Efficient Placement of Decomposable Aggregation Functions for Stream Processing over Large Geo-Distributed Topologies

Xenofon Chatziliadis
Technische Universität Berlin
x.chatziliadis@tu-berlin.de

Eleni Tzirita Zacharatou
IT University of Copenhagen
elza@itu.dk

Alphan Eracar
Technische Universität Berlin
eracar@campus.tu-berlin.de

Steffen Zeuch
Technische Universität Berlin
steffen.zeuch@tu-berlin.de

Volker Markl
Technische Universität Berlin & DFKI
volker.markl@tu-berlin.de

ABSTRACT

A recent trend in stream processing is offloading the computation of decomposable aggregation functions (DAF) from cloud nodes to geo-distributed fog/edge devices to decrease latency and improve energy efficiency. However, deploying DAFs on low-end devices is challenging due to their volatility and limited resources. Additionally, in geo-distributed fog/edge environments, creating new operator instances on demand and replicating operators ubiquitously is restricted, posing challenges for achieving load balancing without overloading devices. Existing work predominantly focuses on cloud environments, overlooking DAF operator placement in resource-constrained and unreliable geo-distributed settings.

This paper presents NEMO, a resource-aware optimization approach that determines the replication factor and placement of DAF operators in resource-constrained geo-distributed topologies. Leveraging Euclidean embeddings of network topologies and a set of heuristics, NEMO scales to millions of nodes and handles topological changes through adaptive re-placement and re-replication decisions. Compared to existing solutions, NEMO achieves up to 6× lower latency and up to 15× reduction in communication cost, while preventing overloaded nodes. Moreover, NEMO re-optimizes placements in constant time, regardless of the topology size. As a result, it lays the foundation to efficiently process continuous data streams on large, heterogeneous, and geo-distributed topologies.

PVLDB Reference Format:
Xenofon Chatziliadis, Eleni Tzirita Zacharatou, Alphan Eracar, Steffen Zeuch, and Volker Markl. Efficient Placement of Decomposable Aggregation Functions for Stream Processing over Large Geo-Distributed Topologies. PVLDB, 17(6): 1501 - 1514, 2024.
doi:10.14778/3648160.3648186

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/xchatzil/NEMO.

1 INTRODUCTION

Internet of Things (IoT) applications often require real-time analysis of large amounts of raw sensor data sourced from distributed locations across vast geographical areas. For instance, in a smart grid scenario described by Lepping et al. [33], an electricity provider monitors millions of sensors in wind turbine farms and solar panels distributed across various geographical regions. The generated sensor data is then transmitted over the Internet to the cloud for real-time analysis. To reduce latency and communication overhead, the provider uses stream processing engines (SPEs) to perform partial computations on devices outside the cloud, such as Raspberry PIs at the edge or fog servers. The paradigm of leveraging resources of the cloud, fog, and edge is called *osmotic computing* [63].

To handle the fact that processing nodes can be overloaded by the huge amounts of data generated by such IoT applications, SPEs often scale horizontally on multiple computing nodes using *data parallelism*. This involves increasing the number of parallel instances (replicas) of operators, with each replica processing a subset of incoming data in parallel [26, 28]. A popular approach is hereby to parallelize the computation of DAFs, such as min, max, count, or sum, which are highly prevalent in stream processing [62, 67] and can be easily replicated due to their decomposability [39, 41, 53].

Challenges. Placing DAFs closer to the data source offers many benefits but also presents new challenges. In particular, cloud-based SPEs operate on a well-defined infrastructure with high-end servers and reliable network connections. In contrast, osmotic computing environments involve geo-distributed, resource-constrained devices and exhibit dynamism, volatility, and large scale [13, 54]. The unpredictable rate at which the sources produce data and the large number of resource-constrained devices, make it impossible to configure the operator replication degree of DAFs manually, potentially leading to quality of service (QoS) violations and node overloading. Therefore, an effective operator placement (OP) approach for osmotic computing environments must consider the following key aspects. First, it should scale seamlessly to extremely large topologies comprising millions of nodes. Second, it should efficiently handle topological changes that are inherent in dynamic environments (e.g., mobile devices) [48]. Third, it should prevent the over-utilization of resource-constrained nodes.

State-of-the-Art. Previous research in stream processing has primarily targeted cloud topologies and treats DAFs as general operators. On the one hand, researchers have explored various techniques to identify a *good* placement considering different modeling assumptions, optimization goals, and heuristics [7, 22, 32, 49]. On the other hand, operator replication techniques were investigated to prevent over-utilization during reconfiguration of deployed workloads [3, 23, 30, 37, 40]. However, none of these approaches

Proceedings of the VLDB Endowment, Vol. 17, No. 6 ISSN 2150-8097.
doi:10.14778/3648160.3648186

have considered replication as part of the optimization process during operator placement. They all perform placement first and only handle replication during runtime if performance problems occur. However, existing SPEs show that this two-stage approach leads to significant downtimes, as it requires a complete re-computation of the placement and a re-scheduling of the application [8, 20]. The first to propose a combined approach that *jointly* handles replication and placement are Cardellini et al. [8]. To provide an optimal solution to this NP-hard problem, they use integer linear programming (ILP). However, this solution does not scale for applications on large topologies with hundreds of nodes or involving wireless devices [7], as the search space grows exponentially in such environments.

A second line of research explores scalable aggregation approaches for dynamic and large geo-distributed topologies, especially in Wireless Sensor Networks (WSNs). WSNs achieve scalable DAF computation through hierarchical data aggregation, reducing both load and network traffic [65]. In this approach, sensors do not transmit data to the nearest base station directly; instead, they form a hierarchical network dynamically where DAFs are replicated to consolidate data at intermediate nodes. These aggregation networks are typically organized into clusters, trees, chains, or a combination of these structures. Aggregation approaches in WSNs are typically designed for managing periodic or event-triggered data collection from battery-powered, wireless devices to the nearest base station. However, stream processing in osmotic computing environments involves continuous, real-time processing of multiple data streams that extend to cloud servers and gateways beyond the nearest base station. It also requires handling more complex operations, such as window functions. Finally, existing approaches in WSNs are resource-agnostic and may lead to node over-utilization [41, 65].

Our Solution. In this paper, we introduce NEMO, a scalable, resource-aware placement approach tailored to SPEs in *osmotic computing* environments, specifically addressing the placement and replication of DAF operators. Drawing inspiration from WSNs, NEMO achieves load balancing and reduces communication by replicating DAFs and forming an aggregation tree among them. Furthermore, instead of solving the NP-hard OP problem on a discrete set of nodes, NEMO takes a different approach by projecting the network topology using Network Coordinate Systems (NCS) into a continuous space. This transformation enables NEMO to approximate the OP problem using an iterative algorithm. As a result, NEMO can efficiently identify the degree and placement of replicas for large topologies with millions of nodes in linear time.

Results. We compare NEMO against heuristics used in SPEs and adaptive aggregation approaches used in WSNs. Furthermore, we integrate NEMO in the optimizer of the IoT data management system (IoTDMS) NebulaStream [68]. We conduct experiments on various settings and workloads based on both simulations and an end-to-end deployment on a local cluster of Raspberry PIs. Unlike state-of-the-art approaches that are resource oblivious and may overload nodes, NEMO avoids over-utilization entirely. Furthermore, NEMO achieves latency improvements of up to 6× and communication cost reduction of up to 15× for various deployments. Finally, NEMO can re-optimize placements in constant time.

The remainder of this paper is structured as follows. In Section 2, we present the necessary background information. Section 3 describes the metrics and system model of our approach, along with a formal problem definition and accompanying proofs. Our approach is described in detail in Section 4, and its evaluation is presented in Section 5. We then explore related work in Section 6, followed by our concluding remarks in Section 7.

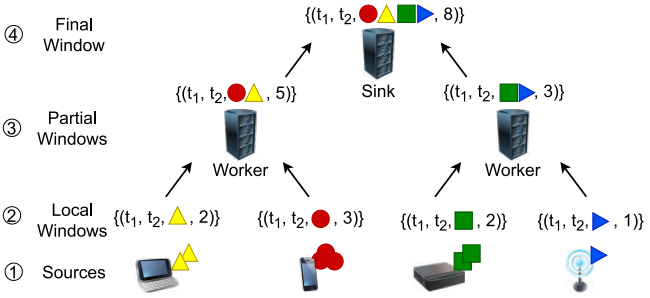


Figure 1: Example of distributed windowing, where partial count-aggregates are computed on remote worker nodes.

