Google cloud storage pricing," https://cloud.google.com/storage/pricing, 2025, accessed: March 2025.

[40] I. C. O. Store, "Ibm cloud object store pricing," https://cloud.ibm.com/objectstorage/create#pricing, 2025, accessed: March 2025.

[41] A. B. Storage, "Azure blob storage pricing," https://azure.microsoft.com/en-us/pricing/details/storage/blobs/, 2025, accessed: March 2025.

[42] A. Cloud, "Alibaba cloud's oss billing details," https://www.alibabacloud.com/zh/product/oss/pricing?_p_lc=1, 2025, accessed: March 2025.

[43] D. Durner, V. Leis, and T. Neumann, "Exploiting cloud object storage for high-performance analytics," *PVLDB*, vol. 16, no. 11, pp. 2769–2782, 2023.

[44] Y. Liu, C. Zhou, J. Gao, and Z. Fan, "Giraphasync: supporting online and offline graph processing via adaptive asynchronous message processing," in *CIKM*, 2016, pp. 479–488.

[45] G. Szárnyas, J. Waudby, B. A. Steer, D. Szakállas, A. Birler, M. Wu, Y. Zhang, and P. Boncz, "The ldbc social network benchmark: Business intelligence workload," *Proceedings of the VLDB Endowment*, vol. 16, no. 4, pp. 877–890, 2022.

[46] S. Qi, H. Lin, Z. Guo, G. Szárnyas, B. Tong, Y. Zhou, B. Yang, J. Zhang, Z. Wang, Y. Shen *et al.*, "The ldbc financial benchmark," *arXiv preprint arXiv:2306.15975*, 2023.

[47] F. Yao, Q. Tao, W. Yu, Y. Zhang, S. Gong, Q. Wang, G. Yu, and J. Zhou, "Ragraph: A region-aware framework for geo-distributed graph processing," *PVLDB*, vol. 17, no. 3, pp. 264–277, 2023.

[48] A. C. Zhou, R. Qiu, T. Lambert, T. Allard, S. Ibrahim, and A. E. Abbadi, "Pgpregel: an end-to-end system for privacy-preserving graph processing in geo-distributed data centers," in *SoCC*, 2022, pp. 386–402.

[49] L. Zeng, C. Yang, P. Huang, Z. Zhou, S. Yu, and X. Chen, "Gnn at the edge: Cost-efficient graph neural network processing over distributed edge servers," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 3, pp. 720–739, 2022.

[50] F. Xia, K. Sun, S. Yu, A. Aziz, L. Wan, S. Pan, and H. Liu, "Graph learning: A survey," *TAI*, vol. 2, no. 2, pp. 109–127, 2021.

[51] W. Fan, X. Wang, Y. Wu, and D. Deng, "Distributed graph simulation: Impossibility and possibility," *PVLDB*, vol. 7, no. 12, pp. 1083–1094, 2014.

[52] M. Goetschalckx, C. J. Vidal, and K. Dogan, "Modeling and design of global logistics systems: A review of integrated strategic and tactical models and design algorithms," *Eur. J. Oper. Res.*, vol. 143, no. 1, pp. 1–18, 2002.

[53] J. Hartmanis, "Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson)," *Siam Review*, vol. 24, no. 1, p. 90, 1982.

[54] A. C. Zhou, J. Luo, R. Qiu, H. Tan, B. He, and R. Mao, "Adaptive partitioning for large-scale graph analytics in geo-distributed data centers," in *ICDE*, 2022, pp. 2818–2830.

[55] G. Szárnyas, "Ldbc social network benchmark graphs," https://hdl.handle.net/11112/e6e00558-a2c3-9214-473e-04a16de09bf8.

[56] F. Ruskey and M. Weston, "Venn diagrams," *The electronic journal of combinatorics*, pp. DS5–Jun, 2012.

[57] H. Qi, K. Luo, L. He, Y. Zhang, M. Cai, J. Dai, B. He, H. Jin, Z. Zhang, J. Zhao *et al.*, "Ohminer: An overlap-centric system for efficient hypergraph pattern mining," in *EuroSys*, 2025, pp. 621–636.

[58] UK-2005, https://law.di.unimi.it/webdata/uk-2005/, accessed: March 2025.

[59] twitter 2010, https://snap.stanford.edu/data/twitter-2010.html, accessed: March 2025.

[60] LDBC-SNB, https://ldbccouncil.org/benchmarks/snb/, accessed: May 2025.

[61] O. Erling, A. Averbuch, J. Larriba-Pey, H. Chafi, A. Gubichev, A. Prat, M.-D. Pham, and P. Boncz, "The ldbc social network benchmark: Interactive workload," in *SIGMOD*, 2015, pp. 619–630.

[62] Z. Qian, C. Min, L. Lai, Y. Fang, G. Li, Y. Yao, B. Lyu, X. Zhou, Z. Chen, and J. Zhou, "{GAIA}: A system for interactive analysis on distributed graphs using a {High-Level} language," in *NSDI*, 2021, pp. 321–335.

[63] Y. Zhang, Q. Gao, L. Gao, and C. Wang, "Maiter: An asynchronous graph processing framework for delta-based accumulative iterative computation," *TPDS*, vol. 25, no. 8, pp. 2091–2100, 2013.

[64] S. Yu, S. Gong, Y. Zhang, W. Yu, Q. Yin, C. Tian, Q. Tao, Y. Yan, G. Yu, and J. Zhou, "Layph: Making change propagation constraint in incremental graph processing by layering graph," in *ICDE*, 2023, pp. 2766–2779.

[65] V. Batagelj and M. Zaversnik, "An o (m) algorithm for cores decomposition of networks. corr," *arXiv preprint cs.DS/0310049*, vol. 37, 2003.

[66] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *JACM*, vol. 46, no. 5, pp. 604–632, 1999.

[67] J. Xie and B. K. Szymanski, "Community detection using a neighborhood strength driven label propagation algorithm," in *IEEE Network Science Workshop*, 2011, pp. 188–195.

[68] F. Yao, Q. Tao, S. Lin, Y. Zhang, W. Yu, S. Gong, Q. Wang, G. Yu, and J. Zhou, "Towards efficient graph processing in geo-distributed data centers," *TPDS*, 2024.

