Nova: Scalable Streaming Join Placement and Parallelization in Resource-Constrained Geo-Distributed Environments

Xenofon Chatziliadis
x.chatziliadis@tu-berlin.de
BIFOLD, TU Berlin
Germany

Eleni Tzirita Zacharatou
eleni.tziritazacharatou@hpi.de
HPI, Uni Potsdam
Germany

Samira Akili
samira.akili@tu-berlin.de
BIFOLD, TU Berlin
Germany

Alphan Eracar
eracar@campus.tu-berlin.de
BIFOLD, TU Berlin
Germany

Volker Markl
volker.markl@tu-berlin.de
BIFOLD, TU Berlin, DFKI
Germany

Abstract

Real-time data processing in large geo-distributed applications, like the Internet of Things (IoT), increasingly shifts computation from the cloud to the network edge to reduce latency and mitigate network congestion. In this setting, minimizing latency while avoiding node overload requires jointly optimizing operator replication and placement of operator instances, a challenge known as the Operator Placement and Replication (OPR) problem. OPR is NP-hard and particularly difficult to solve in large-scale, heterogeneous, and dynamic geo-distributed networks, where solutions must be scalable, resource-aware, and adaptive to changes like node failures. Existing work on OPR has primarily focused on single-stream operators, such as filters and aggregations. However, many latency-sensitive applications, like environmental monitoring and anomaly detection, require efficient regional stream joins near data sources.

This paper introduces Nova, an optimization approach designed to address OPR for join operators that are computable on resource-constrained edge devices. Nova relaxes the NP-hard OPR into a convex optimization problem by embedding cost metrics into a Euclidean space and partitioning joins into smaller sub-joins. This new formulation enables linear scalability and efficient adaptation to topological changes through partial re-optimizations. We evaluate Nova through simulations on real-world topologies and on a local testbed, demonstrating up to 39× latency reduction and 4.5× increase in throughput compared to existing edge-centered solutions, while also preventing node overload and maintaining near-constant re-optimization times regardless of topology size.

Keywords

Distributed stream processing, Edge computing, Operator placement, Operator replication, Stream joins, IoT

1 Introduction

Internet of Things (IoT) applications employ sensors in distributed locations, generating large amounts of streaming data [42, 44]. Combining streams from different sources typically requires transmitting data to a centralized cloud [6, 16, 81], introducing significant communication overhead and latency. This is particularly inefficient when lightweight join operations based on simple predicates could instead be executed on resource-constrained edge nodes [29, 63, 72]. Such joins are common in real-time IoT applications, including the energy sector [56], smart manufacturing [61], healthcare [53], and smart cities [3, 42, 44] (e.g., joining traffic and weather streams in a smart city to dynamically adjust speed limits [84]). In this paper, we focus on a geo-distributed environmental monitoring scenario inspired by the DEBS 2021 Grand Challenge [68]. The challenge studies large-scale air-quality monitoring using geographically distributed sensors, which requires correlating time-windowed measurements across regions to detect environmental changes. We consider a similar geo-distributed environmental sensing workload using data from Sensor.Community [62] where pressure and humidity streams from sensors in multiple regions are continuously joined by region identifier and time window to detect regional climate anomalies. Executing these joins centrally quickly leads to high latency and overload, whereas placing and parallelizing them across edge and fog nodes keeps latency low while respecting resource constraints. We use this scenario as our main end-to-end use case throughout the paper and revisit it in our evaluation.

Challenges in the Edge-Cloud Continuum. To address the limitations of cloud-centered computation, SPEs such as NebulaStream [82] have emerged, enabling low-latency processing by distributing tasks across the entire edge-cloud continuum. This design aligns with the osmotic computing paradigm [73], which dynamically balances computation between cloud and edge resources based on network conditions and resource availability. A key mechanism in this approach is the replication and strategic placement of computational operators (e.g., filters, joins, and aggregations) across nodes to meet objectives such as latency minimization and bottleneck avoidance. This joint decision-making task is referred to as Operator Placement and Replication (OPR) [8, 11]. OPR is an NP-hard problem, as it generalizes the well-known General Assignment Problem [9]. Solving OPR requires accounting for a variety of complex factors, including network topology, resource constraints, and operator dependencies. These challenges are exacerbated in osmotic computing environments, which exhibit three key characteristics: (1) large scale, potentially encompassing millions of nodes [82]; (2) high heterogeneity, with devices ranging from resource-constrained edge nodes (e.g., Raspberry Pis) to powerful cloud servers [27]; and (3) extreme dynamics of fluctuating data rates, and frequent changes in the network due to mobility or IP-level routing [18]. As a result, effective OPR strategies must be scalable, resource-aware, robust to frequent changes, and capable of supporting partial re-optimization [21, 29].

Limitations of Existing Work. Existing work on OPR in osmotic computing mainly spans two domains: distributed stream processing (DSP) and wireless sensor networks (WSNs). In DSP,

EDBT '26, Tampere (Finland)

10.48786/edbt.2026.35

OPR is typically solved using Integer Linear Programming (ILP), which scales poorly and requires full re-computation after system changes [8]. In contrast, WSN research focuses on adaptive methods that parallelize tuple-wise joins across clustered or tree-based overlays toward the nearest base station [47, 78]. However, these approaches are unsuitable for DSP workloads, as they cannot span across geo-distributed deployments and are resource-agnostic, leading to node overloading and high latencies [47, 78].

Recently, we introduced NEMO [11], a scalable, resource-aware, and adaptive solution for the placement of decomposable aggregation operators in osmotic environments. Extending NEMO to joins is non-trivial for three reasons: (1) Joins amplify data rather than reduce it. NEMO parallelizes aggregation by computing partial results hierarchically, reducing data volume at each level. Joins, however, can produce outputs larger than their inputs—in the worst case, a cross-product—making this tree-based parallelization inapplicable. (2) Parallelizing aggregation requires only partitioning the input stream. Parallelizing joins, however, requires partitioning one input while broadcasting the other to all replicas, creating a trade-off between computational load and network traffic that NEMO does not address. (3) Unlike aggregations, joins cannot be computed hierarchically. Each replica operates independently, requiring a different placement approach. As a result, a robust, resource-aware, adaptive, and scalable solution to the OPR problem for join operators in DSP remains an open challenge.

Contributions. This paper presents Nova, an optimization approach for streaming joins in heterogeneous, geo-distributed environments. Building on the cost-space abstraction of NEMO, Nova introduces: (1) a cost model to handle data amplification, (2) a bandwidth-aware partitioning strategy to control network overhead, and (3) a geometric median based placement approach to optimize per-partition latency for independent replicas. Nova jointly optimizes placement, replication degree, and stream partitioning for 2-way joins, achieving linear-time scalability, rapid re-optimization, and resilience to dynamic network behavior. Specific contributions are as follows:

(1) *Problem Definition*: Section 2 introduces our stream processing model and formalizes the Operator Placement and Parallelization (OPP) problem. OPP extends OPR with the identification of optimal stream partitioning, which is required to solve OPR for join operators.
(2) *Nova Optimization Framework*: Section 3 presents our core contributions: (1) To address data amplification, we introduce a decomposition strategy based on the join matrix that partitions joins into sub-joins according to data locality and resource availability, enabling horizontal parallelization near sources. (2) To handle the independence of join replicas, we formulate placement as a convex geometric median problem. (3) To satisfy bandwidth constraints, we develop a partitioning scheme that jointly optimizes replication and network traffic. (4) To support dynamic environments, we present a re-optimization strategy that adapts to topology changes and workload shifts without full recomputation of the placement. Collectively, these techniques address optimization challenges inherent to join placement and parallelization in resource-constrained geo-distributed environments.
(3) *Experiments*: In Section 4, we compare Nova against heuristics used in SPEs and adaptive join techniques used in WSNs. Through simulations, we demonstrate that Nova remains efficient even on topologies with up to a million nodes, while supporting near-constant re-optimizations in under one second. These results highlight Nova's potential as an optimization approach for future SPEs designed to operate at this scale. Furthermore, we integrate Nova into NebulaStream's optimizer [82] and evaluate Nova's placement performance on a local heterogeneous computing cluster. Unlike state-of-the-art approaches that are resource-oblivious and may overload nodes, Nova avoids over-utilization entirely. Nova achieves hereby latency improvements of 4.2–39× and 4.5× higher throughput compared to existing approaches.

2 Preliminaries

This section introduces the fundamental concepts and definitions of the Nova approach. We first define the semantics of a stream processing application and stream join in Section 2.1. Next, we describe how Nova models latency and resource metrics for placement decisions in Section 2.2. Finally, in Section 2.3, we formulate the operator placement and parallelization problem for join operators that Nova addresses.

2.1 Stream Processing Model

A stream processing application typically consists of one or more user-defined queries compiled into a directed graph of operators interconnected by data streams. Operators are self-contained units performing specific functions, while streams are unbounded sequences of data tuples. To enable meaningful computations, data streams are typically discretized using window operators [71].

Stream Model. Let Ω be the universe of operators. Each operator $\omega = (\mathrm{id}, r, \rho, L_{in}, L_{out}) \in \Omega$ is characterized by several attributes. The attribute id represents the operator's unique identifier, while r denotes its replication number. Together, id and r form a unique identifier for each operator instance. The attribute ρ specifies the total number of operator instances that share the same id. The set of incoming streams is defined by $L_{in} = \{s_1, \ldots, s_n\}$, whereas the set of outgoing streams is given by $L_{out} = \{t_1, \ldots, t_n\}$. Operators that exclusively produce streams, i.e., those for which $L_{in} = \emptyset$, are called *sources*, whereas those that only consume streams, i.e., those for which $L_{out} = \emptyset$, are referred to as *sinks*. Two operators in a given Ω are considered connected if the output stream of one operator serves as the input stream of another. We denote the set of connected operators in Ω as $con(\Omega) = \{(\omega_i, \omega_j) \in \Omega^2 \mid \exists t \in \omega_i.L_{out}\ [t \in \omega_j.L_{in}]\}$. The dot notation provides attribute access, e.g, $\omega.L_{in}$, represents the incoming streams of the operator ω.

Logical Plan. We formally define the initial input plan representing a DSP application as a logical plan $\Omega_{log} \subseteq \Omega$. Each operator $\omega_{log} \in \Omega_{log}$ has a replication factor of one, i.e., $\omega_{log}.\rho = 1$.

Stream Join. The stream join operator combines tuples from two streams based on a join condition. Given two streams i and j, a stream join operator produces a new stream $i \bowtie j$ with tuples that satisfy a join condition Θ. Formally, the join is expressed as:

$$i \bowtie j = \{(x, y) \mid x \in i, y \in j, \Theta(x, y)\}.$$

The join condition Θ in our work represents an operation with low computational complexity, e.g., an equality join with $x.A = y.B$. We note that a stream join over windows requires tuples to satisfy both the join condition and window boundary constraints,

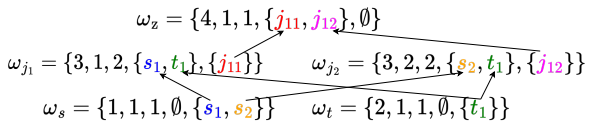


Figure 1: Formalization of a decomposable join $s \bowtie t$, where each join replica $\omega_{j_1}, \omega_{j_2}$ processes a partition of streams s and t, forwarding partial results to the sink.

such as overlapping timestamps for time-based windows. We illustrate our model for joining streams $s \bowtie t$ in Figure 1.