2 BACKGROUND

In this section, we outline the stream processing semantics of distributed windowing operators, which NEMO replicates and places across geo-distributed nodes, and introduce the concept of network coordinate systems, which serve as the basis for NEMO's heuristics.

2.1 Stream Processing Semantics

In stream processing, sources produce data streams and sinks consume them. Computation on data streams happens in parallel across multiple nodes or machines called workers. Data streams are typically unbounded and require discretization using window operators to perform computations. Window operators are usually defined by a window type, a window measure, and a window function [61]. Window types include tumbling (fixed-size, non-overlapping) and sliding (overlapping) windows. Window measures are time-based (triggered by a timestamp) or count-based (ending after a certain number of events). Window functions perform computations on window records, and are categorized as decomposable (incrementally computed) or holistic (requiring access to all records).

SPEs can distribute load by replicating window operators with decomposable functions and deploying them on partitioned data subsets that can be processed independently on different nodes. Consequently, load distribution involves distributed window computation. Figure 1 depicts an example of counting the total number of events with distributed windows on a topology with four sources and two worker nodes. The sources produce in ① individual events from the data streams and generate in ② local window slices, which are discrete partitions of the data streams that map an event to a time interval. Next, in ③, the workers calculate partial intermediate aggregates from the window slices. Finally, in ④, the sink node receives the partial window aggregates and computes the final result through window merging. Distributed windowing allows processing window slices independently, resulting in a hierarchical processing tree [5]. This allows for greater flexibility in data processing and enables nodes to operate independently. However, distributed windowing is restricted to decomposable functions, as holistic functions do not allow partial aggregate computation [67].

2.2 Network Coordinate Systems

Distributed systems often need to identify low-latency network paths to meet application requirements [49, 51]. A naive approach is to measure the round-trip times (RTT) among all nodes, which incurs a high overhead in large distributed systems. To reduce the number of measurements, Network Coordinate Systems (NCSs) have been proposed, which predict latencies with an estimation error instead of relying solely on direct measurements.

NCSs typically follow two approaches [15]. Most traditional NCSs are Euclidean-based NC (ENC) systems [17, 19, 45, 46] that create an abstraction of the topology and map all nodes to an n-dimensional cost space [49, 51]. Every node $v \in V$ in the cost space has a position $\vec{x}_v$ such that the Euclidean distance $d(\overrightarrow{v_i v_j})$ between two arbitrary nodes v_i and v_j corresponds to the latency estimate. ENC systems require the estimates to satisfy the triangle inequality, which is a fundamental property of distances in Euclidean spaces stating that the shortest distance between two points is a straight line. However, Internet latencies often violate this inequality due to the complexity of network routing, leading to inaccuracies in ENC systems. Matrix factorization-based approaches (MFNC) address this limitation by removing the triangle inequality constraint, typically resulting in better prediction accuracy than ENC systems [14, 34, 42]. However, MFNC approaches do not map nodes to a Euclidean space, which provides advantages like intuitive distance calculations, straightforward geometric interpretations, and the ability to apply Euclidean algorithms and transformations for various analyses and optimizations [7, 49, 51].

3 CONCEPTS AND FORMALISM

SPEs take a user query as input and create a *logical operator plan* that represents the processing pipeline and specifies the order and type of operations and their dependencies. Figure 2a shows a logical operator plan for an example of distributed windowing introduced in Section 2.1. The plan has multiple sources, a slice creation operator, a slice merging operator, and a window computation operator. The *replication plan* extends the basic logical operator plan by specifying the number of replicas of each operator. Figure 2b shows a replication plan variant that includes multiple slice creation operator instances. Finally, the physical plan in Figure 2c captures the mapping from logical operators (including replicas) to physical nodes. In this example, the three instances of the slice creation operator are placed on different nodes to balance the workload. NEMO takes a logical plan, like Figure 2a, as input and determines the number of replicas and their placement.

3.1 Metrics

NEMO uses latencies and computational capacities as metrics for making operator placement and replication decisions.

Latency. In a NCS, the proximity between two nodes corresponds to their latency. NEMO assumes that all nodes have coordinates in a NCS and can communicate with each other directly or via intermediate hops. The lowest latency in the NCS occurs when two nodes $v_i, v_j \in V$ communicate directly, which corresponds to their Euclidean distance. If the nodes communicate via an intermediate node v_t, the latency is $d(\overrightarrow{v_i v_t}) + d(\overrightarrow{v_t v_j}) \geq d(\overrightarrow{v_i v_j})$. However, in real-world networks, attaining the lowest latency does not necessarily result from direct communication due to the violation of the triangle inequality (TIV) (cf. Section 2.2). We provide a more in-depth exploration of the implications of TIV in Section 5.

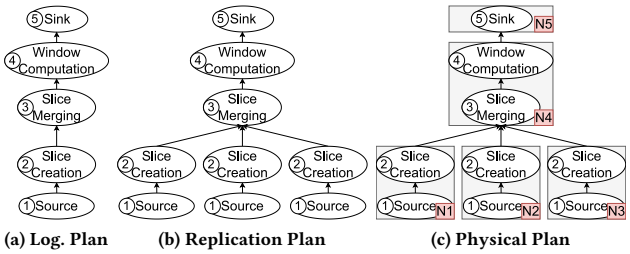


Figure 2: Example of a) a logical plan with distributed windowing operators, b) its replication plan, and c) the physical plan with the placed operators on five nodes (N1-N5).

Capacities. We model the maximum computational capacity of a node v_i as $C_t(v_i) \in \mathbb{N}$. Nodes with high capacities are typically servers within the cloud, while those with lower ones are edge or sensor devices. We consider v_i to be overloaded if $C_u(v_i) > C_t(v_i)$, where C_u represents the utilized capacity. Finally, C_r represents the required capacity, and $C_a = C_t - C_u$ the available capacity.

3.2 System Model

In the following, we formally describe the underlying concepts of our approach, including the logical operator plan, replication of operators, and the resulting replication plan.

Logical Plan. We model a query as a directed acyclic graph (DAG) of connected operators as proposed by Rizou et al. [51]. Hereby, an operator DAG $G = \{\Omega, S, A, L\}$ comprises a set $\Omega = \{\omega_1, ..., \omega_n\}$ of *operators* connected by a set $L = \{\overline{\omega_1\omega_i}, ..., \overline{\omega_j\omega_n}\}$ of *links*, where link $\overline{\omega_i\omega_j} \in \Omega \times \Omega$ denotes that operator ω_i produces a stream consumed by operator ω_j. $L_{\omega i}$ represents the set of in- and out-going links attached to operator ω_i. For each $\overline{\omega_i\omega_j}$, the weight $w(\overline{\omega_i\omega_j})$ denotes the extent to which the capacity of the node containing ω_j is utilized. The value of $C_u(v_i)$ is thus calculated as the sum of the weights of all incoming links to node v_i. Additionally, we designate two subsets $S \subset \Omega$ and $A \subset \Omega$ of operators as sources and sinks, respectively. Sources only have outgoing links (i.e., they only produce streams) and sinks have only incoming links (i.e., they only consume streams). Sources and sinks are pinned operators, i.e., their mapping to physical nodes is fixed, while other operators can be assigned freely to any available node in V.

Replication. To model replication, we define an operator $\omega \in \Omega$ using a set of attributes: $\omega = \{\omega_{id}, R_{id}, v_i, \rho\}$. The combined attributes operator ID ω_{id} and replication ID R_{id} create a unique identifier. v_i denotes the node in the physical topology where the operator is located, while ρ represents the number of operator instances, which can be between 1 and in-deg(ω). Pinned operators have a pre-defined fixed placement and are, thus, not replicated.

Note that in cloud-based SPEs, parallel processing is often achieved by replicating operators and running them on partitioned data [26, 28]. The partitioning process involves breaking down large data

streams into smaller parts that can be processed independently, typically through the application of a hash function [26]. In fog/edge topologies, data is often generated by multiple geo-distributed sources and is thus already partitioned inherently.