[69] C. Mayer, M. A. Tariq, C. Li, and K. Rothermel, "Graph: Heterogeneity-aware graph computation with adaptive partitioning," in *ICDCS*, 2016, pp. 118–128.

[70] PuLP, "Pulp apis," https://coin-or.github.io/pulp/technical/solvers.html, 2025, accessed: August 2025.

[71] WIKI-vote, https://snap.stanford.edu/data/wiki-Vote.html, accessed: August 2025.

[72] Y. Shao, C. Li, and H. Tang, "A data replica placement strategy for iot workflows in collaborative edge and cloud environments," *Computer Networks*, vol. 148, pp. 46–59, 2019.

[73] H.-C. Hsieh and M.-L. Chiang, "The incremental load balance cloud algorithm by using dynamic data deployment," *Journal of grid computing*, vol. 17, pp. 553–575, 2019.

[74] T. Zhang, "A qos-enhanced data replication service in virtualised cloud environments," *Int. J. Netw. Virtual Organ.*, vol. 22, no. 1, pp. 1–16, 2020.

[75] C. Li, C. Wang, H. Tang, and Y. Luo, "Scalable and dynamic replica consistency maintenance for edge-cloud system," *Future Generation Computer Systems*, vol. 101, pp. 590–604, 2019.

[76] M. Ramanan and P. Vivekanandan, "Efficient data integrity and data replication in cloud using stochastic diffusion method," *Cluster Computing*, vol. 22, pp. 14 999–15 006, 2019.

[77] L. Luo, L. Xing, and G. Levitin, "Optimizing dynamic survivability and security of replicated data in cloud systems under co-residence attacks," *Reliab. Eng. Syst. Saf.*, vol. 192, p. 106265, 2019.

[78] K. L. Bogdanov, W. Reda, G. Q. Maguire Jr, D. Kostić, and M. Canini, "Fast and accurate load balancing for geo-distributed storage systems," in *SoCC*, 2018, pp. 386–400.

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

[80] Y. Yuan, D. Ma, Z. Wen, Y. Ma, G. Wang, and L. Chen, "Efficient graph query processing over geo-distributed datacenters," in *SIGIR*, 2020, pp. 619–628.

[81] A. P. Iyer, A. Panda, M. Chowdhury, A. Akella, S. Shenker, and I. Stoica, "Monarch: Gaining command on {Geo-Distributed} graph analytics," in *HotCloud*, 2018.

[82] S. Hosseinalipour, A. Nayak, and H. Dai, "Power-aware allocation of graph jobs in geo-distributed cloud networks," *TPDS*, vol. 31, no. 4, pp. 749–765, 2019.

[83] A. C. Zhou, J. Luo, R. Qiu, H. Tan, B. He, and R. Mao, "Adaptive partitioning for large-scale graph analytics in geo-distributed data centers," in *ICDE*, 2022, pp. 2818–2830.

[84] H. Tan, Y. Xiao, A. C. Zhou, K. Lu, and X. Yang, "Distributed and adaptive partitioning for large graphs in geo-distributed data centers," *TPDS*, 2025.

[85] A. C. Zhou, B. Shen, Y. Xiao, S. Ibrahim, and B. He, "Cost-aware partitioning for efficient large graph processing in geo-distributed datacenters," *TPDS*, vol. 31, no. 7, pp. 1707–1723, 2019.

[86] Z. Chen, F. Zhang, J. Guan, J. Zhai, X. Shen, H. Zhang, W. Shu, and X. Du, "Compressgraph: Efficient parallel graph analytics with rule-based compression," *SIGMOD*, vol. 1, no. 1, pp. 1–31, 2023.

[87] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: Faster and simpler hierarchical routing in road networks," in *International workshop on experimental and efficient algorithms*, 2008, pp. 319–333.

[88] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "Hierarchical hub labelings for shortest paths," in *ESA*, 2012, pp. 24–35.

[89] A. T. Bharucha-Reid, "Fixed point theorems in probabilistic analysis," *Bulletin of the American Mathematical Society*, vol. 82, no. 5, pp. 641–657, 1976.

[90] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions—i," *Mathematical programming*, vol. 14, no. 1, pp. 265–294, 1978.

## References

[1] F. Yao, X. Yang, S. Gong, S. Yu, Y. Zhang, and G. Yu, "Geolayer: Towards low-latency and cost-efficient geo-distributed graph stores with layered graph (full version)," *arXiv preprint arXiv:2509.02106*, 2025, https://arxiv.org/abs/2509.02106.

[2] L. Jiao, J. Li, T. Xu, and X. Fu, "Cost optimization for online social networks on geo-distributed clouds," in *ICNP*, 2012, pp. 1–10.

[3] P. Li, S. Guo, T. Miyazaki, X. Liao, H. Jin, A. Y. Zomaya, and K. Wang, "Traffic-aware geo-distributed big data analytics with predictable job completion time," *TPDS*, vol. 28, no. 6, pp. 1785–1796, 2016.

[4] A. Elliott, M. Cucuringu, M. M. Luaces, P. Reidy, and G. Reinert, "Anomaly detection in networks with application to financial transaction networks," *arXiv preprint arXiv:1901.00402*, 2019.

[5] L. Gu, J. Hu, D. Zeng, S. Guo, and H. Jin, "Service function chain deployment and network flow scheduling in geo-distributed data centers," *TNSE*, vol. 7, no. 4, pp. 2587–2597, 2020.

[6] "Facebook daily active users (daus)," https://investor.fb.com/investor-events/event-details/2021/Facebook-Q2-2021-Earnings/default.aspx, 2021.

[7] A. group, "Alipay official website," www.alipay.com, 2025, accessed: May 2025.

[8] PayPal, "Paypal official website," https://www.paypal.com, 2025, accessed: May 2025.

[9] X. Zhu, Z. Fu, Z. Pan, J. Jiang, C. Hong, Y. Liu, Y. Fang, W. Chen, and C. He, "Taking the pulse of financial activities with online graph processing," *SIGOPS*, vol. 55, no. 1, pp. 84–87, 2021.

[10] R. Mitra, A. Dongre, P. Dangare, A. Goswami, and M. K. Tiwari, "Knowledge graph driven credit risk assessment for micro, small and medium-sized enterprises," *International Journal of Production Research*, vol. 62, no. 12, pp. 4273–4289, 2024.