Join Matrix. We represent permissible joins between stream partitions $S = \{s_1, \ldots, s_m\}$ and $T = \{t_1, \ldots, t_n\}$ using a binary $m \times n$ matrix M, where $M_{p,q} = 1$ means s_p can join with t_q. For predefined conditions (e.g., joins on spatial region identifiers), M is known beforehand. If join validity is uncertain, M is initialized as dense (all ones), requiring evaluation of all stream pairs. Section 3.6 explains how Nova handles uncertainties and updates to M at runtime. Importantly, M indicates only which stream partitions can join; tuple-level validity must still be checked during execution. Since we focus on two-way joins, M captures only direct joinability without considering transitive joins or join ordering.

2.2 Resource Model

We model the topology as a directed graph $G_T = (V, E)$, where V is the set of nodes (e.g., servers or sensors) and E the communication links between them. The processing cost of a streaming join operator depends primarily on input data rates, window size, and join predicate complexity, than on the window type [30, 84]. On resource-constrained edge devices, this behavior simplifies further. Our empirical results for edge-computable lightweight joins (e.g., equi-joins) indicate that compute demand is mainly driven by tuple arrival rate, whereas end-to-end delay is dominated by network latency (cf. Section 4.7). We therefore adopt the following model. For an operator ω with input streams L_{in} and per-stream data rates $dr(\cdot)$, the required compute capacity is $C_r(\omega) = \sum_{s \in L_{in}(\omega)} dr(s)$. This capacity is benchmarked per node type and operator class in advance. Window size and predicate complexity remain fixed for a given Ω_{log} and are omitted from the notation for brevity.

Path delay is approximated by the sum of link latencies along the route, as millisecond-scale network delays dominate the submicrosecond processing time of lightweight joins. A placement is feasible only if $C_r(\omega) \leq C_a(v)$ for every replica on node v and if the aggregate traffic on each link does not exceed $b(v_i, v_j)$ when bandwidth budgets are enforced (cf. Eqs. 2 and 4). Nova is agnostic to whether links are physical paths or logical channels because the cost-space embedding (cf. Section 3.2) abstracts routing details.

2.3 Problem Formulation

Operator Placement. Operator placement (OP) maps each operator $\omega \in \Omega_{log}$ to a node $v \in V$ in a topology using a mapping function $f_m \colon \Omega \mapsto V$ [7]. Each operator must be assigned to exactly one node, which is enforced by the exclusivity constraint, i.e., $\forall \omega \in \Omega_{log}\ [\exists! v_j \in V\ [f_m(\omega) = v_j]]$. Some operators, like sources and sinks, are fixed to specific nodes due to dependencies, while others, like joins, can be placed flexibly across nodes. In the classical OP formulation, the optimization objective can vary (e.g., minimizing bandwidth usage, balancing load). In this work, we focus on latency minimization as the primary optimization objective.

Operator Placement and Replication. Cardellini et al. [8] show that the general operator placement problem can be extended to the operator placement and replication problem (OPR), which jointly determines both the placement of operators and their degree of replication to optimize a given objective while satisfying resource constraints. Since OPR generalizes OP, it remains NP-hard.

Operator Placement and Parallelization. In this work, we further extend OPR by considering *stream partitioning* alongside placement and replication. We define this as the Operator Placement and Parallelization problem (OPP). While OPR optimizes placement and replication of whole operators, OPP enables fine-grained parallelization at the stream level via partitioning, allowing a single operator's input stream to be divided across multiple replicas.

In OPP, parallelization is controlled by a function $f_p \colon \Omega_{log} \mapsto 2^{\Omega}$, which maps each operator $\omega_{log} \in \Omega_{log}$ to a set of parallel instances $\omega'_1, \ldots, \omega'_\rho$. Each instance processes a partition of the operator's input/output streams, ensuring $\bigcup_{\omega' \in f_p(\omega_{log})} \omega'.L_{out} = \omega_{log}.L_{out}$. Each instance of f_p defines a parallelization strategy, yielding a parallelized logical plan $\Omega'_{log} = \bigcup \omega_{log} \in \Omega_{log} f_p(\omega_{log})$. Parallelization reduces resource demands per replica, since partitions lower input data rates ($dr(l') \leq dr(l)$). As OPP generalizes OPR through additional partitioning, the problem remains NP-hard.

Optimization Objective. Given a logical plan Ω_{log} and a topology $G_T = (V, E)$, the goal of OPP is to jointly determine:

(1) A parallelization strategy f_p, which defines a parallelized logical plan $\Omega'_{log} \in 2^{\Omega}$, selecting a subset of operator replicas from the full search space Ω.
(2) A placement strategy f_m, mapping each operator replica $\omega' \in \Omega'_{log}$ to a node $v \in V$.

The objective is to select f_p and f_m such that 1) all constraints are satisfied, and 2) the total latency between connected operators is minimized. Formally, OPP can be expressed as finding:

$$\Omega^{\star}_{log} = \arg\min_{\Omega'_{log} \subseteq 2^{\Omega}} \sum_{(\omega'_i, \omega'_j) \in con(\Omega'_{log})} d(f_m(\omega'_i), f_m(\omega'_j)), \quad (1)$$

subject to the following constraints:

1) Resource Capacity Constraint. The required capacity of operators must not exceed their nodes' available resources:

$$C_r(\omega') \leq C_a(f_m(\omega')), \quad \forall \omega' \in \Omega^{\star}_{log}. \quad (2)$$

2) Resource Availability Constraint. The available resources of assignable nodes must be above a threshold C_{min}.

$$C_a(v_i) \geq C_{min}, \quad \forall v_i \in img(f_m). \quad (3)$$

3) Network Bandwidth Constraint. Since the required capacity $C_r(\omega')$ is defined as the sum of incoming data rates, it also represents the bandwidth utilization. Thus, the bandwidth demand must remain within the available bandwidth threshold t_b:

$$C_r(\omega') \leq t_b, \quad \forall \omega' \in \Omega^{\star}_{log}. \quad (4)$$

This OPP formulation minimizes the sum of all end-to-end latencies while respecting constraints. An alternative objective

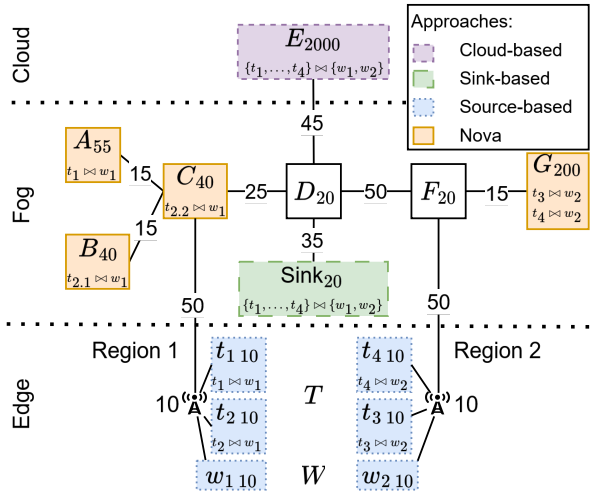


Figure 2: Running example to demonstrate Nova for joining geo-distributed sensor readings of $\{t_1, \ldots, t_4\} \in T$ with $\{w_1, w_2\} \in W$. Subscripts indicate node processing capacity, and edge labels denote network path latency.

would be to minimize the maximum end-to-end latency across partitions, corresponding to a min–max relay placement problem that optimizes only the slowest partition. We choose the min–sum objective for three reasons. First, it yields a geometric median formulation with a unique and stable optimum that can be computed efficiently. Second, it optimizes aggregate latency across all partitions rather than focusing solely on a single worst-case path. Third, it is significantly more robust to noise, estimation errors, and latency fluctuations in dynamic networks, since a single inaccurate or stale measurement can dominate a min–max objective.

3 The Nova Approach

This section introduces Nova, a scalable, adaptive, and resource-aware solution to the OPP problem for join operators in resource-constrained distributed stream processing. Algorithm 1 outlines Nova's workflow: Given the topology G_T, a logical plan Ω_{log}, and the join matrix M, Nova produces an operator-to-node mapping for a parallelized logical plan Ω'_{log}, including replicated operators with partitioned streams. Since solving the OPP over a *discrete* set of nodes is NP-hard (cf. Section 2.3), Nova instead reformulates the problem in a continuous space and tackles it by decomposing it into three heuristic phases, each solvable in linear time.

(1) **Phase I: Cost Space Construction**: The discrete topology is embedded into a continuous cost space by assigning each node $v \in V$ a coordinate vector $\hat{v} \in \mathbb{R}^d$ (line 2). This mapping enables an iterative approximation of the OPP in the continuous domain, which is required for Phase II. Section 3.2 provides details on Phase I.
(2) **Phase II: Virtual Join Placement**: In the cost space, the optimization objective of OPP reduces to identifying the geometric median, a convex problem that can be efficiently solved using iterative algorithms. This formulation avoids combinatorial search over discrete placements and enables efficient approximation of optimal operator positions (lines 3–4). Section 3.3 describes Phase II.
(3) **Phase III: Physical Replica Assignment**: After placement in continuous space, Nova maps the virtual solution back to the discrete topology while enforcing all resource constraints (lines 5–13). Section 3.4 describes Phase III.

Algorithm 1: The Nova approach

Input: G_T (topology), Ω_{log}(logical plan), M (join matrix)

1 $placements \leftarrow \{\}$;
2 $\hat{V} \leftarrow$ compute_coordinates(G_T);
3 $\Omega'_{log} \leftarrow$ resolve_operators(Ω_{log}, M);
4 $\hat{\Omega} \leftarrow$ compute_optima($\hat{V}, \Omega'_{log}$);
5 **foreach** $\omega' \in \Omega'_{log}$ **do**
6 **if** *is_pinned*(ω') **then**
7 $placements[\omega'] \leftarrow$ get_pinned_node(ω');
8 **end**
9 **else**
10 $result \leftarrow$ parallelize_and_place($\omega', \Omega'_{log}, \hat{V}, \hat{\Omega}$);
11 $placements$.update($result$);
12 **end**
13 **end**
14 **return** $placements, \Omega'_{log}$

To handle dynamic changes (e.g., node failures), Nova leverages the convexity of the cost space objective, enabling local re-optimization of affected operators without full recomputation. We discuss the re-optimization of Nova in Section 3.5. While Nova's cost space models latency by default, it is metric-agnostic and adaptable to other objectives. In Section 3.6 we discuss possible extensions.

3.1 Running Example

For clarity, we use a simplified environmental monitoring workload, shown in Fig. 2, as a running example throughout the rest of Section 3. Two sensor streams, a pressure stream $T = \{t_1, t_2, t_3, t_4\}$ and a humidity stream $W = \{w_1, w_2\}$, are generated by geographically distributed *Sensor.Community* nodes [62] and joined on region identifier and time window before being forwarded to a local sink to detect regional climate anomalies. The topology follows an edge–fog–cloud pattern with sources at the edge, several medium-capacity fog nodes (A–G), a high-capacity cloud node (E), and a local sink. Each source emits at 25 Hz, edge labels denote network latency (ms), and node subscripts indicate processing capacity (tuples/s). The join decomposes into two region-specific sub-joins: $T \bowtie W = \big((t_1 \bowtie w_1) \cup (t_2 \bowtie w_1)\big) \cup \big((t_3 \bowtie w_2) \cup (t_4 \bowtie w_2)\big)$.