To limit the number of possible replicas and ensure that load is reduced at every aggregation level, NEMO leverages the geo-distribution-based partitioning scheme to set $w(\overline{\omega_i\omega_j}) = 1$ and limit the number of maximum replicas to the number of geo-distributed sources, i.e., max in-deg$(\omega) = |S|, \forall\omega \in \Omega$. Although it is realistic to assume that the load is reduced at every aggregation level in WSNs, certain stream processing workloads may deviate from this norm. In such scenarios, intermediate operators may require additional resources, possibly due to occurrences of backpressure, necessitating the distribution of the data stream among multiple downstream nodes [23]. In Section 4.4, we further describe an extension of NEMO that enables operator placement and replication for arbitrary link weights and replica numbers, thereby allowing to further partition a stream during intermediate aggregation.

Replication Plan. A replication plan G^* is an extension of the logical plan with replicated operators. G and G^* distinguish themselves by $|\Omega|$ and $|L|$. The total number of operators in the replication plan $|\Omega^*| \in G^*$ is equal to the sum of all replicas for each operator, where $|\Omega| \leq |\Omega^*|$. The number of theoretical links between two non-replicated operators $\omega_j, \omega_k \in \Omega$ can be $|L(\omega_j, \omega_k)| = \rho_j \times \rho_k$. To prevent duplicate tuples in the final results and comply with partitioning restrictions, there are limitations on the links between replicas of the same operator. We inherit the linking model from WSNs [65], where nodes usually transmit data to a single destination to minimize energy utilization and network traffic. Formally this means that replicas cannot share the same input, which greatly reduces the number of possible links. We express this constraint as $\forall\omega(\omega \in L_j \rightarrow \omega \notin L_k)$, where L_j and L_k are the sets of incoming links for two replicas of the same operator. We denote the superset of all logical plans that contain all permutations of possible paths between operators for all possible ρ with G'.

3.3 Problem Definition

General Operator Placement Problem. In a distributed stream processing system with a set of nodes V and a logical plan G, the general problem of operator placement is to determine the optimal placement of operators on nodes while adhering to given constraints and optimization objectives. We model this problem with a binary function that maps each operator $\omega \in \Omega$ to a computing node $v \in V$, where map$(\omega, v) = 1$ if the operator is deployed on node v, otherwise map$(\omega, v) = 0$. The general operator placement problem can be reduced to the general assignment problem, which is well-known to be NP-hard [31].

Operator Placement and Replication Problem. Cardellini et al. [8] showed that the general operator placement problem can be extended to the operator placement and replication problem (OPR), which seeks the optimal degree of replication for all operators in addition to their placement, based on the given optimization objective and constraints. Since this problem is an extension of the general operator placement problem, it is also NP-hard.

Problem Formulation. The goal of OPR is to minimize the total aggregated latency Lat(Ω^*) for all paths of operators from the sources to the sinks while avoiding over-utilized nodes. Formally, we have the following optimization problem:

$$\min \text{Lat}(\Omega^*) = \sum_{\forall\omega_x,\omega_y\in\Omega^*} d(\overrightarrow{\omega_x\omega_y}), \forall\Omega^* \in G^*, \forall G^* \in G' \tag{1}$$

subject to the constraint

$$C_u(v_i) \leq C_t(v_i), \forall v_i \in V. \tag{2}$$

3.4 Proofs

NEMO uses a heuristic approach to find a practical solution to the OPR problem. This approach is based on insights derived from the following theorems and their proofs.

Theorem 3.1. *The unconstrained optimal placement of all replicated operators is equivalent to the unconstrained optimal placement of the corresponding non-replicated operator.*

Proof. Our approach determines replication based on the capacity of the optimal host node $v'(\omega)$ of a non-replicated operator. When $C_u(v'(\omega)) > C_t(v'(\omega))$ after map$(\omega, v) = 1$, the operator is replicated and distributed on other nodes to reduce the load in v'. If $C_t(v'(\omega)) = \infty$, then $C_u(v_i) < C_t(v_i)$. In such cases no replication is necessary, which corresponds to the optimal placement of the non-replicated operator. □

Theorem 3.2. *The unconstrained optimal placement for all replication plans is equivalent to the optimal placement of the corresponding non-replicated logical plan.*

Proof. Theorem 3.1 proves that the placement of a single operator is independent of ρ, in case $C_t(v'(\omega)) = \infty$, as no replication of that operator is required. If $C_t(v_i) = \infty, \forall i \in V$, all operators can be placed at their optimal nodes and thus require no replication. □

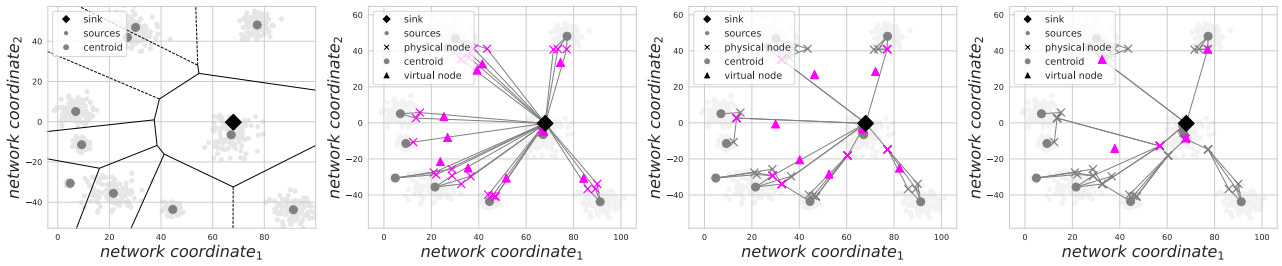
Theorem 3.3. *The unconstrained optimal placement defined in Equation* (1) *is a convex function. Therefore, an optimal placement of all operators can be obtained through the optimal placement of each individual operator. Formally, this can be expressed as:*

$$min\ Lat(\omega_i) \cup ... \cup Lat(\omega_n) = min\ Lat(\omega_i, ..., \omega_n) \tag{3}$$

Proof. As stated by Rizou et al. [51], the optimal placement of a single unpinned operator (SOP) is equivalent to solving the well-known Weber problem [9]. The convex objective function of the Weber problem allows approximation through an iterative algorithm. Solving the SOP problem for all operators leads to eventually reaching a local optimal position for each, resulting in an all-local optimal solution. The convexity property guarantees that any local optimum is a global optimum, confirming that an all-local optimal solution is also globally optimal. □

4 NEMO

This section describes NEMO, our scalable, resource-aware approach for determining the replication factor and placement of DAFs. NEMO is tailored to large-scale, heterogeneous, geo-distributed topologies in osmotic computing environments. It transforms a logical plan into a physical one considering latency and available



(a) Grouping (b) Iteration 1 (c) Iteration 2 (d) Iteration 3

Figure 3: Progression of NEMO: In each iteration, a new aggregation level is added to reduce the number of incoming links to the sink. The final aggregation nodes before transmitting data to the sink are highlighted.

resources. To this end, NEMO structures the topology hierarchically and identifies the level of replication for DAFs. NEMO's main optimization goal is minimizing latency to ensure IoT applications meet their low latency requirements, while avoiding node overloading. It can be also extended to accommodate additional constraints and optimization objectives represented in a cost space. Next, we introduce NEMO's heuristics (Section 4.1), describe its different phases (Section 4.2) and re-optimizations (Section 4.3), and discuss extensions (Section 4.4).

4.1 Heuristics

Solving OP on a *discrete* set of nodes is NP-hard [51]. However, in a *continuous* space, OP can be solved with an iterative algorithm (cf. Theorem 3.3). Therefore, NEMO's key idea is projecting the topology to a Euclidean space with a NCS. Although this projection may introduce latency errors due to triangle inequality violations (cf. Section 2.2), it reduces network overhead for collecting latency measurements and increases NEMO's robustness against continuous latency noise in osmotic computing environments. Furthermore, it enables NEMO to perform near-optimal operator placement in linear time and re-optimizations in constant time.

For determining the placement we assume the existence of a virtual node (i.e., NCS node without a mapping to a physical node) at each operator's optimal location. With this assumption, a mapping between virtual and real nodes can be computed efficiently using a neighborhood search that identifies real nodes with the smallest distance from the virtual ones. To quickly adapt to changes (e.g., device unavailability), NEMO leverages Theorem 3.3 to calculate the global optimum for each operator independently. As a result, it can handle changes by relocating affected operators to the next available nodes near the original virtual node without re-solving the operator placement problem. The mapping from the continuous NCS space to physical nodes introduces errors, as physical and virtual nodes may not always be close to each other. These errors lead to minor deviations from the optimum in terms of latency but make NEMO computationally efficient, allowing it to scale to large topologies (cf. Section 5.3).