[11] Q. Yuan, Y. Liu, Y. Tang, X. Chen, X. Zheng, Q. He, and X. Ao, "Dynamic graph learning with static relations for credit risk assessment," in *AAAI*, vol. 39, no. 12, 2025, pp. 13 133–13 141.

[12] S. V. Subramanyam, "Cloud computing and business process re-engineering in financial systems: The future of digital transformation," *IJITMIS*, vol. 12, no. 1, pp. 126–143, 2021.

[13] F. Moshirian, "Financial services in an increasingly integrated global financial market," *Journal of Banking & Finance*, vol. 32, no. 11, pp. 2288–2292, 2008.

[14] Y.-C. Chou, C.-T. Chen, and S.-H. Huang, "Modeling behavior sequence for personalized fund recommendation with graphical deep collaborative filtering," *Expert Systems with Applications*, vol. 192, p. 116311, 2022.

[15] Z. Pan, T. Wu, Q. Zhao, Q. Zhou, Z. Peng, J. Li, Q. Zhang, G. Feng, and X. Zhu, "Geaflow: a graph extended and accelerated dataflow system," *SIGMOD*, vol. 1, no. 2, pp. 1–27, 2023.

[16] A. Docs, "Fraud protection," https://docs.antom.com/ac/risk_control/overview, accessed: March 2025.

[17] B. Yu and J. Pan, "Location-aware associated data placement for geo-distributed data-intensive applications," in *INFOCOM*, 2015, pp. 603–611.

[18] S. Liu, X. Mo, M. Hershcovitch, H. Zhang, A. Cheng, G. Girmonsky, G. Vernik, M. Factor, T. Bang, S. Ponnapalli *et al.*, "Skystore: Cost-optimized object storage across regions and clouds," *arXiv preprint arXiv:2502.20818*, 2025.

[19] H. Wang, H. Shen, Z. Li, and S. Tian, "Geocol: A geo-distributed cloud storage system with low cost and latency using reinforcement learning," in *ICDCS*, 2021, pp. 149–159.

[20] Z. Wu, C. Yu, and H. V. Madhyastha, "{CosTLO}:{Cost-Effective} redundancy for lower latency variance on cloud storage services," in *NSDI*, 2015, pp. 543–557.

[21] W.-C. Chang and P.-C. Wang, "Write-aware replica placement for cloud computing," *JSAC*, vol. 37, no. 3, pp. 656–667, 2019.

[22] C. Li, J. Liu, W. Li, and Y. Luo, "Adaptive priority-based data placement and multi-task scheduling in geo-distributed cloud systems," *KBS*, vol. 224, p. 107050, 2021.

[23] T. Bang, C. Douglas, N. Crooks, and J. M. Hellerstein, "Skypie: A fast & accurate oracle for object placement," *SIGMOD*, vol. 2, no. 1, pp. 1–27, 2024.

[24] Y. Mansouri, A. N. Toosi, and R. Buyya, "Cost optimization for dynamic replication and migration of data in cloud data centers," *TCC*, vol. 7, no. 3, pp. 705–718, 2017.

[25] L. Jiao, J. Lit, W. Du, and X. Fu, "Multi-objective data placement for multi-cloud socially aware services," in *INFOCOM*, 2014, pp. 28–36.

[26] B. Yu and J. Pan, "Sketch-based data placement among geo-distributed datacenters for cloud storages," in *INFOCOM*, 2016, pp. 1–9.

[27] K. Liu, J. Peng, J. Wang, W. Liu, Z. Huang, and J. Pan, "Scalable and adaptive data replica placement for geo-distributed cloud storages," *TPDS*, vol. 31, no. 7, pp. 1575–1587, 2020.

[28] B. Yu and J. Pan, "A framework of hypergraph-based data placement among geo-distributed datacenters," *IEEE Transactions on Services Computing*, vol. 13, no. 3, pp. 395–409, 2020.

[29] Y. Huang, Y. Shi, Z. Zhong, Y. Feng, J. Cheng, J. Li, H. Fan, C. Li, T. Guan, and J. Zhou, "Yugong: Geo-distributed data and job placement at scale," *PVLDB*, vol. 12, no. 12, pp. 2155–2169, 2019.

[30] X. Zhang, A. Kashaf, Y. Zou, W. Zhang, W. Liao, H. Song, J. Ye, Y. Li, R. Shi, Y. Tian *et al.*, "Reslake: Towards minimum job latency and balanced resource utilization in geo-distributed job scheduling," *PVLDB*, vol. 17, no. 12, pp. 3934–3946, 2024.

[31] A. Deutsch, N. Francis, A. Green, K. Hare, B. Li, L. Libkin, T. Lindaaker, V. Marsault, W. Martens, J. Michels *et al.*, "Graph pattern matching in gql and sql/pgq," in *SIGMOD*, 2022, pp. 2246–2258.

[32] Alipay, "Alipay," https://https://global.alipay.com, 2025, accessed: August 2025.

[33] Q. Zhong, Y. Liu, X. Ao, B. Hu, J. Feng, J. Tang, and Q. He, "Financial defaulter detection on online credit payment via multi-view attributed heterogeneous information network," in *WWW*, 2020, pp. 785–795.

[34] Y.-L. Zhang, J. Zhou, W. Zheng, J. Feng, L. Li, Z. Liu, M. Li, Z. Zhang, C. Chen, X. Li *et al.*, "Distributed deep forest and its application to automatic detection of cash-out fraud," *TIST*, vol. 10, no. 5, pp. 1–19, 2019.

[35] B. Fatemipour, Z. Zhang, and M. St-Hilaire, "A survey on replica transfer optimization schemes in geographically distributed data centers," *IEEE TNSM*, 2024.

[36] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," *SIGCOMM*, vol. 45, no. 4, pp. 421–434, 2015.

[37] W. Fan, P. Lu, W. Yu, J. Xu, Q. Yin, X. Luo, J. Zhou, and R. Jin, "Adaptive asynchronous parallelization of graph algorithms," =*TODS*, vol. 45, no. 2, pp. 1–45, 2020.

[38] A. S3, "Amazon s3 pricing," https://aws.amazon.com/cn/s3/glacier/pricing/, 2025, accessed: March 2025.