We contrast Nova with three baselines. In a source-based or sink-based placement, all six sources together generate $6 \times 25 = 150$ tuples/s, but each source processes only 10 tuples/s and the sink 20 tuples/s, leading to overload and backpressure. A cloud-based strategy places the join on node E, where Region 1 traffic must traverse C and D, and Region 2 traffic F and D, which yields path delays of roughly 130 ms and 155 ms, respectively. Returning results to the sink adds about 100 ms, so the maximal end-to-end latency reaches about 275 ms. Nova decomposes the join into two region-specific sub-joins, placing Region 2's sub-join on node G and parallelizing Region 1's sub-join across nodes A, B, and C by broadcasting w_1 and partitioning t_2 into t_{21} and t_{22}. This ensures that for each operator ω on node v, the condition $C_r(\omega) \leq C_a(v)$ is met, reducing end-to-end latency to roughly 150 ms for Region 1 and 175 ms for Region 2. The following subsections explain how Nova derives this plan.

3.2 Cost Space Construction

Following NEMO [11, 70], the first phase is a pre-processing stage that maps the discrete network topology to a continuous cost space (Algorithm 1, line 2). Conceptually, Nova represents network latencies by a symmetric matrix $A \in \mathbb{R}^{n \times n}$, where each entry A_{ij} denotes the end-to-end latency between nodes v_i and v_j. The corresponding embedding can be formulated as the multidimensional scaling (MDS) problem of finding an embedding matrix $\hat{G}_T \in \mathbb{R}^{n \times d}$ whose induced distance matrix $D(\hat{G}_T)$ approximates A by minimizing

$$\min_{\hat{G}_T \in \mathbb{R}^{n \times d}} \|D(\hat{G}_T) - A\|_F^2. \tag{5}$$

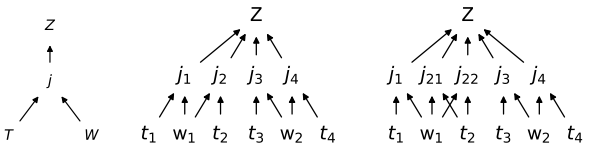
Here, each row of $\hat{G}_T$ represents the coordinates of a node in the cost space, with the ith row of $\hat{G}_T$ corresponding to the coordinates of the ith node $v_i \in V$. We denote the coordinates of node v_i as $\hat{v}_i$ and represent the set of all node coordinates as $\hat{V} = \{\hat{v}_1, \ldots, \hat{v}_n\}$.

For very large latency matrices A, Nova does not solve the dense MDS formulation. Instead, Nova computes node coordinates using Vivaldi [19], which maintains a small neighbor set $\mathcal{N}(i)$ for each node v_i and materializes latency entries only for pairs (i, j) with $j \in \mathcal{N}(i)$, where $|\mathcal{N}(i)| = m \ll |V|$. Vivaldi thus acts as a stochastic solver for the above MDS objective over this neighborhood-induced sparse distance matrix and directly produces the coordinate set $\hat{V} = \{\hat{v}_1, \ldots, \hat{v}_n\}$ used by Nova, while avoiding quadratic measurement and computation overhead.

Benefits. Nova relies on the cost space to efficiently solve the OPP using heuristic methods. Beyond enabling placement optimization, the embedding offers several key advantages: 1) It abstracts physical network paths, making it well-suited for geo-distributed systems with unknown or dynamic routing. 2) It increases robustness to continuous latency fluctuations, reducing the need for frequent re-optimization (cf. Section 4.5). 3) It significantly reduces the number of latency measurements and makes the cost-space construction scalable to large topologies.

Limitations. Euclidean embeddings assume that network latencies satisfy the triangle inequality, but real-world Internet latencies often violate this property, a phenomenon known as Triangle Inequality Violations (TIVs) [43]. Like other NCS-based approaches [19, 55], Nova's placement quality depends on the accuracy of estimated latencies, and severe TIVs can reduce placement quality. We evaluate the practical impact of TIVs in Section 4.4 by comparing estimated and measured latencies across real-world topologies. In practice, geo-distributed topologies are highly dynamic, and even exact measurements quickly become stale due to routing and network changes. Nova, therefore, emphasizes robustness to moderate estimation errors and continuous adaptation rather than strict accuracy.

Example. Figure 4 illustrates the Euclidean embedding of the example topology, with worker nodes shown as •, sources as ■, and the sink as ♦. The embedding is derived from the measured latencies in Figure 2. For instance, the latency from t_1 to C is 60 ms (10 ms to the base station and 50 ms to C), whereas the end-to-end latency from t_1 to the sink is 110 ms. Accordingly, the matrix A contains entries $A[t_1, C] = 60$ and $A[t_1, \text{sink}] = 110$. In the Euclidean space, t_1 and C are therefore placed close together, while t_1 appears farther from the sink. These distances guide the continuous optimization of the join operator locations (shown as ▲).



(a) Input **(b) Resolved operators** **(c) Partitioned streams**

Figure 3: Different stages of the logical operator plan in Nova.

3.3 Virtual Join Placement

In the second phase, Nova efficiently approximates the OPP problem by first solving it in the continuous cost space. The process consists of three steps: 1) source expansion, 2) pair-wise join replication, and 3) computation of the optimal placements of the resolved operators in the cost space. Steps 1 and 2 together form Nova's *resolving operators* function (cf. Algorithm 1, line 3).

Source Expansion. Nova begins by expanding the logical source streams in Ω_{log} into their corresponding physical streams located at data-producing nodes. A single logical stream may comprise arbitrarily many physical streams sharing the same schema. Thus, even if a join has only two logical inputs, the underlying number of physical streams can be very large.

Pair-wise Join Replication. Next, Nova creates replicas of the initial join operator $\omega_j \in \Omega_{log}$, one for each join pair defined by the join matrix M (cf. Section 2.1). Each replica corresponds to a specific pair of input partitions to be joined, resulting in an intermediate parallelized logical plan Ω'_{log}.

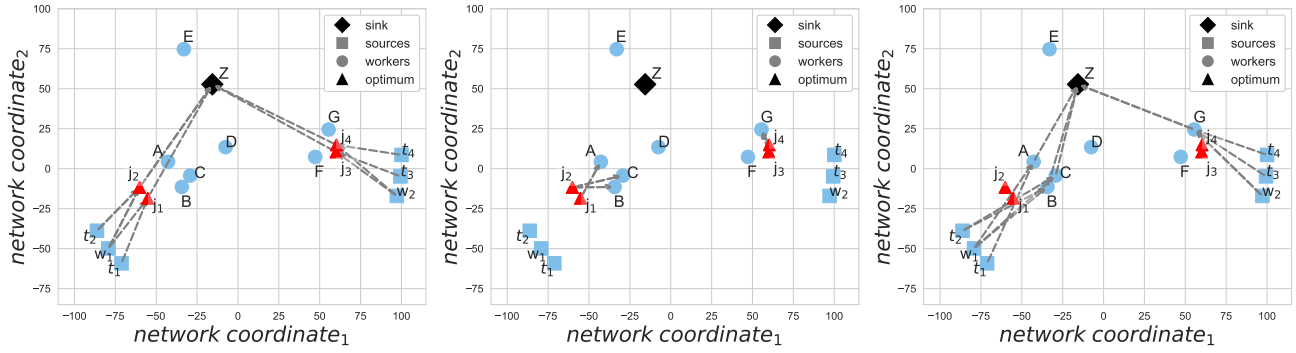
Placement in the Cost Space. After resolving the logical streams, Nova computes the placement of the join replicas in the cost space (Algorithm 1, line 4). This step conceptually resembles the virtual placement stage of NEMO [11], where operator locations are determined by minimizing a spring-energy objective. However, Nova's formulation differs significantly due to the structural properties of operator graphs for join replicas. In NEMO, aggregation operators form a hierarchy, necessitating coupled optimization where placement decisions at one level influence those at other levels. In contrast, join replicas are independent: each connects only to its two sources and the sink, with no inter-replica dependencies. As a result, the energy is decoupled across the replicas, and the objective reduces to the geometric median in the cost space [24].

Formally, for each join replica $\omega_j \in \Omega'_{log}$, Nova considers the set of coordinates $\hat{X} = \{\hat{v_1}, \ldots, \hat{v_n}\}$ representing the pinned nodes (i.e., sources and sinks) connected to ω_j. The goal is to find the virtual position $\hat{\omega}_j \in \mathbb{R}^d$ that minimizes the sum of distances to all nodes in $\hat{X}$. This position corresponds to each join replicas' geometric median, which is defined as:

$$\hat{\omega}_j = \arg\min_{y \in \mathbb{R}^d} \sum_{\hat{v} \in \hat{X}} d(\hat{v}, y) \tag{6}$$

The geometric median is a convex optimization problem, which we solve iteratively using gradient descent [60]. We denote the optimal virtual position of each operator ω_i as $\hat{\omega}_i$, and the set of all virtual placements as $\hat{\Omega} = \{\hat{\omega}_1, \ldots, \hat{\omega}_n\}$.

Example. In the first step (source expansion), Nova replaces the logical sources T and W (Figure 3a) with their physical data-producing nodes, expanding T into $t_1, \ldots, t_4$ and W into w_1, w_2 (Figure 3b). During pair-wise join replication (step 2), Nova uses



(a) Virtual Placement **(b) Physical Placement** **(c) Final Placement**

Figure 4: Progression of Nova illustrated on the example of Section 3.1, where the topology is embedded in a $\mathbb{R}^2$ Euclidean space.

the join matrix M to create one replica for each join pair, yielding two replicas j_1, j_2 for $t_1, t_2 \bowtie w_1$ in Region 1 and j_3, j_4 for $t_3, t_4 \bowtie w_2$ in Region 2, as shown in the parallelized plan Ω'_{log} (Figure 3b). Finally, in the virtual placement step, Nova computes for each replica the geometric median of the embedded coordinates of its connected nodes (i.e., the two sources and the sink). For example, the replica joining t_1 and w_1 minimizes the sum of distances to $\{\hat{t}_1, \hat{w}_1, \hat{\text{sink}}\}$ in the cost space. The resulting geometric median locations, shown as triangles ▲ in Figure 4a, represent the replicas' ideal positions before constraints are applied.

3.4 Physical Replica Assignment

In the final phase, Nova iterates over the operators in Ω'_{log}. Pinned operators (e.g., source, sink) are placed on their predefined nodes (Algorithm 1, line 7), while join operators (line 10) are parallelized. This phase addresses physical placement by mapping virtual operators to physical nodes under resource limits. Unlike NEMO [11], which reduces rates through decomposable aggregations, joins amplify traffic due to replication and key-based repartitioning and thus require different algorithms. Nova introduces per-link bandwidth constraints alongside node-capacity limits, decomposes joins into key-local and region-aligned subjoins, and applies bandwidth-aware partitioning to size replicas. It selects candidate nodes with a k-NN search in cost space and places replicas only if node capacity and link bandwidth permit. This approach yields horizontal replication near sources, which is fundamentally different from NEMO's vertical aggregation trees. The following subsections explain stream partitioning, candidate selection, and replica placement.

Bandwidth-Aware Stream Partitioning. Nova decomposes the left (l_l) and right (r_l) input streams of a join operator ω_j into disjoint partitions, creating multiple instances of $\omega_j \mapsto \{\omega'_{j_1}, \ldots, \omega'_{j_\rho}\}$ with reduced resource requirements to satisfy the capacity constraint (Equation 2). For instance, consider a join operator ω_j with $dr(l_l) = 25$ and $dr(r_l) = 25$ results in $C_r(\omega_j) = 50$. By partitioning each stream into 25 sub-streams, i.e., $l_l \mapsto \{l'_{l_1}, \ldots, l'_{l_{25}}\}$ and $r_l \mapsto \{r'_{l_1}, \ldots, r'_{l_{25}}\}$, the join can be parallelized across $25 \times 25 = 625$ replicas, where each replica ω'_{j_i} has $C_r(\omega'_{j_i}) = 2$. However, excessive partitioning increases network traffic from 50 tuples/sec (no partitioning) to $625 \times 2 = 1250$ tuples/sec (maximum partitioning), a 25× increase that can violate the bandwidth constraints (Equation 4).