4.2 Phases in NEMO

NEMO is an iterative approach with three phases, 1) pre-processing, 2) virtual operator placement, and 3) re-assignment and replication. In Figure 3, we visualize the progression of NEMO on a simulated topology with 1000 nodes. In the initial phase (Figure 3a), NEMO limits the search space by grouping nodes into clusters based on their latency in the cost space and identifies each cluster's centroid. For visualization purposes, the Figure also shows the Voronoi diagram of the centroids. Next, NEMO iteratively creates an aggregation tree over the topology, determining the level of replication for DAFs. Figures 3b-d depict the evolution of the aggregation tree across the initial three iterations. Each iteration introduces a new level in the aggregation tree, systematically reducing the number of incoming streams at the sink. In the figures, we highlight the virtual nodes (△) that correspond to the optimal placement location in the cost space and the final physical cluster heads (**x**). Note that cluster heads are always close to a virtual node. In the remainder of this section, we present each phase in detail.

First Phase: Grouping the Cost Space. The initial phase of NEMO serves as a pre-processing phase, where the nodes in the cost space are divided into distinct groups with minimal latency, as depicted in Figure 3a. This grouping restricts the search space for the subsequent phases of NEMO. The objective hereby is to minimize the latency between all nodes within the same group, while maximizing the latency to nodes outside the group. The goal of our optimization function is, therefore, to maximize the mean of the silhouette coefficient $s(i)$ over all grouped nodes in the cost space [52]. For each cluster C_I and node $i \in C_I$, the silhouette coefficient is defined as:

$$s(i) = \begin{cases} \frac{b(i)-a(i)}{\max(a(i),b(i))} & \text{if } |C_I| > 1, \\ 0 & \text{if } |C_I| = 1 \end{cases} \tag{4}$$

Hereby $a(i)$ defines the cohesion of nodes within the same cluster and $b(i)$ defines the separation of nodes between different clusters:

$$a(i) = \frac{1}{|C_I| - 1} \sum_{j \in C_I, i \neq j} d(i,j) \tag{5}$$

$$b(i) = \min_{J \neq I} \frac{1}{|C_J|} \sum_{j \in C_J} d(i,j) \tag{6}$$

$|C_I|$ and $|C_J|$ are the number of nodes in clusters I and J respectively. The distance function $d(i,j)$ defines the latency between nodes i and j, which corresponds to the Euclidean distance in the

cost space. Clusters can be created using any arbitrary clustering approach. In Fig. 3, NEMO uses the *k-means* algorithm [36].

Second Phase: Virtual Operator Placement. In the second phase, NEMO calculates the optimal placement of an operator in the cost space, referred to as a virtual node, between a given set of upstream nodes and the sink. Algorithm 1 shows the pseudocode of NEMO. It takes as input the source nodes grouped by their clusters, the sink node, and an optional parameter that controls the number of aggregation levels of the tree. When the required capacities of the current upstream nodes (which initially correspond to the sources) exceed the available capacities of the sink, NEMO enters an iterative process. In each iteration, NEMO creates a new partial window aggregation operator to extend the aggregation tree with an additional level. The goal of each iteration is to identify a set of physical nodes where the operator can be replicated. The physical nodes for each cluster are stored in a lookup table (line 5). Then, NEMO iterates over the clusters and determines the virtual node for each of them using an iterative optimization algorithm (line 7). Specifically, NEMO uses the spring relaxation algorithm of Fruchterman and Reingold [25]. The main idea is to model each link in the DAG of operators as a spring. Then, for each spring $s_i \in L$ the low energy state can be determined by minimizing the sum of potential energies E_i stored in the springs:

$$arg\min_{\vec{s_i}} \sum_i E_i = \sum_i \frac{1}{2} k_i \vec{s_i}^2 \tag{7}$$

Using spring relaxation to determine the placement has several advantages. First, its iterative nature allows seamless re-assignments of operators to new nodes in case of topology changes. Second, spring relaxation is decentralized and does not require coordination between nodes, allowing each latency cluster to operate independently. Finally, it can be naturally extended to recursively add and compute the placement of new operators (e.g., partial windows, window merging) and thus build a multi-hierarchical tree. This feature is especially useful if additional levels are required, for example, to further reduce load, enable fault tolerance, or meet security restrictions in some networks.

Third Phase: Re-assignment and Replication. Finally, NEMO determines the number and placement of replicas for each newly created operator ω_{ik} in cluster k and iteration i to distribute the load more evenly (line 8). To that end, it iterates through a sorted list n_k of (v_i, C_a) tuples (nodes and their available capacities), allocating capacities to each node until the total number of required capacities is exhausted. The result is a placement list indicating the nodes hosting the replicas and their allocated capacities. For example, $C_r = 8$, and $n_k = [(3,2),(1,4),(5,6),(2,10),(4,3)]$, results in $p_k = [(3,2),(1,4),(5,2)]$. The replication factor for ω_{ik} is thus $\rho_{ik} = |p_k|$.

The elements in n_k are initially nodes in the group with available resources above a certain threshold t. If n_k does not provide enough resources, NEMO augments it with potential cluster heads from adjacent groups. This process is repeated until nodes with sufficient resources for intermediate aggregation are found. In the worst case, NEMO needs to consolidate all groups.

NEMO sorts the list n_k using a distance function that penalizes nodes with significantly lower capacities than others in their group. This reduces the number of replicas and prevents the selection of

Input: *clustered_sources*, *sink*, *limit*

```
1  level ← 0;
2  av ← available_capacity(sink);
3  current_nodes ← clustered_sources;
4  while av < load(current_nodes) & level ≤ limit do
5  |  pn ← map() ;                          // physical nodes
6  |  foreach cluster, nodes ∈ current_nodes do
7  |  |  vn ← get_optimum(nodes, sink) ; // virtual node
8  |  |  pn[cluster] ← reassign(nodes, vn);
9  |  end
10 |  current_nodes ← merge_clusters(pn);
11 |  level ← level + 1;
12 end
```

Algorithm 1: The NEMO approach

nodes with proportionally fewer resources as cluster heads. The function to calculate the penalized distance from a real node v_i to a virtual node v_v is defined as:

$$d'(\overrightarrow{v_v v_i}) = \frac{C_r(v_i)}{C_a(v_i)} * d(\overrightarrow{v_v v_i}). \tag{8}$$

The physical nodes hosting the newly created replicas are the cluster heads for the recently formed aggregation level. After each iteration, NEMO sets the cluster heads of the current level as the new upstream nodes. Additionally, it merges clusters if cluster heads in different groups overlap or are in close proximity. This iterative process continues until the required capacities of the sink do not exceed its available capacities. Each iteration reduces the number of required capacities, guaranteeing NEMO's convergence.

4.3 Re-optimizations for Partial Changes

NEMO supports re-optimizations without the need to recompute the entire operator placement when: 1) Adding sources (leaf nodes) and workers (i.e., nodes with no active role in the current workload) or, 2) Removing sources, cluster heads, and workers. We note that changes in node coordinates are inherent in NCSs due to route changes in IP-based routing protocols. NEMO addresses high divergences in latency estimation errors (e.g., mobile devices) by removing and re-adding nodes to the topology. Section 5.6 discusses further details on our experiences with real topology changes.

Node Addition. When adding a new node to the topology, NEMO computes at first its coordinates in the cost space. This involves collecting latency measurements from a specified number of nodes within its NCS neighborhood. The optimal coordinates are subsequently determined by minimizing the relative distance error, ensuring that Euclidean distances in the cost space match the measured latencies as explained in Section 2.2. The time complexity of coordinate computation for a single node is constant, as the neighborhood size is fixed. If the new node is a source, NEMO assigns it to the latency group with the nearest centroid. If the group has available cluster heads with sufficient capacity, NEMO designates the closest one as the parent node. For efficient searches, NEMO keeps a spatial index on cluster heads and centroids and a lookup table with available cluster heads in each group. If NEMO cannot identify a suitable cluster head, it executes Algorithm 1

for the entire group. In the worst case, if NEMO cannot achieve a balanced placement within the group, it recomputes the placement for the entire topology. To avoid this, we set the threshold in the third phase to be equal to the median of link weights, so that a group can always have cluster heads with available capacities.