[39] G. C. Storage, "Google cloud storage pricing," https://cloud.google.com/storage/pricing, 2025, accessed: March 2025.

[40] I. C. O. Store, "Ibm cloud object store pricing," https://cloud.ibm.com/objectstorage/create#pricing, 2025, accessed: March 2025.

[41] A. B. Storage, "Azure blob storage pricing," https://azure.microsoft.com/en-us/pricing/details/storage/blobs/, 2025, accessed: March 2025.

[42] A. Cloud, "Alibaba cloud's oss billing details," https://www.alibabacloud.com/zh/product/oss/pricing?_p_lc=1, 2025, accessed: March 2025.

[43] D. Durner, V. Leis, and T. Neumann, "Exploiting cloud object storage for high-performance analytics," *PVLDB*, vol. 16, no. 11, pp. 2769–2782, 2023.

[44] Y. Liu, C. Zhou, J. Gao, and Z. Fan, "Giraphasync: supporting online and offline graph processing via adaptive asynchronous message processing," in *CIKM*, 2016, pp. 479–488.

[45] G. Szárnyas, J. Waudby, B. A. Steer, D. Szakállas, A. Birler, M. Wu, Y. Zhang, and P. Boncz, "The ldbc social network benchmark: Business intelligence workload," *Proceedings of the VLDB Endowment*, vol. 16, no. 4, pp. 877–890, 2022.

[46] S. Qi, H. Lin, Z. Guo, G. Szárnyas, B. Tong, Y. Zhou, B. Yang, J. Zhang, Z. Wang, Y. Shen *et al.*, "The ldbc financial benchmark," *arXiv preprint arXiv:2306.15975*, 2023.

[47] F. Yao, Q. Tao, W. Yu, Y. Zhang, S. Gong, Q. Wang, G. Yu, and J. Zhou, "Ragraph: A region-aware framework for geo-distributed graph processing," *PVLDB*, vol. 17, no. 3, pp. 264–277, 2023.

[48] A. C. Zhou, R. Qiu, T. Lambert, T. Allard, S. Ibrahim, and A. E. Abbadi, "Pgpregel: an end-to-end system for privacy-preserving graph processing in geo-distributed data centers," in *SoCC*, 2022, pp. 386–402.

[49] L. Zeng, C. Yang, P. Huang, Z. Zhou, S. Yu, and X. Chen, "Gnn at the edge: Cost-efficient graph neural network processing over distributed edge servers," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 3, pp. 720–739, 2022.

[50] F. Xia, K. Sun, S. Yu, A. Aziz, L. Wan, S. Pan, and H. Liu, "Graph learning: A survey," *TAI*, vol. 2, no. 2, pp. 109–127, 2021.

[51] W. Fan, X. Wang, Y. Wu, and D. Deng, "Distributed graph simulation: Impossibility and possibility," *PVLDB*, vol. 7, no. 12, pp. 1083–1094, 2014.

[52] M. Goetschalckx, C. J. Vidal, and K. Dogan, "Modeling and design of global logistics systems: A review of integrated strategic and tactical models and design algorithms," *Eur. J. Oper. Res.*, vol. 143, no. 1, pp. 1–18, 2002.

[53] J. Hartmanis, "Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson)," *Siam Review*, vol. 24, no. 1, p. 90, 1982.

[54] A. C. Zhou, J. Luo, R. Qiu, H. Tan, B. He, and R. Mao, "Adaptive partitioning for large-scale graph analytics in geo-distributed data centers," in *ICDE*, 2022, pp. 2818–2830.

[55] G. Szárnyas, "Ldbc social network benchmark graphs," https://hdl. handle.net/11112/e6e00558-a2c3-9214-473e-04a16de09bf8.

[56] F. Ruskey and M. Weston, "Venn diagrams," *The electronic journal of combinatorics*, pp. DS5–Jun, 2012.

[57] H. Qi, K. Luo, L. He, Y. Zhang, M. Cai, J. Dai, B. He, H. Jin, Z. Zhang, J. Zhao *et al.*, "Ohminer: An overlap-centric system for efficient hypergraph pattern mining," in *EuroSys*, 2025, pp. 621–636.

[58] UK-2005, https://law.di.unimi.it/webdata/uk-2005/, accessed: March 2025.

[59] twitter 2010, https://snap.stanford.edu/data/twitter-2010.html, accessed: March 2025.

[60] LDBC-SNB, https://ldbccouncil.org/benchmarks/snb/, accessed: May 2025.

[61] O. Erling, A. Averbuch, J. Larriba-Pey, H. Chafi, A. Gubichev, A. Prat, M.-D. Pham, and P. Boncz, "The ldbc social network benchmark: Interactive workload," in *SIGMOD*, 2015, pp. 619–630.

[62] Z. Qian, C. Min, L. Lai, Y. Fang, G. Li, Y. Yao, B. Lyu, X. Zhou, Z. Chen, and J. Zhou, "{GAIA}: A system for interactive analysis on distributed graphs using a {High-Level} language," in *NSDI*, 2021, pp. 321–335.

[63] Y. Zhang, Q. Gao, L. Gao, and C. Wang, "Maiter: An asynchronous graph processing framework for delta-based accumulative iterative computation," *TPDS*, vol. 25, no. 8, pp. 2091–2100, 2013.

[64] S. Yu, S. Gong, Y. Zhang, W. Yu, Q. Yin, C. Tian, Q. Tao, Y. Yan, G. Yu, and J. Zhou, "Layph: Making change propagation constraint in incremental graph processing by layering graph," in *ICDE*, 2023, pp. 2766–2779.

[65] V. Batagelj and M. Zaversnik, "An o (m) algorithm for cores decomposition of networks. corr," *arXiv preprint cs.DS/0310049*, vol. 37, 2003.

[66] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *JACM*, vol. 46, no. 5, pp. 604–632, 1999.

[67] J. Xie and B. K. Szymanski, "Community detection using a neighborhood strength driven label propagation algorithm," in *IEEE Network Science Workshop*, 2011, pp. 188–195.

[68] F. Yao, Q. Tao, S. Lin, Y. Zhang, W. Yu, S. Gong, Q. Wang, G. Yu, and J. Zhou, "Towards efficient graph processing in geo-distributed data centers," *TPDS*, 2024.