To adhere to bandwidth constraints, Nova introduces a scaling factor σ ($0 \leq \sigma \leq 1$) that controls the degree of partitioning. Smaller σ increases partitioning, improving robustness to overload but increasing bandwidth overhead and Phase III complexity, while larger σ reduces these costs at the expense of higher overload risk.

Using σ, we define a maximum partition load threshold p_{max} for streams s and t as:

$$p_{max}(s,t) = \max(1, \sigma \cdot 0.5(dr(s) + dr(t))) \tag{7}$$

Each stream is then decomposed into partitions such that no partition exceeds $p_{\max}$ in data rate, ensuring that the resulting join replicas satisfy the capacity constraint (Equation 2). Assigning an equal weight of 0.5 to each data rate relative to σ improves resource utilization and reduces data movement compared to partitioning streams independently based on σ.

For example, consider a join operator ω_j with input streams s and t, where $dr(s) = 2$, $dr(t) = 10$, and $\sigma = 0.5$. Independent partitioning yields $s \mapsto \{s'_1, s'_2\}$, with $dr(s'_1) = dr(s'_2) = 1$, and $t \mapsto \{t'_1, t'_2\}$ with $dr(t'_1) = dr(t'_2) = 5$, requiring $C_r(\omega'_{j_i}) = 6$ for each join replica ω'_{j_i} of ω_j, and a total transfer of 24 tuples/sec. In contrast, our weighting sets $p_{max}(s,t) = \max(1, 0.5 \cdot 0.5 \cdot 12) = 3$, leaving s unpartitioned ($s \mapsto \{s'\}$) while splitting t into $t \mapsto \{t'_1, \ldots, t'_4\}$, with $dr(t'_i) = 3$ for $t'_i \in \{t'_1, t'_2, t'_3\}$, and $dr(t'_4) = 1$. This strategy reduces the individual resource requirements to $C_r(\omega'_{j_i}) = 5$ for the replicas ω'_{j_i} joining $s \bowtie \{t'_1, t'_2, t'_3\}$) and $C_r(\omega'_{j_i}) = 3$ for $s \bowtie t'_4$, while reducing the network transfer rate to 18 tuples/sec.

The scaling factor σ can be derived based on the data rates between streams s and t, subject to a bandwidth constraint t_b, by solving the following convex optimization problem:

$$\arg\min_{\sigma \in [0,1]} (\sigma \cdot 2 \cdot dr(s) \cdot dr(t) - t_b)^2. \tag{8}$$

Alternatively, operators may manually tune σ rather than strictly adhering to bandwidth constraints. As network load grows exponentially with increasing partition count, balancing partitioning efficiency with bandwidth utilization often yields better performance. In our experiments, we set $\sigma = 0.4$, which our empirical analysis showed provides a well-balanced trade-off across diverse workloads and topologies.

Candidate Selection. After partitioning a join operator's input streams, Nova places replicas on nodes to execute the decomposed joins. For each operator $\omega \in \Omega'_{\text{log}}$, Nova selects candidate nodes $V_{\text{knn}}(\omega) \subseteq V$ via a k-nearest-neighbors (kNN) search in the cost space, using an exact neighborhood search index (e.g., a k-d tree) for small topologies and an approximate Annoy-based index [4] for large ones. The k nodes whose coordinates are closest to the operator's virtual coordinates $\hat{\omega}$ are selected as candidates. To determine k, Nova computes the ratio of an operator's total required capacity to the median of available capacity per node. This ensures that the number of candidates considered for each operator automatically scales with its workload demand. Importantly, Nova does not attempt to use all available nodes. Only nodes that satisfy the compute and bandwidth constraints are considered as candidates, and replicas are placed on as many nodes as needed to avoid overload. Unused nodes remain idle.

Replication and Placement. In the final step, Nova assigns the partitioned input $l_l \mapsto \{l'_{l_1}, \ldots, l'_{l_m}\}$ and $r_l \mapsto \{r'_{r_1}, \ldots, r'_{r_n}\}$ of a join operator ω_j to $m \times n$ replicas, $\omega_j \mapsto \{\omega'_{j_{11}}, \ldots, \omega'_{j_{mn}}\}$. Each replica ω'_{j_i} of ω_j is then placed sequentially on a node $v_i \in V_{\text{knn}}(\omega_j)$ that satisfies $C_r(\omega'_{j_i}) \leq C_a(v_i)$ (Equation 2). If no node meets this requirement, Nova either (1) distributes the remaining replicas evenly across $V_{\text{knn}}(\omega_j)$, accepting a risk of overload, or (2) expands $V_{\text{knn}}(\omega_j)$ to include additional nodes, potentially increasing network overhead.

Example. Figure 4b shows the parallelization and placement of the join operators $\omega_j \mapsto j_1, \ldots, j_4$ in the cost space for the provided example (cf. Section 3.1), where all source streams have a data rate of 25. We use a k-NN search with $k = 2$, $C_{\text{min}} = 15$, and $\sigma = 0$. In the following we explain how Nova places these replicas. For clarity, $v|C_a(v)$ denotes a node v with available capacity $C_a(v)$, and $\omega|C_r(\omega)$ denotes an operator ω with required capacity $C_r(\omega)$. The search for j_1 returns $V_{\text{knn}}(j_1) = [A|55, B|40]$. Since $A|55 \geq j_1|50$, operator j_1 is placed on node A. The search for j_2 returns $V_{\text{knn}}(j_2) = [B|40, C|40]$, as $C_a(A)$ is now below C_{min}. Since $B|40 \leq j_2|50$, the operator j_2 is replicated 625 times due to $p_{\text{max}} = 1$, each representing a join between the partitions $t_{2_1} \bowtie w_{1_1}, \ldots, t_{2_{25}} \bowtie w_{1_{25}}$. Half of the join replicas are placed and merged on node B and the other half on node C. For j_3 and j_4, the searches return $V_{\text{knn}}(j_3) = [G|200, F|20]$ and $V_{\text{knn}}(j_4) = [G|150, F|20]$, resulting in both operators being placed on node G.

Figure 3c shows the final logical plan Ω'_{log} generated by Nova. Figure 4c depicts its placement in the cost space, illustrating the complete data flow from all sources $\{t_1, \ldots, t_4\} \bowtie \{w_1, w_2\}$ to the sink, with the join distributed across nodes A, B, C, and G.

3.5 Re-optimization and Adaptivity

Nova supports re-optimization without recomputing the full operator placement by distinguishing three types of changes: (1) adding sources or workers, i.e., nodes not yet involved in processing; (2) removing sources, join nodes, or workers; and (3) workload imbalances caused by changes in input rates or available resources. In NCS-based embeddings, changes in IP routes may alter latency estimates and shift node coordinates. To handle substantial changes, such as those introduced by mobile nodes [54], Nova removes the affected nodes and re-adds them to ensure consistent embeddings. Nova focuses exclusively on optimizing operator placement and parallelization, independent of deployment, fault-tolerance and monitoring mechanisms, which are addressed by complementary work in volatile, geo-distributed environments [5, 12–14, 41].

Topology Changes. When a node is added, Nova first determines its role and then applies the corresponding re-optimization procedure. For a new worker node, Nova collects latency measurements from a fixed set of neighboring nodes in the NCS and computes the node's coordinates in the cost space by minimizing the relative distance error, as described in Section 3.2. Since the neighborhood size is constant, this computation has fixed time complexity. Nova then updates the neighborhood search index to make the node available for subsequent candidate selection. If the added node is a source, Nova updates the logical plan and the join matrix M (cf. Figure 3b) and re-executes Algorithm 1 only for the affected sub-branch.

Before removing a node, Nova again identifies its role. Idle workers are removed directly from the cost space and the neighborhood search index. If the node is a source, Nova deletes its entry from the join matrix M and removes all associated join replicas and dependencies from both the logical plan and the placement mapping. If the node hosts a join operator, Nova retrieves the operator's precomputed virtual position in the cost space (cf. Section 3.3) and re-executes Phase III to re-place and parallelize only the affected operator (cf. Section 3.4).

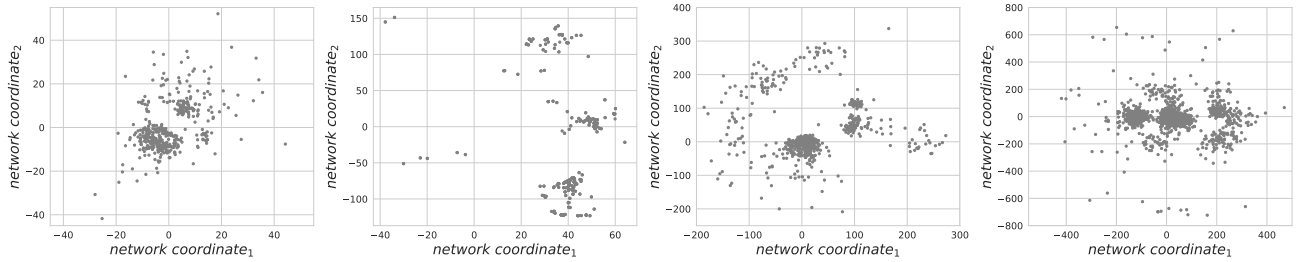
Workload Changes. When workload characteristics change, Nova identifies the cause of the change and rebalances only the affected sub-joins. If source data rates change, Nova undeploys all upstream replicas associated with the affected sources and re-executes physical placement. If available resources on a worker change, Nova undeploys all operators running on that worker and re-executes physical placement for the affected replicas. Because the virtual placements in the cost space remain optimal under such changes, Nova skips virtual placement and recomputes only the physical placement (Phase III) for the affected operators, as described in Section 3.4.

3.6 Extensibility

In this section, we describe modifications to Nova's optimization to support additional cost metrics and extensions to support more complex workloads.

Integrating Additional Cost Metrics. A key strength of Nova is that its cost-space formulation can be extended to support optimization over additional metrics beyond latency. Formally, Nova represents network characteristics through the symmetric matrix A (cf. Section 3.2), which encodes pairwise distances between nodes and is embedded into a Euclidean cost space. Additional distance-based metrics, such as energy consumption or monetary cost, can be incorporated by augmenting A with corresponding metric-specific distance matrices, following prior work by Pietzuch et al. [55]. These additional matrices can be embedded as additional dimensions using techniques such as MDS [35] or Vivaldi [19], as described in Section 3.2. During virtual placement (Phase II, Section 3.3), operators are placed by minimizing distances in this augmented cost space, such that the optimization objective in Equation 1 implicitly balances latency with the added metrics without changing the structure of the optimization problem.

Metrics that are not distance-based, such as memory capacity or specialized hardware availability, are integrated as constraints during physical placement (Phase III, Section 3.4). Specifically, the candidate set $V_{\text{knn}}(\omega)$ is filtered to include only nodes that satisfy the corresponding resource constraints (cf. Equations 2–3).



(a) FIT IoT Lab **(b) PlanetLab** **(c) RIPE Atlas** **(d) King**

Figure 5: Comparison of the network coordinate systems of the different topologies we used to evaluate Nova.