Node Removal. Before removing a node from the topology, NEMO determines its role. If the node serves as a source or idle worker, it can be seamlessly removed from both the topology and the physical plan. If the node is a cluster head, NEMO attempts to redistribute the workload among nodes in its subtree and other cluster heads in the same group. The redistribution uses Algorithm 1 by setting the sources of the cluster as upstream nodes and the parent of the original cluster head as the sink. If a balanced placement cannot be achieved, this process iterates over the entire group and, in the worst-case, the entire topology. Strategies to reduce the likelihood of complete re-computation include increasing the value of t, maintaining unassigned worker nodes as backups, and enlarging the size of latency groups to provide additional capacities.

4.4 Extensions

In this section, we discuss how our approach could be extended to handle arbitrary weights and restricted communication. We also describe how we can extend NEMO's optimization target.

Handling Arbitrary Weights. NEMO assumes that load is reduced at every aggregation level. However, in high-throughput stream processing workloads, additional replication of input streams may be necessary during intermediate aggregation, requiring nodes to have multiple output streams (i.e., multiple parents). In NEMO's system model, this implies that link weights w can have arbitrary values. Here, we introduce an extension of NEMO, called NEMO+, to support arbitrary weights. The main difference between NEMO and NEMO+ is the merging of the subtrees of the different latency groups (cf. Algorithm 1, line 10), which affects the list of sorted nodes n_k during the re-assignment phase. In NEMO, a tree t_1 can always be merged with another tree t_2 by either adding the root node of t_2 as a new child to $t1$ or replacing an existing child of t_1 with the root node of t_2. In NEMO+, this is no longer possible, as a node might require multiple parents if the edge weight is greater than the capacity of the parent. In NEMO+, merging subtrees of different latency groups is controlled via two hyperparameters: step size ss and merge factor mf. ss represents the spring constant for calculating the force between the sink and the virtual nodes and affects how quickly the virtual nodes converge toward the sink. mf controls the number of new clusters for each tree level when merging cluster heads from multiple latency groups. The number of new clusters is calculated as $num\_clusters = max(round(mf * num\_old\_clusters), 1)$. We note that ss and mf require tuning. A poor choice of these hyperparameters can have a detrimental effect on the placement quality and may in certain cases prevent NEMO from converging.

Handling Restricted Communication. Restricted communication between devices in osmotic environments is a common issue due to the absence of standardized communication protocols and restricted network access [2]. In these environments, nodes typically communicate only through predefined gateways or border routers, as seen, for example, in the FIT IoT Lab [1]. We address such restricted communication in NEMO in two steps. First, we replace the missing entries in the latency matrix M used for computing network coordinates. If a path exists between two nodes i, j over a gateway node g, we set $M[i, j] = M[i, g] + M[g, j]$. Additionally, we create an entry $R[i, j] = (i, g, j)$ in an internal routing table R, indicating the path between i, j. Then, we run NEMO normally on the cost space generated from M. If NEMO produces a placement between i, j, where $[i, j] \in R$, we resolve the path in the physical plan. To achieve this, we introduce forwarding operators, i.e., pinned operators on routing nodes that take an input stream and produce an identical output stream. This ensures that routing decisions are taken into account during placement.

Extending the Cost Space. To extend NEMO's optimization objective with additional metrics beyond latency (e.g., QoS), we need to integrate the new metrics into the cost space as new dimensions, as shown in Pietzuch et al. [49]. For distance-based metrics (e.g., bandwidth), Euclidean NCSs like Vivaldi [19] can generate a cost space given a matrix that represents the new distance metric between nodes. To modify the constraints of NEMO (e.g., reliability), it suffices to adapt n_k during the re-assignment phase. Specifically, the sorting and inclusion of nodes in the list can be adapted to only include nodes that meet specified constraints.

5 EVALUATION

We evaluate the performance of NEMO in two parts. In the first part (Sections 5.2 - 5.6), we evaluate NEMO based on simulations that are conducted on a local workstation within single-threaded Python scripts. In these simulations, we use NCSs that are derived from several real-world and artificial topologies. In the second part (Section 5.7), we integrate NEMO in the optimizer of the IoTDMS NebulaStream [68, 69] and evaluate an end-to-end deployment on a distributed Raspberry PI cluster using five different queries.

5.1 Experimental Setup

Hardware. The simulations (Python scripts) are run single-threaded on a local workstation with an Intel I7 9700k CPU and 32 GB of RAM. The end-to-end deployment experiments are performed on a cluster of RaspberryPi model 4B devices, each having 4 GB of RAM and a Quad-Core Cortex-A72 (ARM v8), 64 Bit, 1.5 GHz CPU. All devices are connected via Gigabit Ethernet to a switch.

Simulation. We use the Vivaldi algorithm [19] to create NCSs from latency measurements of several real-world and artificial topologies. We choose Vivaldi due to its widespread adoption [18, 38, 50, 57] and its ability to represent the topology in a Euclidean space, as NEMO's heuristics require (cf. Section 4.1). We use latency measurements of the following topologies: 1) FIT IoT Lab [1] is an IoT testbed deployed across different regions in France. Our evaluation uses RTTs of 433 geographically distributed nodes comprising different types of microcontrollers and four gateway servers. 2) RIPE Atlas [56] is a widely used Internet measurement platform that provides real-time data for network performance analysis. Our evaluation uses RTTs of 723 anchors geo-distributed across the globe. Anchors serve as fixed measurement points for latency and routing behavior. 3) PlanetLab [16] measurements represent RTTs from 335 nodes hosted by universities and research institutions across Europe and North America. 4) King [27] contains latency

measurements of 1740 Internet DNS servers. 5) We also generate artificial NCSs with varying latency distributions and sizes. The x-axis of the NCSs ranges in $[0, 100]$ and the y-axis in $[-50, 50]$. These ranges represent a combined set of latency distributions found in other topologies. Nodes belong to different Gaussian distributions with uniformly distributed centers across the plane. The artificial NCSs ranges from 1k to 1M nodes.

To identify the number of neighbors m (i.e., number of direct measurements of each node) for Vivaldi, we created and evaluated network coordinates of the tested topologies using the network coordinate simulation tool NCSIM [14]. We consider the mean absolute error (MAE) and 90th percentile absolute error (NPAE). For Ripe Atlas and FIT IoT Lab, we measured the best trade-off between accuracy and communication cost with $m = 20$. For PlanetLab and King, we set $m = 32$ based on the results presented in [14, 19, 64].

End-to-end Deployment. To evaluate the end-to-end performance of NEMO, we extended the IoTDMS NebulaStream with NEMO as the underlying approach for operator placement. The experiments are performed on a cluster comprising 11 RaspberryPIs, where one node acts as the coordinator of NebulaStream and sink. Seven nodes serve as sources and two as worker nodes that can be used for intermediate aggregation. Based on the workload, we use one additional PI interchangeably, either as a source or worker.

Workloads. The simulations are based on a generic DAF monitoring workload that collects metrics from all devices in the topology and computes window aggregates. This workload represents a common use case in many monitoring systems [4, 11, 24, 43, 53, 58]. Hereby, the load increases proportionally with the topology size as all nodes act as data sources, allowing a thorough evaluation of NEMO's performance across different scales. The selection of the sink node is randomized to ensure a fair and unbiased representation. The simulation uses the semantics of distributed window computation with four operators: sources, partial windows, final window, and sink as explained in Section 2.1. We model various loads by varying the window aggregation types, amount of collected metrics, and ingestion rates through link weights and capacities as explained in more detail below.