[69] C. Mayer, M. A. Tariq, C. Li, and K. Rothermel, "Graph: Heterogeneity-aware graph computation with adaptive partitioning," in *ICDCS*, 2016, pp. 118–128.

[70] PuLP, "Pulp apis," https://coin-or.github.io/pulp/technical/solvers.html, 2025, accessed: August 2025.

[71] WIKI-vote, https://snap.stanford.edu/data/wiki-Vote.html, accessed: August 2025.

[72] Y. Shao, C. Li, and H. Tang, "A data replica placement strategy for iot workflows in collaborative edge and cloud environments," *Computer Networks*, vol. 148, pp. 46–59, 2019.

[73] H.-C. Hsieh and M.-L. Chiang, "The incremental load balance cloud algorithm by using dynamic data deployment," *Journal of grid computing*, vol. 17, pp. 553–575, 2019.

[74] T. Zhang, "A qos-enhanced data replication service in virtualised cloud environments," *Int. J. Netw. Virtual Organ.*, vol. 22, no. 1, pp. 1–16, 2020.

[75] C. Li, C. Wang, H. Tang, and Y. Luo, "Scalable and dynamic replica consistency maintenance for edge-cloud system," *Future Generation Computer Systems*, vol. 101, pp. 590–604, 2019.

[76] M. Ramanan and P. Vivekanandan, "Efficient data integrity and data replication in cloud using stochastic diffusion method," *Cluster Computing*, vol. 22, pp. 14 999–15 006, 2019.

[77] L. Luo, L. Xing, and G. Levitin, "Optimizing dynamic survivability and security of replicated data in cloud systems under co-residence attacks," *Reliab. Eng. Syst. Saf.*, vol. 192, p. 106265, 2019.

[78] K. L. Bogdanov, W. Reda, G. Q. Maguire Jr, D. Kostić, and M. Canini, "Fast and accurate load balancing for geo-distributed storage systems," in *SoCC*, 2018, pp. 386–400.

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

[80] Y. Yuan, D. Ma, Z. Wen, Y. Ma, G. Wang, and L. Chen, "Efficient graph query processing over geo-distributed datacenters," in *SIGIR*, 2020, pp. 619–628.

[81] A. P. Iyer, A. Panda, M. Chowdhury, A. Akella, S. Shenker, and I. Stoica, "Monarch: Gaining command on {Geo-Distributed} graph analytics," in *HotCloud*, 2018.

[82] S. Hosseinalipour, A. Nayak, and H. Dai, "Power-aware allocation of graph jobs in geo-distributed cloud networks," *TPDS*, vol. 31, no. 4, pp. 749–765, 2019.

[83] A. C. Zhou, J. Luo, R. Qiu, H. Tan, B. He, and R. Mao, "Adaptive partitioning for large-scale graph analytics in geo-distributed data centers," in *ICDE*, 2022, pp. 2818–2830.

[84] H. Tan, Y. Xiao, A. C. Zhou, K. Lu, and X. Yang, "Distributed and adaptive partitioning for large graphs in geo-distributed data centers," *TPDS*, 2025.

[85] A. C. Zhou, B. Shen, Y. Xiao, S. Ibrahim, and B. He, "Cost-aware partitioning for efficient large graph processing in geo-distributed datacenters," *TPDS*, vol. 31, no. 7, pp. 1707–1723, 2019.

[86] Z. Chen, F. Zhang, J. Guan, J. Zhai, X. Shen, H. Zhang, W. Shu, and X. Du, "Compressgraph: Efficient parallel graph analytics with rule-based compression," *SIGMOD*, vol. 1, no. 1, pp. 1–31, 2023.

[87] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: Faster and simpler hierarchical routing in road networks," in *International workshop on experimental and efficient algorithms*, 2008, pp. 319–333.

[88] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "Hierarchical hub labelings for shortest paths," in *ESA*, 2012, pp. 24–35.

[89] A. T. Bharucha-Reid, "Fixed point theorems in probabilistic analysis," *Bulletin of the American Mathematical Society*, vol. 82, no. 5, pp. 641–657, 1976.

[90] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions—i," *Mathematical programming*, vol. 14, no. 1, pp. 265–294, 1978.

We first present the theoretical analysis of GeoLayer, followed by its concrete implementation. The major notations are summarized in Table IV.

### A. Convergence of DHD Model

In Theorem 1, we provide a theoretical guarantee for the existence of a non-trivial steady-state solution in the heat system defined by the DHD model.

**Theorem 1** (Non-trivial steady-state existence). *If* $\alpha < \frac{\gamma}{(1-\gamma)\left\|L_{dir}^*\right\|}$, *then the steady-state Equation (12) admits a unique non-trivial solution* $\mathcal{H}^* = \beta\left(\gamma\mathbf{1} - X^*\right)^{-1}Q^*$, *where* $X^* = \alpha\left(1 - \gamma\right)L_{dir}^*$, *and* $\left\|L_{dir}^*\right\|$ *is the induced infinity norm.*

**Proof Sketch.** Let $X^* = \alpha(1-\gamma)L_{dir}^*$. Then Equation (12) can be rewritten as

$$\left(\gamma\mathbf{1} - X^*\right)\mathcal{H}^* = \beta\,Q^*. \tag{15}$$

Since $\alpha < \frac{\gamma}{(1-\gamma)\|L_{dir}^*\|}$, we have

$$\|X^*\| = \alpha(1-\gamma)\|L_{dir}^*\| < \gamma,$$

so $\left\|X^*/\gamma\right\| < 1$. Hence the Neumann series

$$\left(\gamma\mathbf{1} - X^*\right)^{-1} = \frac{1}{\gamma}\sum_{k=0}^{\infty}\left(\frac{X^*}{\gamma}\right)^k$$

converges, implying $\gamma\mathbf{1} - X^*$ is invertible. Solving Equation (15) yields the unique solution $\mathcal{H}^* = \beta(\gamma\mathbf{1} - X^*)^{-1}Q^*$.

Moreover, $L_{dir}^*$ is an M-matrix, so $\gamma\mathbf{1} - X^*$ remains an M-matrix whose inverse is entrywise non-negative. Because $Q^* \not\equiv 0$, it follows that $\mathcal{H}^* \not\equiv 0$, i.e. a non-trivial steady state exists.