Complex Workloads. Nova focuses primarily on the placement of partitionable, geo-distributed joins, but its concepts naturally extends to richer operator graphs that include filters, aggregations, and multi-way joins. Regarding joins over highly diverse key spaces, Nova mitigates skew through the join matrix, which specifies the sensors participating in each join. When joinable sensors exhibit high key-space diversity or high data rates, Nova further balances the load by partitioning the join as defined in Phase III. Joins over non-partitionable key spaces are outside Nova's scope, as Nova fundamentally relies on partitioning and replication for scalability.

The key idea for complex workloads is to generalize the placement objective in Phase II (Section 3.3) using a spring-force model, as proposed by Rizou et al. [57], where springs represent communication costs between operators, yielding a convex optimization problem that remains efficiently solvable. Simple, stateless filters can be colocated with upstream operators due to their negligible overhead. For aggregations, operators can be added to the spring-force model in Phase II to determine their virtual placement within the same cost-space abstraction as NEMO [11]. Phase III then parallelizes and places decomposable aggregations using NEMO.

For multi-way joins, the query can be decomposed into a sequence of two-way joins, each modeled as a node in the spring-force graph. Join-order optimization is orthogonal to Nova and can be addressed using existing work [84]. However, since multi-way joins can produce exponentially growing outputs, Nova's bandwidth-aware partitioning helps manage intermediate data volumes by splitting streams proportionally to input rates and node capacities.

4 Evaluation

We evaluate Nova in two parts. First, we extend the simulations of Chatziliadis et al. [11], using cost spaces derived from both real and synthetic topologies (Sections 4.2–4.6). Second, we integrate Nova into NebulaStream's optimizer [82, 83] and evaluate an end-to-end deployment on a physical cluster (Section 4.7).

4.1 Experimental Setup

Hardware. Simulations are executed using sequential Python scripts on a workstation with an AMD Ryzen 7 5700X CPU and 32 GB RAM. Following the experimental setup of NEMO [11], we conduct our end-to-end experiments on the same testbed, consisting of a cluster of 14 Raspberry Pi 4B devices (Quad-Core Cortex-A72, 1.5 GHz, 4 GB RAM). All nodes run Ubuntu 22.04 and are interconnected via Gigabit Ethernet through a switch.

Simulation. We use the Vivaldi algorithm [19] to generate network coordinate systems (NCSs) from latency measurements of real-world and artificial topologies. Specifically, we use latency measurements from the following topologies (cf. Figure 5): 1) *FIT IoT Lab* [2], an IoT testbed in France, where we evaluate round-trip times (RTTs) from 433 geographically distributed nodes, including microcontrollers, ARM devices, and four gateway servers. 2) *RIPE Atlas* [66], an Internet measurement platform, where we analyze RTTs from 723 globally distributed anchors that serve as fixed latency measurement points. 3) *PlanetLab* [17], consisting of RTTs from 335 nodes hosted by universities and research institutions in Europe and North America. 4) *King* [31], containing latency measurements from 1,740 Internet DNS servers.

To select the number of neighbors m for Vivaldi, we evaluate network coordinate accuracy using NCSIM [15] and measure the mean absolute error (MAE). We observe that MAE converges quickly as m increases, with negligible accuracy improvements beyond a small neighborhood size. Our experiments thus lead to the same neighborhood size as prior work, i.e., $m = 20$ for RIPE Atlas and FIT IoT Lab, and $m = 32$ for PlanetLab and King.

To evaluate Nova's scalability on large topologies beyond the scale of existing systems, we generate synthetic NCSs with varying latency distributions and sizes from 10^3 to 10^6 nodes. Nodes are positioned within $[0, 100]$ (x-axis) and $[-50, 50]$ (y-axis), using Gaussian clusters to emulate heterogeneous, geo-distributed networks. These synthetic topologies enable controlled scalability testing beyond real-world measurements (cf. Section 4.6).

End-to-end Deployment. To evaluate Nova's end-to-end performance, we integrate it into NebulaStream [82] for operator placement and deploy NebulaStream on the testbed. In this setup, one Raspberry Pi acts as the coordinator/sink, eight serve as sources, and the remaining nodes function as workers. To emulate real-world conditions of network and compute heterogeneity, we follow the methodology of NEMO [11]: injecting artificial latency between nodes based on RIPE Atlas data using Linux's `tc` tool, and applying artificial load to the source nodes with `stress` (full CPU utilization and 80 % memory usage).

Workloads. In our simulations, we evaluate Nova using geo-distributed join queries involving two conceptual streams, each originating from multiple joinable data sources. To assess performance under varying loads, we adjust the computational capacities, number of sources, and data rates. We control the heterogeneity of the topology by varying node capacities while keeping the total capacity approximately constant. Specifically, we transition from a near-uniform capacity distribution (low heterogeneity) to increasingly skewed distributions (high heterogeneity), ranging from uniform (capacities between 1 and 200, median around 35) to exponential (capacities between 1 and 1000, median around

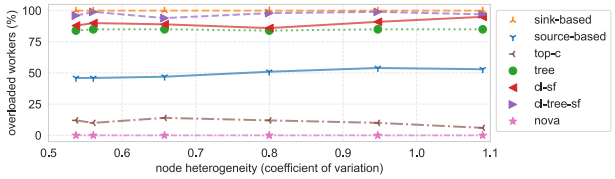


Figure 6: Percentage of overloaded nodes versus node heterogeneity (Coefficient of Variation of node capacities). Higher CV indicates greater resource imbalance.

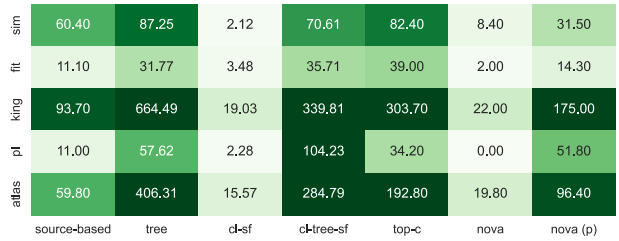


Figure 7: Latency deltas (90th percentile) relative to sink-based direct transmission across different approaches.

28). These variations produce increasing coefficients of variation (CV), which we use to quantify node heterogeneity. Minor fluctuations in CV values may occur due to topology size and distribution sampling. In addition to node-capacity heterogeneity, we vary the per-source input rates to create load imbalances similar to those caused by skewed key distributions. We randomly designate 60% of the nodes as sources and the remaining 40% as workers, mirroring the hardware distribution in the FIT IoT Lab [2]. As the topology grows, the increasing number of sources proportionally raises the overall load. The sink is randomly selected to avoid bias. Since Nova is designed for 2-way joins, each source is randomly assigned to one of two logical streams and joined with exactly one other source, such that the corresponding join matrix contains exactly one entry per row. Finally, we uniformly distribute source data rates between 1 and 200 to represent varying resource requirements.

We evaluate Nova in our end-to-end deployments using an environmental monitoring workload inspired by the DEBS 2021 Grand Challenge [68]. We join pressure and humidity readings from four regions using tumbling windows ranging from 1 ms to 1 s, resulting in four parallel two-way joins across eight sensors. Each sensor generates data at a fixed rate of 1 kHz (1000 tuples per second), such that source windows contain between one tuple for 1 ms windows and 1000 tuples for 1 s windows.

We intentionally include very small windows (e.g., 1 ms) to induce high operational pressure in our small-scale testbed, rather than to reflect typical application workloads. Such window sizes substantially increase the frequency of window creation, watermark propagation, and state and buffer management, leading to significant system overhead. This synthetic load enables us to approximate the behavior of much larger deployments, where comparable overhead arises from many concurrent sources, without requiring a proportional increase in physical nodes. At the other extreme, larger windows (e.g., 1 s) shift the bottleneck toward per-window processing and buffering costs.

Baselines. We compare Nova against the following baselines: 1) Sink-based: The default approach of NebulaStream [82], where joins are computed at the sink node. 2) Source-based: A locality-aware heuristic adapted for streaming joins, resolving the join matrix by placing each join at the source with the highest data rate [67]. 3) Top-c: A resource-aware heuristic representing cloud-based approaches by placing and parallelizing the join on the node with the highest computational capacity. 4) Tree: A WSN-based approach where the topology forms a minimum spanning tree (MST), joining data at intersections [49]. 5) Cl-SF: An adapted WSN approach that clusters the topology using LEACH-SF [64], computing joins at intersecting cluster heads or the sink if none exist. 6) Cl-Tree-SF: A hybrid of Tree and Cl-SF, clustering the topology, forming an MST among cluster heads, and computing joins at intersections.

4.2 Mitigation of Over-Utilization

In this section, we evaluate Nova's ability to avoid node over-utilization under increasing heterogeneity. Figure 6 shows the percentage of overloaded nodes compared to six baselines, under varying node heterogeneity, quantified by the CV of node capacities (cf. Section 4.1). Higher CV values indicate greater heterogeneity and resource-imbalance. We report overloaded nodes, defined as those exceeding their capacity, as a percentage of total workers. We focus on results from a 1000-node synthetic topology, as trends remained consistent across different scales and topologies.

As shown in Figure 6, Nova consistently outperforms all baselines, maintaining zero overloaded nodes across all capacity distributions. In contrast, the sink-based approach performs the worst, invariably overloading 100% of its workers since the sink node alone handles all computation. Among the remaining baselines, the cluster- and tree-based approaches used by WSNs show the highest overload, with the hybrid variant (Cl-Tree-SF) ranging from 94% to 99% overloaded nodes, followed by Cl-SF at 86% to 95%, and the pure tree-based approach at roughly 85% across all distributions. Because these methods are resource-agnostic, they concentrate the join workload on a limited set of nodes chosen primarily for their topological position, rather than their capacity.

The source-based approach shows similar behavior but nearly halves the overload compared to WSN-based methods by distributing computation among more source nodes. Nonetheless, it remains resource-agnostic, resulting in 46% overload in the homogeneous scenario and rising to 54% under higher heterogeneity. The best-performing baseline is top-c (6%-14% overloaded nodes), as it assigns sub-joins to the node with the highest available capacity, but lacks distributed parallelization, so large computations can still overwhelm individual nodes.

In summary, Nova's resource awareness, combined with parallelization and partitioning, prevents worker over-utilization across capacity distributions, significantly outperforming all baselines.

4.3 Placement Quality

In this section, we evaluate the theoretical latencies of each approach by comparing their 90th percentile (90P) latencies (in milliseconds) against the theoretical lower bound defined by the sink-based solution (Figure 7). For clarity, we exclude processing delays and estimation errors in this analysis. The effects of estimation errors are discussed in Section 4.4, while end-to-end latencies in real-world deployments are evaluated in Section 4.7.

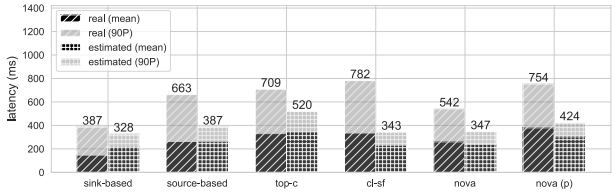


Figure 8: Mean and 90th percentile latencies using NCS estimates (left) vs. real measurements (right) on RIPE Atlas.

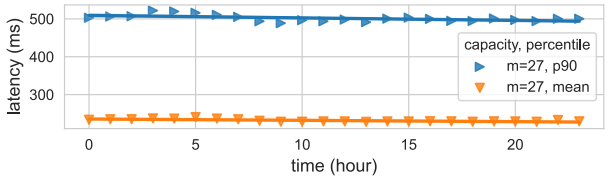


Figure 9: Mean and 90th percentile latencies of Nova for RIPE Atlas over 24 hours.