In the end-to-end deployments, we evaluate NEMO on five different queries that vary by their window types and ingestion rates. The first query is a monitoring query used by NebulaStream to collect network metrics from the nodes in its topology. The query uses a tumbling window with a length of one second keyed by the node ID. For this workload, we use 8 sources and 2 worker nodes, with each source generating 10 events per second, resulting in a total of 80 events per second. The remaining four queries are based on the DEBS 2013 grand challenge, where data is generated by sensors embedded in the shoes of soccer players [44]. Specifically, we evaluate NEMO on the following DEBS queries: 1) tumbling window with a length of 1 second, 2) tumbling window with a length of 1 minute, 3) sliding window with a length of 1 second, emitted every 50ms, 4) sliding window with a length of 1 minute, emitted every 50ms. In this workload, each source produces 200 events per second. As these queries have a significantly higher ingestion rate than the monitoring query, we use 7 sources and 3 worker nodes. To test the approaches under different load conditions, we run experiments using two setups. In the first, we run the workloads normally on the topology. In the second, we use the Linux stress tool to occupy all CPU cores and 80% of the available memory of the sources. We run each experiment for 5 minutes.

Capacities and weights. Systems like Flink, Spark, Storm, or NebulaStream rely on a configuration file that specifies the computational capacities of workers. To show that NEMO supports a wide range of capacities, in our simulations we evaluate it on different capacity distributions, representing different levels of heterogeneity. To ensure consistent results across all experiments, we change the individual node capacities while maintaining a nearly constant total sum of capacities. Slight deviations may occur due to rounding. The capacities follow a log-normal distribution with $\sigma = 1.4, \mu = 7.3$. We vary the range of capacities assigned to nodes from $[50, 50]$ to $[0, 350]$. In all distributions, the mean is fixed at 50, while only the median changes.

To assess the performance of NEMO+ under varying loads, we assign different link weights (w) to sources. Initially, we conduct tests with uniform link weights ($w = 1$) for all sources, establishing a baseline for comparison with NEMO and other aggregation approaches. Furthermore, we evaluate NEMO+ with all sources having $w = 2$ and in a configuration where each source is assigned a random weight following a log-normal distribution within the range $[1, 50]$. In the weight simulations, the median is not fixed, allowing for a total increase in load.

In the end-to-end deployments, we determine capacities empirically. Following the execution and analysis of workloads, we observe that each worker node can effectively handle four and three sources for the monitoring and DEBS workload, respectively. Consequently, we assign the total capacity (C_t) of worker nodes to 4 for monitoring and 3 for DEBS. We set the capacities of sources and sinks to 1, ensuring that no more operators other than the pinned operators are placed on these nodes. Additionally, we set link weights between operators to 1, as no further load balancing between intermediate aggregations is required.

Baselines. We compare NEMO against the following baselines: 1) Optimal: We implement an optimal solution based on Cardellini et al. [8]. 2) Bottom-Up: A heuristics approach used by NebulaStream, where all DAFs are pushed down to the data sources, aiming to optimize the processing at the source nodes [12]. 3) Top-Down: A heuristics approach used by NebulaStream, where all DAFs are placed at the sink node, aiming to optimize the aggregation process at the sink [12]. 4) LEACH [29]: A cluster-based approach commonly used in WSNs, which pre-aggregates data from nearby sources at randomly selected cluster heads. In our experiments, we use a central implementation of LEACH that uses k-d trees for the neighborhood search. We define the number of cluster heads to be equal to 10% of nodes in the topology, as suggested by [29]. 5) LEACH-SF [55]: A variant of LEACH that uses fuzzy c-means [6] for clustering. It employs an additive weighting scheme for cluster head selection, considering sink distance and centroid distance. Our extension augments LEACH-SF's weighting scheme with node capacities. 6) MST: A greedy approach based on the Prim algorithm [47], which is a common representative for tree-based aggregation in WSNs. MST constructs a minimum spanning tree from the sources to the sink, enabling data aggregation at intermediate levels. 7) Chain: A chain-based method used in WSNs, which creates a chain from all sources to the sink and aggregates data at every node in between. Our implementation is based on a

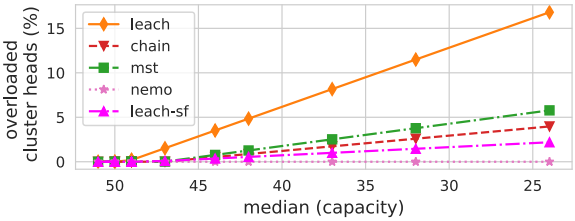


Figure 4: Impact of capacity distributions towards overloaded cluster heads for the simulated (1000 nodes) topology.

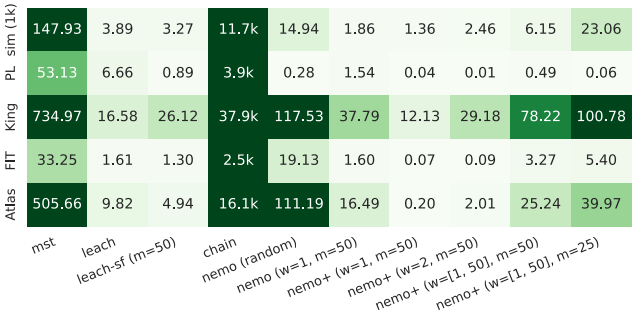


Figure 5: Comparison of the 90th percentile latency deltas against bottom up/top down across different approaches.

central probabilistic approach that uses stochastic gradient descent and simulated annealing. 8) NEMO+: The extension of NEMO that supports arbitrary weights where we tune the *merge factor* and *step size* for each topology and weight distribution.

5.2 Mitigation of Over-Utilization

In this section, we evaluate NEMO's effectiveness in preventing over-utilized nodes. Figure 4 compares the four WSN approaches to NEMO regarding their percentages of overloaded nodes for different capacity distributions. We exclude the optimal approach as it did not produce results within a reasonable time. The topology's heterogeneity and the number of resource-constrained nodes increase with decreasing median capacity, as explained in Section 5.1. The percentage of overloaded nodes is calculated as the ratio of overloaded cluster heads to the total number of cluster heads.

As shown in Figure 4, NEMO consistently outperforms the other approaches across all capacity distributions due to its distinct advantage of having zero overloaded nodes. In comparison, the WSN approaches LEACH-SF, MST and Chain exhibit no overloaded nodes for a homogeneous topology with a median capacity of 50. However, as the level of heterogeneity increases, the percentage increases to around 5% for a median capacity of 25. LEACH-SF achieves with 3.5% the lowest value amongst WSNs, as it penalizes in its cluster head selection scheme nodes with low capacities. LEACH follows a similar pattern, achieving zero overloaded nodes for a median capacity of 50 and reaching 15% for a median capacity of 25, which is three times higher than MST and Chain. For large topologies (i.e., with a smaller capacity than the number of sources), bottom-up and top-down approaches that transmit data directly without intermediate aggregation are not feasible as they always overload the sink. They are thus excluded from this plot.

The linear increase of overloaded nodes with increasing heterogeneity can be explained as follows. When the heterogeneity is low (i.e., high median capacity), all nodes have sufficient resources to perform data aggregation. However, as the number of resource-constrained devices increases, the likelihood of data aggregation occurring on nodes with insufficient capacities also increases. The rate of this increase in the baseline approaches depends on the number of cluster heads. A higher number of cluster heads leads to more intermediate aggregation and, consequently, better load balancing, resulting in a less steep increase in overloaded nodes. Among the baseline approaches, Chain exhibits the smallest incline as it has the highest number of cluster heads ($N-2$). In comparison, MST has $(N-1)/2$ cluster heads, while LEACH has the lowest number of cluster heads ($0.1N$).

In summary, this experiment highlights NEMO's effectiveness in preventing over-provisioning of cluster heads in large topologies and various capacity distributions, outperforming other baselines. This advantage stems from NEMO's resource awareness. Although the optimal approach theoretically could have avoided over-provisioning, it failed to scale to the tested topology size and workload and could not produce any results.

5.3 Placement Quality

In this section, we study the general performance of different aggregation approaches by evaluating the theoretical latencies of their placements in the NCS. Latencies represent the delta with respect to the lower bound, given by top-down/bottom-up. We note that the lower bound and computed latencies are purely theoretical, excluding processing time and estimation errors. We evaluate the impact of estimation errors in Section 5.4 and provide a comprehensive assessment of real end-to-end processing latencies in Section 5.7.

Figure 5 presents the latency deltas of the simulations in a heat map. The y-axis represents the tested topologies and the x-axis the evaluated baselines. In addition to NEMO and NEMO+ for different capacity distributions and weights, we include NEMO with randomly assigned clusters (performing aggregation at the centroids). For $w=1$ and $w=2$, we only display results for median=50, as no significant difference was observed amongst these capacities.