To see uniqueness from a fixed-point view, define

$$\mathcal{F}(\mathcal{H}) = \frac{X^*}{\gamma}\mathcal{H} + \frac{\beta}{\gamma}Q^*.$$

A fixed point of $\mathcal{F}$ satisfies $\mathcal{H} = \mathcal{F}(\mathcal{H})$, equivalent to Equation (15). For any $\mathcal{H}_1, \mathcal{H}_2$,

$$\|\mathcal{F}(\mathcal{H}_1) - \mathcal{F}(\mathcal{H}_2)\| = \left\|\frac{X^*}{\gamma}(\mathcal{H}_1 - \mathcal{H}_2)\right\| \leq \frac{\|X^*\|}{\gamma}\|\mathcal{H}_1 - \mathcal{H}_2\|,$$

and since $\frac{\|X^*\|}{\gamma} < 1$, $\mathcal{F}$ is a contraction. The Banach fixed-point theorem [89] then guarantees a unique fixed point. □

### B. Guarantees of Pre-Caching

We analyze the necessity of pre-caching high-heat regions. Given $\mathcal{V} = \{v \in V \mid \mathcal{H}_v^* \geq \theta\}$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Partition $\mathcal{E}$ into observed edges $\widetilde{\mathcal{E}}$ and unobserved edges $\mathcal{E} \setminus \widetilde{\mathcal{E}}$. For each $(u, v) \in \widetilde{\mathcal{E}}$, define transition probability as

$$T_{uv} = \frac{A_{uv}}{\sum_{w \in \mathcal{N}_u^{out}} A_{uw}}, \quad T_{obs} = \min_{(u,v) \in \widetilde{\mathcal{E}}} T_{uv},$$

so that $T_{uv} \geq T_{obs}$. For $(u, v) \in \mathcal{E} \setminus \widetilde{\mathcal{E}}$, impose $T_{uv} \geq T_{min}$.

**Lemma 1** (Edge-Visit Probability Lower Bounds). *For any* $(u, v) \in \mathcal{E}$, *the visit probability* $\mathcal{P}_{uv} = \mathcal{H}_u^* \cdot T_{uv}$ *satisfies*

$$\mathcal{P}_{uv} \geq \begin{cases} \theta \cdot T_{obs}, & (u, v) \in \widetilde{\mathcal{E}}, \\ \theta \cdot T_{min}, & (u, v) \in \mathcal{E} \setminus \widetilde{\mathcal{E}}. \end{cases}$$

**Proof Sketch.** Since $u \in \mathcal{V}$ implies $\mathcal{H}_u^* \geq \theta$ and $\mathcal{P}_{uv} = \mathcal{H}_u^* \cdot T_{uv}$, substituting the respective lower bound for $T_{uv}$ yields the result. □

**Theorem 2** (Pattern-Visit Guarantee). *Let* $ph$ *be a path of* $n$ *edges in* $\mathcal{E}$, *and define the observed-edge ratio* $r = |ph \cap \widetilde{\mathcal{E}}|/n$. *Then the path visit probability satisfies* $\mathcal{P}_{ph} \geq \jmath = \theta^n \cdot T_{obs}^{rn} \cdot T_{min}^{(1-r)n}$. *Moreover, for a pattern* $p = \{ph_1, \ldots, ph_m\}$, *one obtains* $\mathcal{P}_p \geq \prod_{j=1}^{m} \jmath_j$.

**Proof Sketch.** The visit probability along a path $ph$ is

$$\mathcal{P}_{ph} = \prod_{i=1}^{n}\mathcal{P}_{u_i v_i} = \prod_{i=1}^{n}(\mathcal{H}_{u_i}^* T_{u_i v_i}).$$

Applying Lemma 1 to each edge yields the lower bound $\jmath$.

For a pattern $p = \{ph_1, \ldots, ph_m\}$, we invoke a worst-case independence relaxation. Since the events "visiting $ph_j$" are positively correlated, the FKG inequality implies:

$$\mathcal{P}\left(\bigcap_{j=1}^{m} ph_j\right) \geq \prod_{j=1}^{m}\mathcal{P}(ph_j) \geq \prod_{j=1}^{m}\jmath_j.$$

Thus $\mathcal{P}_p \geq \prod_{j=1}^{m}\jmath_j$, completing the proof. □

### C. Complexity Discussion

This section explains how each component of GeoLayer acts in concert to shrink the effective complexity of solving the objective in Equation (6).

The joint BIP formulation over the variable set $\{\delta, \sigma, \rho\}$ couples replica placement and routing across both DCs and query patterns, which is theoretically NP-hard. The goal of GeoLayer is not to compute an exact global optimum, but rather to reduce effective problem complexity through structured decomposition:

(i) **Latency-driven sinking shrinks candidates.** Let $\mathcal{C}_k$ be the set of clusters at layer $L_k$. The sinking operation (Algorithm 1) confines each pattern $p$ to cluster $C \in \mathcal{C}_k$, shrinking candidates from $|D|$ to $|C|$. Accordingly, the decision space contracts from

$$O(|I||D| + |P||D|) \longrightarrow \sum_k \sum_{C \in \mathcal{C}_k} O(|I_C||C| + |P_C||C|),$$

where $I_C$ and $P_C$ denote the subsets of items and patterns confined to cluster $C$ (by sinking). Under this clustering, global max-latency constraints are satisfied within each layer, eliminating cross-layer coupling in the optimization.