Overall, Nova and Cl-SF achieve the lowest latencies, consistently approaching the theoretical lower bound across all topologies. Cl-SF shows minimal deviations from the lower bound, with deltas of 2–3.5 ms for PlanetLab, FIT, and 1K-node simulations, 15.57 ms for RIPE Atlas, and 19.03 ms for King, thanks to its clustering design that prioritizes minimal latency. Similarly, Nova achieves near-optimal latencies (deltas of 0 ms on PlanetLab, 2 ms on FIT, 8.4 ms on the 1K simulation, 19.8 ms on RIPE Atlas, 22 ms on King) when workloads can be accommodated on a small number of nodes. Nova(p) denotes the placement generated for the most heterogeneous topology, which requires the highest degree of replication to achieve load balance. In this challenging setting, Nova(p)'s latencies increase up to 14.3 ms (FIT), 31.5 ms (1K simulation), 51.8 ms (PlanetLab), 96.4 ms (RIPE Atlas), and 175 ms (King).

Compared to Nova and Cl-SF, source-based and top-c approaches exhibit moderately higher latencies, often exceeding 10 ms for PlanetLab and FIT, and reaching up to 190+ ms for RIPE Atlas and King. However, they still outperform tree-based methods (Tree, Cl-Tree-SF), which show significantly larger latencies due to multi-hop routing. For instance, tree-based methods exhibit deltas ranging from 35 ms (FIT) to over 650 ms (King), which are more than twice as high as Nova's fully parallelized placements (Nova(p)).

In summary, Nova's resource-aware and parallelized design allows it to operate close to the theoretical lower bound when feasible. It also scales gracefully under heavier workloads by leveraging operator replication and stream partitioning to prevent overload, while maintaining low latency.

4.4 Impact of Estimation Errors

This section evaluates the impact of NCS estimation errors by comparing estimated and real mean and 90th percentile latencies using measurements from a 418 node RIPE Atlas subset. Figure 8 summarizes the results, with Tree and Cl-Tree-SF omitted because their latencies are orders of magnitude higher than all other approaches.

For mean latency, Nova exhibits low estimation error with 237 ms estimated versus 259 ms measured. Source-based and top-c also show small discrepancies at 261 ms versus 262 ms and 345 ms versus 330 ms, while sink-based shows a larger deviation at 220 ms versus 146 ms. In contrast, tree-based approaches suffer extreme underestimation, with Tree increasing from 512 ms to 11.7 s and Cl-Tree-SF from 450 ms to 2 s.

At the 90th percentile, tail effects increase estimation error across all approaches, with Nova rising from 347 ms to 542 ms and Nova(p) from 424 ms to 754 ms. Source-based increases from 387 ms to 663 ms and top-c from 520 ms to 709 ms, while tree-based approaches again exhibit extreme underestimation, jumping from 733 ms to 19 s. Overall, WSN-based approaches that are not cost-space optimized are highly sensitive to TIVs, whereas Nova maintains accurate mean estimates and competitive tail latency, achieving a real 90th percentile latency of 542 ms that is 35 times lower than tree-based approaches and comparable to top-c, demonstrating that cost-space optimization yields deployments robust to NCS estimation errors.

4.5 Resilience to Latency Variations

To evaluate Nova's resilience to latency variations, we measured mean and 90P latencies over a 24-hour period on a fixed 418-node subset of RIPE Atlas. We show only the results for the highest heterogeneity and level of parallelization with a median of 27 ms, as other distributions showed similar patterns. During this period, the number of changed latency entries between successive measurements over a 10 ms threshold ranged from 7k to 14k, with a median change magnitude of 24 ms.

Figure 9 shows that the 90P latencies range from approximately 489 ms to 522 ms, while the mean latencies stay between 228 ms and 241 ms. Despite day-to-night transitions and varying network conditions, the observed standard deviations remain within tens of milliseconds for both metrics, indicating that Nova's placements are largely unaffected by transient latency spikes or routing changes. These results highlight Nova's robustness in long-running stream processing scenarios, reducing the need for frequent re-optimizations even under dynamic network conditions.

4.6 Placement and Re-optimization Scalability

In this section, we evaluate Nova's scalability for full optimization and re-optimization. For each topology size, we trigger five re-optimization events per run: adding a source, removing a source, removing a worker, updating a node's coordinates, and changing the data rate of a source. Each event is applied to a single randomly selected node. To ensure comparability, the number of updates remains constant across all topology sizes, while query complexity grows proportionally with topology size (i.e., the number of operators and streams increases). Figure 10 shows optimization time versus topology size, with the x-axis showing the number of nodes, and y-axis the cumulative runtime in seconds (log scale).

Nova scales efficiently, placing operators in topologies of up to 1 million nodes in approximately 135 seconds. Runtime grows consistently with topology size, increasing from 0.015 seconds at 100 nodes to around 135 seconds at 1 million nodes. While the underlying complexity remains near-linear, runtimes span several orders of magnitude, motivating the use of a log scale for clearer visualization. In contrast, Tree, Cl-SF, and Cl-Tree-SF baselines exceed the 10-minute timeout beyond 20,000 nodes, making them impractical for large-scale deployments. While sink-based, source-based, and top-c methods remain consistently

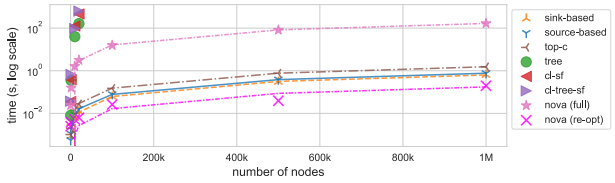


Figure 10: Optimization and re-optimization times (log scale) for workloads where both topology size and query complexity (number of operators and streams) grow proportionally.

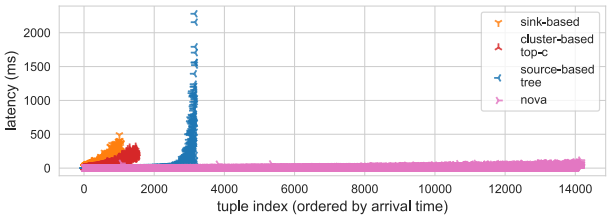


Figure 11: Latency trends over processed tuples for the DEBS 2021 end-to-end deployment (non-stressed)

fast (under 2 seconds even for 1 million nodes), they ignore node capacity constraints, which leads to overloading. Nova, in contrast, ensures resource-aware placement at competitive runtimes.

Nova's re-optimization times remain below one second even at 1 million nodes. Adding a new source, which is the most expensive event, takes 0.2 s, while all other re-optimization events complete in well under 0.1 s. This demonstrates Nova's efficiency in adapting to dynamic changes without full recomputation. In contrast, the baselines require full placement recomputation for the same events, as they lack incremental update mechanisms.

4.7 End-to-end Performance

This section evaluates Nova's performance on the DEBS 2021 benchmark workload using a 14-node Raspberry Pi cluster as proposed by Chatziliadis et al. [11]. We measure end-to-end latency (network and computational delays) and throughput (total tuples processed) over a two-minute runtime, comparing Nova against our baselines. To ensure comparability, node churn is disabled. In this topology, the cluster-based approaches cl-sf and cl-tree-sf, as well as top-c, produce identical placements and are grouped together. Similarly, the source-based and the tree approach yield the same placement and are also grouped together.

Throughput. Figure 11 illustrates Nova's throughput advantage by depicting the number of processed tuples within the two-minute query runtime (x-axis) versus their latency (y-axis) for the non-stressed workload. The sink-based approach achieves the lowest throughput, processing 1,057 tuples, as it performs all computations centrally, overloading the sink node. The cluster-based and top-c approaches achieve slightly higher throughput (1,503 tuples) by computing joins on a single cluster head, which has more resources than the sink but remains a bottleneck.

The source-based/tree baselines process 3,176 tuples, doubling throughput by distributing joins across multiple source nodes. However, these nodes lack computational capacity, as they share resources with data ingestion, leading to overloads. In contrast, Nova achieves the highest throughput, processing 14,159 tuples, 4.5× more than the source-based/tree and 13.4× more than the sink-based approach. This order-of-magnitude improvement stems from Nova's ability to parallelize join operations, distributing them across four worker nodes to prevent overload.

Latency. Nova's parallelized execution and resource-aware placement are also reflected in the latency distributions, shown in Figure 12, which presents latency under both normal and stress conditions. Nova achieves a mean latency of 8 ms, significantly outperforming the sink-based (115 ms, 14.4× higher), cluster-based/top-c (80 ms, 10× higher), and source-based/tree (37 ms, 4.6× higher) approaches. Nova's 99.99th percentile latency remains tightly bounded at 91 ms, significantly lower than the sink-based (481 ms, 5.3× higher) and cluster-based/top-c (310 ms, 3.4× higher) baselines.

Under stress, Nova remains robust, with its mean latency increasing to 13 ms and a 99.99th percentile of 113 ms. In contrast, the cluster-based/top-c approach spikes to 4,433 ms at the 99.99th percentile (39× higher than Nova), while the source-based/tree approach reaches a mean latency of 54 ms (4.2× higher than Nova).

In summary, Nova achieves orders-of-magnitude improvements in both throughput and latency by optimizing the parallelization and placement of join operators to prevent node overload and minimize network delay. Its throughput surpasses the best baseline by 4.5×, while its latency is 4.6–14.4× lower under normal conditions and 4.2–39× lower under stress. These results validate Nova's suitability for resource-constrained, geo-distributed environments.

5 Related Work

We group existing approaches into three main categories: WSN-based methods, cloud-based methods, and wide-area (WA)-based methods (including NEMO [11]). Although each category addresses crucial aspects of distributed stream processing, such as in-network data aggregation, elasticity, and operator scheduling, most fail to unify placement, partitioning, and replication in a way that is both resource-aware and capable of re-optimizing at large scales. In contrast, Nova addresses capacity constraints by parallelizing joins in an integrated fashion while being scalable and adaptable.

WSN-based Methods. Early work in WSNs emphasizes reducing energy use and bandwidth through in-network aggregation. Cluster-based protocols (e.g., LEACH [34], HEED [79], CLUDDA [10]) select cluster heads that summarize data locally. Tree-based methods like EADAT [22] or PEDAP [69] construct an MST to minimize transmissions, while chain-based solutions such as PEGASIS [45] organize nodes in a linear chain for incremental aggregation. These protocols effectively conserve energy but do not account for latency and node overloading.

Beyond aggregation, various WSN-oriented approaches tackle joins. Multi-hop in-network joins [48, 49] optimize communication by decomposing join operations into partial computations distributed across network paths. Broadcast [77, 80] and hash-based [32, 52] joins localize semijoin computations near data sources. Filtering methods such as predicate embedding [1] and fragmented semijoins [39] reduce network traffic by discarding irrelevant readings locally and eliminating low-frequency data. While these techniques parallelize workloads and reduce communication costs through partial in-network computation, they

Figure 12: Comparison of the mean and 90-99.99 percentile latencies for the DEBS 2021 end-to-end deployment.

are primarily designed for small-scale topologies and do not effectively handle node capacity constraints, latency requirements, or overload in larger heterogeneous environments.

Cloud-based Methods. Many cloud-based SPEs elastically adjust operator parallelism to handle workload fluctuations [25, 26, 37, 38, 75]. Streambed [58] focuses on capacity planning for distributed dataflow, while Das et al. [20] tune batch sizes to balance throughput and latency. Although effective in cloud settings, these approaches generally assume abundant resources or easy horizontal scaling and thus overlook tight capacity constraints, complex routing paths, and node heterogeneity.