Overall, the results in Figure 5 show that bottom up/top down consistently achieve the lowest theoretical latency in the cost space, as they do not perform any intermediate aggregation and transmit all data directly to the sink. Regarding the aggregation approaches, NEMO+ outperforms all baselines in a comparable setting with $w=1$ as it consistently achieves a latency close to the lower bound for all types of resource distributions, while still maintaining zero overloaded nodes. Especially for PlanetLab, with an absolute deviation of 0.04ms, RIPE Atlas with a deviation 0.2ms, and FIT with 0.07ms, NEMO achieves a latency, which is very close to the theoretical lower bound. For larger weight distributions, it can be observed that the latency increases in NEMO+. This is caused by the fact the overall load of the topology increases, so that NEMO+ needs to perform more aggregations. The same applies also for the median. To show the impact of the pre-processing, we have also added to

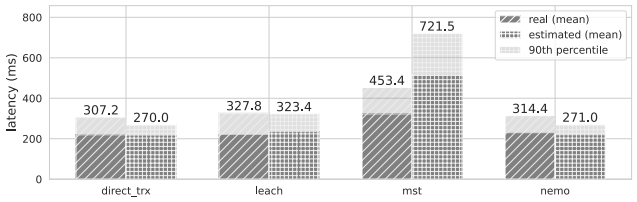


Figure 6: Performance comparison of latencies between real RTT measurements and NCS estimates for RIPE Atlas.

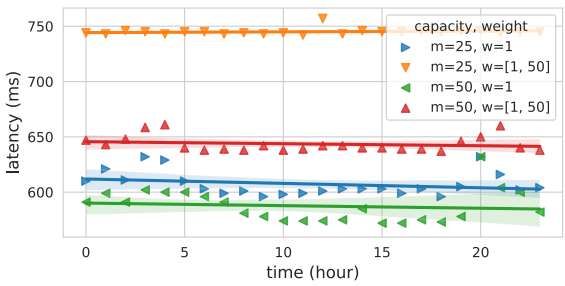


Figure 7: 90th percentile latencies of NEMO for RIPE Atlas over a time span of 24 hours for a single placement.

the evaluation a NEMO variant that assigns nodes randomly to groups instead of using density-based grouping. For this random grouping, the results show significantly higher latencies with up to 7× for FIT than for the density-based counterpart.

Closest to NEMO are LEACH and LEACH-FS, with latencies between 0.89ms (PlanetLab) and 26.12ms (FIT). LEACH-FS is achieving in general lower latencies due to the usage of fuzzy c-means for clustering. The difference of 10ms for King is explained by the fact, that LEACH-FS takes also resources into account for prioritizing a suitable cluster head. However, both LEACH implementations do not take into account whether the cluster head has in total enough resources to handle all of its assigned nodes. NEMO takes these resources into consideration and ensures that the cluster heads do not become overloaded. The worst performance is achieved by MST with latencies of 734*ms* (King) and 147 (sim 1000), and by Chain with more than 20× higher latencies than the compared baselines. These latencies result from the large number of intermediate aggregations, which increase linearly with the size of the topology.

In summary, this experiment demonstrates NEMO as a promising approach for operator placement in geo-distributed stream processing environments. It consistently achieves latency close to the lower bound and outperforms existing baselines that do not take node resources into account. Additionally, we show the importance of the pre-processing phase in NEMO, as the density-based technique proves to be significantly more effective than random grouping of nodes. These findings reveal the potential benefits of using NEMO to optimize operator placement in geo-distributed systems, leading to improved system performance and user experience. In contrast, the aggregation approaches of MST and Chain induce significantly higher latencies than the other approaches, and are thus unsuitable for large-scale stream processing environments.

5.4 Impact of Estimation Errors

NCSs introduce an estimation error as they collect measurements only from a subset of nodes and due to the violation of the triangle inequality. Here, we analyze the impact of this error by comparing the performance of the tested approaches with latency estimates from the NCS against actual latency measurements. We conduct this experiment on a subset of nodes in RIPE Atlas (n=418), where real latency measurements between all nodes are available.

Figure 6 shows that real and estimated mean latencies are nearly equal for direct transmission and grouping-based approaches like LEACH and NEMO. This observation indicates that the estimation error has minimal impact on most nodes in these approaches. Looking at the 90th percentile, LEACH remains largely unaffected by the error with a latency discrepancy of 4.4ms. However, direct transmission and NEMO show noticeable latency discrepancies, deviating by approximately 11% with 37.2ms and 43.4ms, respectively. The most significant deviations are observed in MST and Chain (omitted from the plot due to the large scale of latencies) with a discrepancy of around 40% and 20%, respectively.

In summary, this experiment shows that MST and Chain are unsuitable for placement decisions on NCSs due to their susceptibility to errors introduced by NCSs. This is mainly because they rely on pairwise correct latencies among all nodes. In contrast, LEACH and NEMO require pairwise correct latencies only among cluster heads and between leaf nodes and cluster heads, which makes them more robust against the error induced by TIV.

5.5 Robustness

RTTs in real-world networks are subject to constant variations due to factors such as network congestion, dynamic routing changes, and fluctuations in server loads. Given that many stream processing workloads involve long-running queries, NEMO emphasizes creating placement strategies resilient to these latency fluctuations, aiming to minimize the need for frequent re-optimizations.

To test the robustness of NEMO, we compared latencies of different NEMO placements on real RIPE Atlas RTT measurements over a time span of 24 hours, excluding removals and additions of nodes. Figure 7 shows the 90th percentile latencies of the monitoring query (cf. Section 5.1) with different capacity and weight distributions. In general, latencies exhibit a similar pattern as outlined in Section 5.3, where higher weights correspond to increased latencies due to additional intermediate aggregation levels. The observation over a 24-hour period reveals that latencies fluctuate over time but consistently stay within a standard deviation of approximately 20ms for all placements. This underscores the robustness of NEMO's placements, showcasing resilience to latency fluctuations.

5.6 Scalability

In this section, we evaluate the scalability of NEMO's optimization and re-optimization on topologies of different sizes. The x-axis of Figure 8 shows an increasing number of nodes in the topology, which also linearly increases the total number of operators in the logical plan. The y-axis represents the time taken to compute a full placement of the monitoring query (cf. Section 5.1). We compare NEMO against MST, LEACH, LEACH-SF, Chain, and the optimal

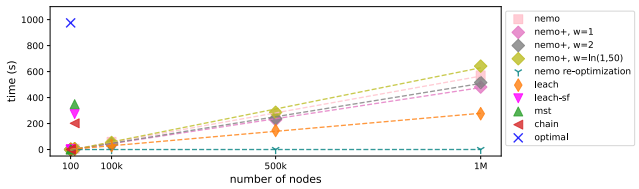


Figure 8: Full-optimization and re-optimization times for the monitoring workload (cf. Section 5.1) that increases its complexity with topology size.

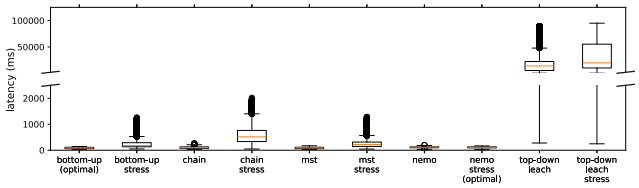


Figure 9: Aggregated latency distributions for all DEBS 2013 workloads grouped by approach.

solution. To evaluate the re-optimization of NEMO, we tested the following scenarios: 1) removal of random leaf nodes, 2) removal of random cluster heads, 3) addition of nodes, 4) computation of coordinates for a node. All re-optimizations produced similar results, each completing in under one second. Therefore, we summarized them together by computing their average.

Our evaluation in Figure 8 demonstrates that NEMO's heuristics (Section 4.1) provide significant advantages in terms of computation time for operator placement and replication, achieving linear-time complexity. For topologies with fewer than 100k nodes, NEMO optimizes the placement in less than a minute, and for topologies of 1 million nodes, placement takes around 10 minutes. In contrast, the optimal approach, which is the only other method avoiding node overloading, requires more than 15 minutes for topologies with less than 100 nodes. In all other experiments, we terminated the runs of the optimal approach manually after 20 minutes. The same applies also to the MST, Chain, and LEACH-SF approaches. The biggest topology processed with MST and LEACH-SF had 10k nodes and took around 6 minutes, while Chain took around 5 minutes for a topology with 1k nodes.