(ii) **Cluster-level separability weakens objective coupling.** Within each cluster, WAN bandwidth is far more homogeneous. Hence, the heterogeneity-driven coupling term in $C^{(A)}$ (Section III-A, Equation (5)) is largely attenuated intra-cluster, making the total objective approximately additive across clusters and enabling parallel per-cluster solving.

| Symbol | Definition |
|---|---|
| $I, D, P,$ | Sets of data items ($I$), geo-distributed DCs ($D$), and read request patterns ($P$) |
| $p$ | A read request pattern which involves multiple data items in one read transaction, $p \in P$ |
| $s_x$ | Size of data item $x$ |
| $c_d^{store}, c_d^{read}, c_d^{write}, c_{yd}^{net}$ | *Costs:* storage cost at DC $d$ per unit size; read (GET) processing cost at DC $d$; write (PUT) processing cost at DC $d$; network transmission cost per unit size from DC $y$ to DC $d$ |
| $\delta_{xd}, \sigma_{xyd}, \rho_{pyd}$ | *Binary indicators:* $\delta_{xd}$ indicates whether item $x$ has a replica in DC $d$; $\sigma_{xyd}$ indicates whether a read request for $x$ from DC $y$ is served by DC $d$; $\rho_{pyd}$ indicates whether a pattern $p$ read from DC $y$ is routed to DC $d$ |
| $R_{xy}, W_{xy}, R_{py}$ | *Request rates:* $R_{xy}$ is the read rate for item $x$ from DC $y$; $W_{xy}$ is the write rate for item $x$ from DC $y$; $R_{py}$ is the read rate for pattern $p$ from DC $y$ |
| $V_d, E_d, V^B, E^B$ | Vertex set and edge set in DC $d$ ($V_d$, $E_d$); boundary-vertex set and cross-partition edge set ($V^B$, $E^B$) |
| $\alpha, \gamma, \beta$ | Diffusion intensity; heat decay rate; external heat-source injection coefficient. |
| $BS, \mathcal{C}$ | Bridge subgraph and a cluster |

(iii) **Overlap-centric placement lowers granularity.** For non-replicated patterns, we further decompose them into disjoint overlap regions. At layer $L_k$, let $\mathcal{C}_k$ be the set of clusters. For a cluster $C \in \mathcal{C}_k$, let $BS_C$ denote its candidate bridge subgraphs and $\{\omega\}_\mathcal{C}$ the set of disjoint regions. The DHD model provides a score for each region $\omega$, based on which a single-criterion greedy assignment is applied. The per-region cost is

$$O\big(|BS_C| + T_{\mathrm{DHD}}(C)\big),$$

where $T_{\mathrm{DHD}}(C)$ is the time required for the diffusion to reach a steady state on cluster $C$. Let $E_C^B$ be the set of cross-partition edges in $C$ used by the DHD computation and candidate filtering. The total complexity of the placement phase is thus

$$O\left(\sum_k \sum_{C \in \mathcal{C}_k} \left(|E_C^B| + |\{\omega\}_\mathcal{C}| \cdot \big(|BS_C| + T_{\mathrm{DHD}}(C)\big)\right)\right).$$

(iv) **Stepwise routing prunes the search tree.** Routing is handled in stages. In the online mode, bottom-up greedy selection within each layer quickly prunes participating DCs by maximizing coverage before escalating to upper layers. In the offline mode, routing first performs top-down localization and then bottom-up assembly; a local migration test (Section VI, Equation (14)) replaces an exponential global subset search across all DCs, thereby avoiding combinatorial explosion.

### D. Guarantees and Decision Rules

This section discusses how the components of GeoLayer optimize the objective function and presents the associated approximation guarantees.

Since the joint objective in Equation (6) is NP-hard and non-submodular, we do not claim a global constant-factor approximation to Equation (6). Instead, GeoLayer establishes: (i) feasibility by construction; (ii) monotone non-increase of a principled *surrogate objective* at each actionable step; and (iii) cluster-level near-additivity, which renders subproblems separable and enables parallel optimization.

*Feasibility.* Latency requirements of patterns are enforced by layering constraints. Capacity/budget checks are verified before committing replicas or routes. Any action violating a hard constraint is rejected.

*Monotone non-increase.* For each candidate action $\tau$, we evaluate the marginal change $\Delta\widehat{C}(\tau) = \Delta C^R(\tau) - \big(\Delta C^S(\tau) + \Delta C^W(\tau) + \Delta C^A(\tau)\big)$. Following Equation (13) in replica placement and Equation (14) in stepwise routing, we accept $\tau$ only if $\Delta\widehat{C}(\tau) < 0$; otherwise $\tau$ is a no-op. Hence every committed action is non-increasing in $\widehat{C}$, and strictly improving whenever the marginal gain is strictly negative.

*Cluster-level near-additivity and parallelism.* Within a layer, bridge subgraphs induce latency-homogeneous clusters, which attenuate the heterogeneity term in $C^{(A)}$ (Section III-A, Equation (5)). Consequently, the objective is approximately additive across clusters, rendering the subproblems separable and enabling parallel optimization.

*Approximation (surrogate-level).* Under standard cluster-local assumptions (monotone diminishing-returns read benefit and additive storage/write/association costs), the regional greedy admits the classic $(1 - 1/e)$ approximation to the surrogate objective under a cardinality constraint [90]. We emphasize that this guarantee applies to the surrogate, not to Equation (6) in full generality.

### E. Implementation

This section presents the full implementation of Overlap-Centric Replica Placement in GeoLayer. We decompose the process into three components: the pattern replica sinking operation (Algorithm 1), the Overlap-Centric Replica Placement (Algorithm 2), and the eviction of cold replicas during online query execution (Algorithm 3).

Algorithm 1 describes the pattern replica sinking process. The input pattern set $P$ is derived from historical access logs and carries request frequency and latency requirement features. The layer set $L$ is constructed using a fixed-threshold bucketing scheme (*e.g.,* 100 ms per interval). If layer $L_k$ meets the latency requirements of pattern $p$, then $p$ is placed into all bridge subgraphs in this layer that request $p$ (Line 6). Otherwise, $p$ will sink to layer $L_{k-1}$ and be replicated across all bridge subgraphs that request it. At layer $L_{k-1}$, its latency requirement will then be examined to determine whether it should continue sinking. (Lines 9-11).