Scheduling frameworks [51, 76] and dynamic partitioning techniques [28, 40, 50] aim to meet latency targets and mitigate skew by distributing tasks and splitting streams. However, they typically treat placement separately from partitioning or replication, which can cause node overload in geo-distributed or resource-constrained environments.

Existing placement algorithms are often too costly or inflexible for online use [7, 8] and may require manual tuning [59]. For example, Heinze et al. [33] propose a bin-packing heuristic that minimizes latency violations but does not explicitly model computational resource constraints such as memory and processing throughput. CAPSys [74] addresses resource contention but lacks support for complex geo-distributed topologies and does not jointly handle partitioning and replication like Nova.

WA-based Methods. SBON [55] and its multi-operator placement extension [57] apply cost-space embedding for iterative operator placement, yielding efficient solutions for single- and multi-operator scenarios, but prioritize placement optimization without incorporating constraints related to node capacities, replication strategies, or stream partitioning. NEMO [11] addresses node capacities and replication for decomposable aggregation functions. However, NEMO cannot process joins because (1) NEMO's tree-based aggregation decomposition reduces data volume at each tree level, but joins amplify data, making this decomposition inapplicable; (2) its optimization couples placement decisions across levels of the aggregation tree, whereas join replicas require independent per-replica optimization; and (3) its cost model does not account for bandwidth constraints and heterogeneous input rates, both critical for joins where operator replication increases network traffic.

Systems like WASP [36], SWAN [65], DART [46], Droplet [23] use ILP, dynamic programming, or heuristics to minimize end-to-end latency, job completion times, or bandwidth usage across data centers. Some solutions (e.g., WASP) focus on inter-data-center scheduling only, while others (e.g., DART) propose decentralized overlays for computation orchestration. While effective at reducing network overhead, most lack support for load reduction through operator replication or re-optimizations when the topology changes.

6 Conclusion

Nova provides a scalable and resource-aware approach for optimizing the placement and parallelization of join operators in geo-distributed environments. By framing placement as a convex geometric median problem in a Euclidean cost space and implementing bandwidth-aware stream partitioning, Nova efficiently approximates the NP-hard Operator Placement and Parallelization problem in linear time, while also enabling adaptivity in dynamic and heterogeneous environments.

Our results show that current latency-optimal strategies are insufficient, often overloading resource-constrained nodes when placing joins near data sources. Nova addresses this challenge by balancing load through joint optimization of placement, replication, and partitioning, thereby minimizing latency without overloading the network. Future work will empirically evaluate multi-way join performance, integrate predictive workload models, and explore different cost metrics beyond latency.

Acknowledgments

This work was funded by the German Federal Ministry for Education and Research as BIFOLD - Berlin Institute for the Foundations of Learning and Data (ref. BIFOLD24B).

Artifacts

The code, datasets, and artifacts for Nova, together with instructions, are available at https://github.com/xchatzil/NOVA.

References

[1] Daniel J. Abadi, Samuel Madden, and Wolfgang Lindner. 2005. REED: Robust, Efficient Filtering and Event Detection in Sensor Networks. In *Proceedings of the 31st International Conference on Very Large Data Bases*. 769–780.

[2] Cédric Adjih, Emmanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noël, Roger Pissard-Gibollet, Frederic Saint-Marcel, Guillaume Schreiner, Julien Vandaele, and Thomas Watteyne. 2015. FIT IoT-LAB: A large scale open experimental IoT testbed. In *2nd IEEE World Forum on Internet of Things*. 459–464.

[3] Arif Ahmed, HamidReza Arkian, Davaadorj Battulga, et al. 2019. Fog Computing Applications: Taxonomy and Requirements. *arXiv preprint arXiv:1907.11621* (2019).

[4] Erik Bernhardsson. 2018. Annoy: Approximate nearest neighbors in c++/python. *Python package version* 1, 0 (2018).

[5] David Burrell, Xenofon Chatziliadis, Eleni Tzirita Zacharatou, Steffen Zeuch, and Volker Markl. 2023. Workload Prediction for IoT Data Management Systems. In *Datenbanksysteme für Business, Technologie und Web, BTW*.

[6] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *The Bulletin of the Technical Committee on Data Engineering* 38, 4 (2015).

[7] Valeria Cardellini, Vincenzo Grassi, Francesco Lo Presti, and Matteo Nardelli. 2016. Optimal operator placement for distributed stream processing applications. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, DEBS*. 69–80.

[8] Valeria Cardellini, Francesco Lo Presti, Matteo Nardelli, and Gabriele Russo Russo. 2018. Optimal operator deployment and replication for elastic distributed data stream processing. *Concurr. Comput. Pract. Exp.* 30, 9 (2018).

[9] Dirk G Cattrysse and Luk N Van Wassenhove. 1992. A survey of algorithms for the generalized assignment problem. *European journal of operational research* 60, 3 (1992), 260–272.

[10] Supriyo Chatterjea and Paul JM Havinga. 2003. A dynamic data aggregation scheme for wireless sensor networks. In *14th Workshop on Circuits, Systems and Signal Processing, ProRISC*.

[11] Xenofon Chatziliadis, Eleni Tzirita Zacharatou, Alphan Eracar, Steffen Zeuch, and Volker Markl. 2024. Efficient Placement of Decomposable Aggregation Functions for Stream Processing over Large Geo-Distributed Topologies. *Proceedings of the VLDB Endowment* 17, 6 (2024), 1501–1514.

[12] Xenofon Chatziliadis, Eleni Tzirita Zacharatou, Steffen Zeuch, and Volker Markl. 2021. Monitoring of Stream Processing Engines Beyond the Cloud: An Overview. *Open J. Internet Things* 7, 1 (2021), 71–82.

[13] Ankit Chaudhary, Kaustubh Beedkar, Jeyhun Karimov, Felix Lang, Steffen Zeuch, and Volker Markl. 2025. Incremental Stream Query Placement in Massively Distributed and Volatile Infrastructures. *41st IEEE International Conference on Data Engineering (ICDE 2025)* (2025).

[14] Ankit Chaudhary, Felix Lang, Danila Ferents, Nils L Schubert, Varun Pandey, Jeyhun Karimov, Ste en Zeuch, Kaustubh Beedkar, and Volker Markl. 2026. Incremental Stream ery Deployment under Continuous Infrastructure Changes in the Cloud-Edge Continuum. *Proceedings of the VLDB Endowment* 19 (2026).

[15] Yang Chen, Xiao Wang, Cong Shi, Eng Keong Lua, Xiaoming Fu, Beixing Deng, and Xing Li. 2011. Phoenix: A Weight-Based Network Coordinate System Using Matrix Factorization. *IEEE Trans. Netw. Serv. Manag.* 8, 4 (2011), 334–347.

[16] Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar, Thomas Graves, Mark Holderbaugh, Zhuo Liu, Kyle Nusbaum, Kishorkumar Patil, Boyang Jerry Peng, et al. 2016. Benchmarking streaming computation engines: Storm, flink and spark streaming. In *2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*. IEEE, 1789–1792.

[17] Brent N. Chun, David E. Culler, Timothy Roscoe, Andy C. Bavier, Larry L. Peterson, Mike Wawrzoniak, and Mic Bowman. 2003. PlanetLab: An overlay testbed for broad-coverage services. *Comput. Commun. Rev.* 33, 3 (2003), 3–12.

[18] Thiago Pereira Da Silva, Thais Batista, Frederico Lopes, Aluizio Rocha Neto, Flávia C Delicato, Paulo F Pires, and Atslands R Da Rocha. 2022. Fog computing platforms for smart city applications: A survey. *ACM Transactions on Internet Technology* 22, 4 (2022), 1–32.

[19] Frank Dabek, Russ Cox, M. Frans Kaashoek, and Robert Tappan Morris. 2004. Vivaldi: A decentralized network coordinate system. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM*. 15–26.

[20] Tathagata Das, Yuan Zhong, Ion Stoica, and Scott Shenker. 2014. Adaptive stream processing using dynamic batch sizing. In *Proceedings of the ACM Symposium on Cloud Computing*. 1–13.

[21] Abdulkadir Dauda, Olivier Flauzac, and Florent Nolot. 2024. A survey on IoT application Architectures. *Sensors* 24, 16 (2024), 5320.

[22] Min Ding, Xiuzhen Cheng, and Guoliang Xue. 2003. Aggregation tree construction in sensor networks. In *58th Vehicular Technology Conference, VTC*. 2168–2172.

[23] Tarek Elgamal, Atul Sandur, Phuong Nguyen, Klara Nahrstedt, and Gul Agha. 2018. Droplet: Distributed operator placement for iot applications spanning edge and cloud resources. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 1–8.

[24] Sándor P Fekete, Joseph SB Mitchell, and Karin Beurer. 2005. On the continuous Fermat-Weber problem. *Operations Research* 53, 1 (2005), 61–76.

[25] Avrilia Floratou, Ashvin Agrawal, Bill Graham, Sriram Rao, and Karthik Ramasamy. 2017. Dhalion: Self-Regulating Stream Processing in Heron. *Proc. VLDB Endow.* 10, 12 (2017), 1825–1836.

[26] Tom ZJ Fu, Jianbing Ding, Richard TB Ma, Marianne Winslett, Yin Yang, and Zhenjie Zhang. 2017. DRS: Auto-scaling for real-time stream analytics. *IEEE/ACM Transactions on networking* 25, 6 (2017), 3338–3352.

[27] Longxiang Gao, Tom H Luan, Bo Liu, Wanlei Zhou, and Shui Yu. 2017. Fog computing and its applications in 5G. *5G Mobile Communications* (2017), 571–593.

[28] Bugra Gedik. 2014. Partitioning functions for stateful data parallelism in stream processing. *VLDB J.* 23, 4 (2014), 517–539.

[29] Panagiotis Gkonis, Anastasios Giannopoulos, Panagiotis Trakadas, Xavi Masip-Bruin, and Francesco D'Andria. 2023. A survey on IoT-edge-cloud continuum systems: Status, challenges, use cases, and open issues. *Future Internet* 15, 12 (2023), 383.

[30] Vincenzo Gulisano, Alessandro V Papadopoulos, Yiannis Nikolakopoulos, Marina Papatriantafilou, and Philippas Tsigas. 2017. Performance modeling of stream joins. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*. 191–202.

[31] P. Krishna Gummadi, Stefan Saroiu, and Steven D. Gribble. 2002. King: Estimating latency between arbitrary internet end hosts. *Comput. Commun. Rev.* 32, 3 (2002), 11.

[32] Himanshu Gupta and Vishal Chowdhary. 2007. Communication-efficient implementation of join in sensor networks. *Ad Hoc Networks* 5, 6 (2007), 929–942.

[33] Thomas Heinze, Yuanzhen Ji, Lars Roediger, Valerio Pappalardo, Andreas Meister, Zbigniew Jerzak, and Christof Fetzer. 2015. FUGU: Elastic Data Stream Processing with Latency Constraints. *IEEE Data Eng. Bull.* 38, 4 (2015), 73–81.

[34] Wendi Rabiner Heinzelman, Anantha P. Chandrakasan, and Hari Balakrishnan. 2000. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In *33rd Annual Hawaii International Conference on System Sciences, HICSS*. 10–20.

[35] Michael C Hout, Megan H Papesh, and Stephen D Goldinger. 2013. Multidimensional scaling. *Wiley Interdisciplinary Reviews: Cognitive Science* 4, 1 (2013), 93–103.

[36] Albert Jonathan, Abhishek Chandra, and Jon Weissman. 2020. WASP: Wide-area adaptive stream processing. In *Proceedings of the 21st international middleware conference*. 221–235.

[37] Vasiliki Kalavri, John Liagouris, Moritz Hoffmann, Desislava Dimitrova, Matthew Forshaw, and Timothy Roscoe. 2018. Three steps is all you need: fast, accurate, automatic scaling decisions for distributed streaming dataflows. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 783–798.

[38] Faria Kalim, Le Xu, Sharanya Bathey, Richa Meherwal, and Indranil Gupta. 2018. Henge: Intent-driven multi-tenant stream processing. In *Proceedings of the ACM Symposium on Cloud Computing*. 249–262.

[39] Hyunchul Kang. 2015. In-Network Processing of an Iceberg Join Query in Wireless Sensor Networks Based on 2-Way Fragment Semijoins. *Sensors* 15, 3 (2015), 6105–6132.

[40] Nikos R Katsipoulakis, Alexandros Labrinidis, and Panos K Chrysanthis. 2017. A holistic view of stream partitioning costs. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1286–1297.

[41] Anastasiia Kozar, Bonaventura Del Monte, Steffen Zeuch, and Volker Markl. 2024. Fault Tolerance Placement in the Internet of Things. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–29.

[42] Billy Pik Lik Lau, Sumudu Hasala Marakkalage, Yuren Zhou, Naveed Ul Hassan, Chau Yuen, Meng Zhang, and U-Xuan Tan. 2019. A survey of data fusion in smart city applications. *Information Fusion* 52 (2019), 357–374. doi:10.1016/j.inffus.2019.05.004

[43] Jonathan Ledlie, Paul Gardner, and Margo I Seltzer. 2007. Network Coordinates in the Wild.. In *NSDI*, Vol. 7. 299–311.

[44] Aljoscha P. Lepping, Hoang Mi Pham, Laura Mons, Balint Rueb, Philipp M. Grulich, Ankit Chaudhary, Steffen Zeuch, and Volker Markl. 2023. Showcasing Data Management Challenges for Future IoT Applications with NebulaStream. *Proc. VLDB Endow.* 16, 12 (2023), 3930–3933.

[45] Stephanie Lindsey, Cauligi S. Raghavendra, and Krishna M. Sivalingam. 2002. Data Gathering Algorithms in Sensor Networks Using Energy Metrics. *IEEE Trans. Parallel Distributed Syst.* 13, 9 (2002), 924–935.

[46] Pinchao Liu, Dilma Da Silva, and Liting Hu. 2021. DART: A scalable and adaptive edge stream processing engine. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 239–252.

[47] Quazi Mamun. 2012. A Qualitative Comparison of Different Logical Topologies for Wireless Sensor Networks. *Sensors* 12, 11 (2012), 14887–14913.

[48] Svilen R Mihaylov, Marie Jacob, Zachary G Ives, and Sudipto Guha. 2008. A substrate for in-network sensor data integration. In *Proceedings of the 5th workshop on Data management for sensor networks*. 35–41.

[49] Svilen R Mihaylov, Marie Jacob, Zachary G Ives, and Sudipto Guha. 2010. Dynamic join optimization in multi-hop wireless sensor networks. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 1279–1290.

[50] Muhammad Anis Uddin Nasir, Gianmarco De Francisci Morales, Nicolas Kourtellis, and Marco Serafini. 2016. When two choices are not enough: Balancing at scale in distributed stream processing. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 589–600.

[51] Kay Ousterhout, Patrick Wendell, Matei Zaharia, and Ion Stoica. 2013. Sparrow: distributed, low latency scheduling. In *Proceedings of the twenty-fourth ACM symposium on operating systems principles*. 69–84.

[52] Aditi Pandit and Himanshu Gupta. 2006. Communication-efficient implementation of range-joins in sensor networks. In *International Conference on Database Systems for Advanced Applications*. Springer, 859–869.

[53] Rahul Krishnan Pathinarupothi, Dipu T. Sathyapalan, Merlin Moni, K A Unnikrishna Menon, and Maneesha Vinodini Ramesh. 2021. REWOC: Remote Early Warning of Out-of-ICU Crashes in COVID Care Areas using IoT Device. In *2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. 2010–2013. doi:10.1109/BIBM52615.2021.9669464

[54] Elena Beatriz Ouro Paz, Eleni Tzirita Zacharatou, and Volker Markl. 2021. Towards Resilient Data Management for the Internet of Moving Things. In *Datenbanksysteme für Business, Technologie und Web, BTW*. 279–301.

[55] Peter R. Pietzuch, Jonathan Ledlie, Jeffrey Shneidman, Mema Roussopoulos, Matt Welsh, and Margo I. Seltzer. 2006. Network-Aware Operator Placement for Stream-Processing Systems. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE*. 49.

[56] Nima Rezaei and Mohammad Nasir Uddin. 2021. An analytical review on state-of-the-art microgrid protective relaying and coordination techniques. *IEEE Transactions on Industry Applications* 57, 3 (2021), 2258–2273.

[57] Stamatia Rizou, Frank Dürr, and Kurt Rothermel. 2010. Solving the Multi-Operator Placement Problem in Large-Scale Operator Networks. In *Proceedings of the 19th International Conference on Computer Communications and Networks, ICCCN*. 1–6.

[58] Guillaume Rosinosky, Donatien Schmitz, and Etienne Rivière. 2024. StreamBed: capacity planning for stream processing. In *Proceedings of the 18th ACM International Conference on Distributed and Event-based Systems*. 90–102.

[59] Antonio J Rubio-Montero, Eduardo Huedo, and Rafael Mayo-García. 2015. User-guided provisioning in federated clouds for distributed calculations. In *Adaptive Resource Management and Scheduling for Cloud Computing: Second International Workshop, ARMS-CC 2015, Held in Conjunction with ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 20, 2015, Revised Selected Papers 2*. Springer, 60–77.

[60] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

[61] Radhya Sahal, John G Breslin, and Muhammad Intizar Ali. 2020. Big data and stream processing platforms for Industry 4.0 requirements mapping for a predictive maintenance use case. *Journal of manufacturing systems* 54 (2020), 138–151.

[62] Sensor.Community. [n. d.]. Sensor.Community: Open Environmental Data from a Global Sensor Network. https://sensor.community/. Accessed: 2026-01-01.
[63] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE internet of things journal* 3, 5 (2016), 637–646.
[64] Mohammad Shokouhifar and Ali Jalali. 2017. Optimized sugeno fuzzy clustering algorithm for wireless sensor networks. *Eng. Appl. Artif. Intell.* 60 (2017), 16–25.
[65] Won Wook Song, Myeongjae Jeon, and Byung-Gon Chun. 2022. SWAN: WAN-aware stream processing on geographically-distributed clusters. In *Proceedings of the 13th ACM SIGOPS Asia-Pacific Workshop on Systems*. 78–84.
[66] RIPE NCC Staff. 2015. Ripe Atlas: A global internet measurement network. *Internet Protocol Journal* 18, 3 (2015), 2–26.
[67] Bruhathi Sundarmurthy, Paraschos Koutris, and Jeffrey Naughton. 2021. Locality-Aware Distribution Schemes. In *24th International Conference on Database Theory (ICDT 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.
[68] Jawad Tahir, Christoph Doblander, Ruben Mayer, Sebastian Frischbier, and Hans-Arno Jacobsen. 2021. The DEBS 2021 grand challenge: analyzing environmental impact of worldwide lockdowns. In *Proceedings of the 15th ACM International Conference on Distributed and Event-based Systems*. 136–141.
[69] Hüseyin Özgür Tan and Ibrahim Korpeoglu. 2003. Power efficient data gathering and aggregation in wireless sensor networks. *SIGMOD Rec.* 32, 4 (2003), 66–71.
[70] Tristan Joel Terhaag, Xenofon Chatziliadis, Eleni Tzirita Zacharatou, and Volker Markl. 2025. GPU-Accelerated Stochastic Gradient Descent for Scalable Operator Placement in Geo-Distributed Streaming Systems. In *Proceedings of the 16th International Workshop on Accelerating Analytics and Data Management Systems Using Modern Processor and Storage Architectures (ADMS25)*.
[71] Jonas Traub, Philipp M. Grulich, Alejandro Rodriguez Cuellar, Sebastian Breß, Asterios Katsifodimos, Tilmann Rabl, and Volker Markl. 2019. Efficient Window Aggregation with General Stream Slicing. In *22nd International Conference on Extending Database Technology, EDBT*. 97–108.
[72] Juliane Verwiebe, Philipp M Grulich, Jonas Traub, and Volker Markl. 2022. Algorithms for windowed aggregations and joins on distributed stream processing systems. *Datenbank-Spektrum* 22, 2 (2022), 99–107.
[73] Massimo Villari, Maria Fazio, Schahram Dustdar, Omer F. Rana, and Rajiv Ranjan. 2016. Osmotic Computing: A New Paradigm for Edge/Cloud Integration. *IEEE Cloud Comput.* 3, 6 (2016), 76–83.
[74] Yuanli Wang, Lei Huang, Zikun Wang, Vasiliki Kalavri, and Ibrahim Matta. 2025. CAPSys: Contention-Aware Task Placement for Data Stream Processing. In *Proceedings of the 20th European Conference on Computer Systems (Eurosys 2025)*. ACM.
[75] Le Xu, Boyang Peng, and Indranil Gupta. 2016. Stela: Enabling stream processing systems to scale-in and scale-out on-demand. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 22–31.
[76] Le Xu, Shivaram Venkataraman, Indranil Gupta, Luo Mai, and Rahul Potharaju. 2021. Move fast and meet deadlines: Fine-grained real-time stream processing with cameo. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 389–405.
[77] Xiaoyan Yang, Hock Beng Lim, Tamer M Özsu, and Kian Lee Tan. 2007. In-network execution of monitoring queries in sensor networks. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. 521–532.
[78] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. 2008. Wireless sensor network survey. *Computer networks* 52, 12 (2008), 2292–2330.
[79] Ossama Younis and Sonia Fahmy. 2004. HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach for Ad Hoc Sensor Networks. *IEEE Trans. Mob. Comput.* 3, 4 (2004), 366–379.
[80] Hai Yu, Ee-Peng Lim, and Jun Zhang. 2006. On in-network synopsis join processing for sensor networks. In *7th International Conference on Mobile Data Management (MDM'06)*. IEEE, 32–32.
[81] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. 2016. Apache spark: a unified engine for big data processing. *Commun. ACM* 59, 11 (2016), 56–65.
[82] Steffen Zeuch, Ankit Chaudhary, Bonaventura Del Monte, Haralampos Gavriilidis, Dimitrios Giouroukis, Philipp M. Grulich, Sebastian Breß, Jonas Traub, and Volker Markl. 2020. The NebulaStream Platform for Data and Application Management in the Internet of Things. In *10th Conference on Innovative Data Systems Research, CIDR*.
[83] Steffen Zeuch, Eleni Tzirita Zacharatou, Shuhao Zhang, Xenofon Chatziliadis, Ankit Chaudhary, Bonaventura Del Monte, Dimitrios Giouroukis, Philipp M. Grulich, Ariane Ziehn, and Volker Markl. 2020. NebulaStream: Complex Analytics Beyond the Cloud. *Open J. Internet Things* 6, 1 (2020), 66–81.
[84] Ariane Ziehn, Jan Szlang, Steffen Zeuch, and Volker Markl. 2025. Unraveling the Impact of Window Semantics: Optimizing Join Order for Efficient Stream Processing. *Proceedings of the VLDB Endowment* 18, 8 (2025), 2468–2481.