## Algorithm 2: Overlap-Centric Replica Placement

**Input:** Layer hierarchy $L = \{L_0, ..., L_h\}$, initial pattern distribution $\mathbb{P}^{(k)} \leftarrow \{p \in \mathbb{P}^{(k)} \mid R_{pL_k} > W_{pL_k}\}$, decomposition set $\mathbb{D}^{(k)} \leftarrow \emptyset$, cluster set $\mathbb{C}^{(k)}$ for $L_k$, and bridge subgraph set $BS_p^{(k)}$ for requested $p$

**Output:** Final data placement decisions

**1 Function** PlaceReplicas()
**2**    $\mathbb{P} \leftarrow$ ReplicaSink()
**3**    **for** $k \leftarrow h$ **to** $0$ **do**
     ***Phase 1: Replication vs. Decomposition Decision.***
**4**      **if** $k > 0$ **then**
**5**        **foreach** $p \in \mathbb{P}^{(k)}$ **do**
**6**          $C_p^{rep} \leftarrow$ CompRepGain$(p, L_k)$
**7**          **if** $C_p^{rep} \geq 0$ **then**
**8**            FullReplicate$(p, L_{k-1})$
**9**            RecordPlacement$(p, L_{k-1})$
**10**            $\mathbb{P}^{(k-1)} \leftarrow \mathbb{P}^{(k-1)} \cup \{p\}$
**11**          **else**
**12**            // For decomposition
**13**            $\mathbb{D}^{(k-1)} \leftarrow \mathbb{D}^{(k-1)} \cup \{p\}$

     ***Phase 2: Allocation of Decomposed Regions.***
**14**      **if** $\mathbb{D}^{(k)} \neq \emptyset$ **then**
**15**        **foreach** $\mathcal{C}_j \in \mathbb{C}^{(k)}$ **do**
**16**          $\{r_i\} \leftarrow$ SplitByOverlap$(\mathbb{D}^{(k)}, \mathcal{C}_j)$
**17**          **foreach** $\omega \in \{r_i\}$ **do**
**18**            $C_\omega^{rep} \leftarrow$ CompRepGain$(\omega, L_k)$
**19**            **if** $C_\omega^{rep} \geq 0$ **then**
**20**              FullReplicate$(\omega, L_k)$
**21**              RecordPlacement$(\omega, L_k)$
**22**            **else**
**23**              **foreach** $b \in BS_p^{(k)}$ **do**
**24**                $h_b^\omega \leftarrow$ DHDModel$(b, \omega)$
**25**              $b^* \leftarrow \arg\max_b h_b^\omega$
**26**              PartialReplicate$(\omega, b^*)$
**27**              RecordPlacement$(\omega, b^*, L_k)$
**28**          **if** $k > 0$ **then**
**29**            $\mathbb{P}^{(k-1)} \leftarrow \mathbb{P}^{(k-1)} \cup \{\omega\}$

     ***Phase 3: Pre-Caching for Base Layer.***
**30**      PrecachehHotRegions$(L_0, \theta)$

---

## Algorithm 3: Online Replica Eviction

**Input:** Cache graph $G^H = (\mathcal{V}, \mathcal{E})$, vertex heat set $\mathcal{H}$, access stream $\mathcal{Q}$, and cold threshold $\theta_c$

**Output:** Eviction decisions

**1 Function** OnlineEvict()
**2**    **while** *serving online queries* **do**
     ***External Heat Injection***
**3**      **foreach** *access* $(q, \mathcal{V}_q)$ *in* $\mathcal{Q}$ **do**
**4**        **foreach** $v \in \mathcal{V}_q$ **do**
**5**          $\mathcal{H}_v \leftarrow \mathcal{H}_v + R_v^q$

**6**      $\mathcal{H} \leftarrow$ DHDModel$(G^H, \mathcal{H})$
     ***Cold Identification and Eviction***
**7**      **foreach** $v \in \mathcal{V}$ **do**
**8**        **if** $\mathcal{H}_v < \theta_c$ **then**
**9**          EvictReplica$(v)$
**10**          UpdateRoutingTable$(v)$

---

## Algorithm 1: Pattern Replica Sinking Process

**Input:** Layer set $L = \{L_0, L_1, \ldots, L_h\}$ with latency intervals $L_k \mapsto [t_{k-1}, t_k)$, historical pattern set $P$ where $\forall p \in P$ has latency requirement $\eta_p \cdot \Gamma_{\max}$, a initial pattern distribution $\mathbb{P}^{(k)}$, $k \in \{h, \ldots, 1\}$

**Output:** Final pattern distribution $\mathbb{P}$

**1 Function** ReplicaSink()
**2**    **for** $k \leftarrow h$ **to** $1$ **do**
**3**      **while** $\mathbb{P}^{(k)} \neq \emptyset$ **do**
**4**        $p \leftarrow \mathbb{P}^{(k)}$.pop()
**5**        **if** $\eta_p \cdot \Gamma_{\max} \in (t_{k-1}, t_k)$ **then**
**6**          PlaceReplica$(p, \mathbb{P}^{(k)}, L_k)$
**7**        **else**
**8**          // Latency violation - sinking
**9**          FullReplicate$(p, L_{k-1})$
**10**          RecordPlacement$(p, L_{k-1})$
**11**          $\mathbb{P}^{(k-1)} \leftarrow \mathbb{P}^{(k-1)} \cup \{p\}$

**12**    **return** $\mathbb{P}$

---

After the pattern replica sinking is completed, GeoLayer performs the overlap-centric replica placement strategy. This strategy quantifies the benefit of data replication and leverages a directed heat diffusion model to make the final placement decisions. Algorithm 2 outlines the replica placement process. Initially, any external pattern $p$ retrieved at a given layer $L_k$ is included in the replication procedure if it satisfies $R_{pL_k} > W_{pL_k}$. The process begins with the layer-by-layer sinking of pattern replicas, guided by latency requirements (Line 2). Next, the data replication gain $C_p^{rep}$ is computed to determine whether the pattern should be fully replicated or decomposed in the next lower layer (Lines 5–13). If $C_{py}^{rep} < 0$, the pattern is decomposed across the corresponding BSs (Line 13). For pattern decomposition, patterns are first partitioned into a set of disjoint regions based on their overlapping components (Line 16). The replication gain $C_\omega^{rep}$ of each region is then evaluated to determine whether the region should be replicated across $BS$s or enter a competitive allocation process (Line 18). For regions subject to competition, each $BS$ executes the DHD model (Line 24), and region ownership is determined based on the resulting heat distribution (Line 25). This process proceeds layer by layer until all data items reach the DCs at layer $L_0$ (Line 29). Finally, high-heat regions are pre-cached in the DCs at layer $L_0$ (Line 30).

In addition, Algorithm 3 describes the replica eviction process during online query execution. For each access batch, vertex access frequencies are injected into the heat system and converted into vertex heat values (Line 5). The heat is subsequently diffused and updated via the DHD model (Line 6). Replicas identified as cold data are evicted, and the routing information is updated accordingly (Lines 8–10).