Alongside NEMO, LEACH is the only other approach that achieves linear runtime with respect to the topology size, due to our efficient LEACH implementation using a k-d tree for nearest neighbor search. For topologies up to 100k nodes, LEACH completes optimization in less than 1 minute, which increases to approximately 5 minutes for topologies of 1 million nodes. NEMO takes roughly double the time of LEACH for a full optimization run due to its extra calculations, including optimal operator placement in the NCS and load redistribution to avoid node overloading. Despite this initial computation time, NEMO's advantage lies in its ability to efficiently adapt to dynamic changes. Identifying new cluster heads takes around one second regardless of the topology size. In contrast, LEACH needs to recompute the neighborhood search of all nodes for new cluster heads after the failure of a cluster head.

In summary, our evaluation shows that NEMO solves OPR for an initial deployment in linear time and can adapt to changes in constant time, significantly outperforming the state-of-the-art.

5.7 End-to-end Performance

NEMO's effectiveness as a placement approach extends to cloud-like topologies with workloads that risk overloading processing nodes. Hereby the superiority of NEMO over related work is evident even in a relatively small topology consisting of 11 resource-constrained nodes. We deploy NebulaStream on a local Raspberry PI cluster as described in Section 5.1, utilizing NEMO as the underlying approach for operator placement. We evaluate the end-to-end latency (including both network and processing latency) and the communication cost. We measure communication cost by counting the number of generated buffers that are transmitted through the network. We compare NEMO to the operator placement heuristics in NebulaStream (bottom-up and top-down), MST, and Chain. We note that on this small topology, LEACH and LEACH-SF produce the same placement as top-down. The optimal ILP approach of Cardellini et al. [8] produces the same placement as bottom-up.

Latency. For the monitoring workload in NebulaStream and all DEBS workloads, we observed a consistent pattern. Therefore, Figure 9 shows only the aggregated end-to-end latencies for all DEBS workloads, grouped by approach and experimental setup (with or without stress). In the non-stressed experiment, bottom-up achieves the lowest latencies equal to the optimal approach. Hereby, bottom-up reaches on the aggregated latencies a 90th percentile (NPL) of 129 ms by placing all computations to non-overloaded sources. The NPLs of Chain, MST, and NEMO in the non-stressed experiment are all comparable around 144ms. The similarity in latencies is attributed to the low network latencies of the local topology, which are $\leq$ 1 ms per node pair. Irrespective of the stress condition, top-down consistently exhibits the highest NPL, exceeding 72.5k ms, due to overloading the sink node.

The effectiveness of NEMO's resource-aware placement becomes evident in the second experiment, where the sources are over-utilized. Here, bottom-up experiences a significant increase in latencies with an NPL of 349 ms. In contrast, NEMO remains robust, achieving a similar performance to the non-stressed setup (NPL of 144 ms), significantly outperforming the bottom-up approach. MST achieves an NPL of 416 ms, and Chain an NPL of 988 ms. Despite both MST and Chain performing intermediate aggregations, they do so on overloaded source nodes, resulting in increased overall processing latencies. Chain, in particular, performs more intermediate aggregations on overloaded nodes than MST, which explains its higher latency.

Compared to the simulated results (cf. Section 5.3), the most significant difference is observed for top-down, MST, and Chain. This difference is due to simulations only considering network latency, while end-to-end experiments account for both network and processing latency. Given the low network latencies, the high processing latency in the overloaded sink of the top-down approach dominates the overall latency. As a result, MST and Chain outperform top-down in the end-to-end experiments.

Communication. For the transmitted buffers, we present values for the 1s tumbling window workload on the DEBS workload. Similar results are observed in other workloads. Chain achieves the lowest value with around 300 buffers. This is due to its deep aggregation tree, where data is aggregated at each node, a strategy aligned with its optimization goal of reducing communication rather than latency. MST aggregates data at six intermediate levels and has roughly double the buffers of Chain, totaling around 600. NEMO performs three intermediate aggregations and transmits approximately 900 buffers, three times higher than Chain. In the bottom-up aproach, each source transmits only its partial window aggregate to the sink, resulting in 1800 buffers. The top-down approach has the highest number of transmitted buffers, around 14k, as no data is pre-aggregated before transmission.

5.8 Summary

Our evaluation indicates that the most effective way to minimize latencies in distributed stream processing is by placing all DAFs at the sources, as done in the bottom-up approach. However, this requires sources with sufficient processing resources and results in a significant increase in processing latencies if operators are placed on overloaded nodes. Our experiments demonstrate that NEMO effectively addresses this issue by avoiding operator placement on overloaded nodes, resulting in significantly lower processing latencies compared to all baselines when source nodes bear additional load. Furthermore, NEMO scales to extremely large topologies with over 1 million nodes. LEACH is the only other approach that achieves such scalability. However, it fails to prevent node overloading, significantly impacting processing latencies in real end-to-end deployments by more than 500 times. For communication reduction, Chain yields the best results due to its numerous intermediate aggregations. However, it cannot scale to large topologies and incurs higher latencies than bottom-up and NEMO.

6 RELATED WORK

OP has been studied extensively in the literature under different modeling assumptions and optimization goals, such as minimizing application end-to-end latency or inter-node traffic migration, mainly focusing on homogeneous cloud computing clusters [22, 60]. To address the heterogeneity of devices, Cardellini et al. [7] propose a general formulation of optimal operator placement that considers computing and networking resources. A joint placement and replication approach is proposed by Cardellini et al. [8] to determine the optimal number of replicas for each operator during initial placement on a given infrastructure. Unlike NEMO, both approaches are impractical for osmotic computing applications due to their excessively long run times on large topologies and their inability to address changes through efficient re-optimizations.

Pietzuch et al. [49] propose an approach called SBON to solve the SOP efficiently using a cost space. Rizou et al. [51] extend the SBON approach to solve MOP using a cost space. They prove that MOP can be reduced to multiple SOPs, which are efficiently calculated using gradient descent. Their approach is resource-agnostic, as it does not have the notion of resources/capacities in its model, and does not address re-optimizations. In contrast, our approach solves OPR efficiently, which is a generalization of MOP. Additionally, it supports efficient re-optimizations and is resource-aware. An approach to handle the placement of newly added operators to an existing deployment has been examined by Heinze et al. [28]. They propose a model to predict latency spikes created by operator movements and use it to develop an operator placement algorithm based on a bin-packing heuristic. However, in contrast to NEMO, this algorithm minimizes latency violations and focuses solely on placing newly added operators.

The computation of DAFs on resource-constrained devices is mainly addressed by WSNs using three types of aggregation approaches. 1) Cluster-based approaches transmit data to a cluster head, which aggregates data from all nodes in its cluster and sends a concise digest to the sink. The most common approach is LEACH [29], which uses a randomized protocol with rotating cluster heads to prolong network lifetime. HEED [66] focuses on energy-efficient clustering using a probabilistic approach to elect cluster heads. CLUDDA [10] is a distributed protocol that relies on local decisions made by cluster heads to form clusters and optimize communication. 2) Tree-based approaches achieve further improvements in energy efficiency by transmitting only to close neighbors. Common approaches like EADAT [21] or PEDAP [59] use the greedy MST algorithm of Prim to create an aggregation tree. 3) Chain-based approaches like PEGASIS [35] achieve the highest reduction in energy utilization by organizing nodes into a linear chain for data aggregation. This is either achieved using a greedy algorithm, or the sink can determine the chain in a centralized manner. Different from NEMO, WSN approaches focus on minimizing energy utilization and do not consider latency and load. Additionally, WSN approaches represent network communication protocols for in-network data aggregation, while NEMO addresses operator placement and replication of DAFs for a holistic SPE.

7 CONCLUSION

This paper introduces NEMO, a heuristic approach for determining the replication factor and placement of DAFs in osmostic computing environments. NEMO prioritizes minimizing latency and preventing node overloading as its main optimization objectives. Furthermore, NEMO can handle topological changes by efficiently recomputing existing placement solutions. NEMO projects a given topology to a Euclidean space in a NCS. This facilitates calculating the optimal location of operators, determining the degree and placement of replicas, and distributing replicas among underutilized nodes near their non-replicated counterparts. Our experiments demonstrate that NEMO outperforms comparable state-of-the-art approaches in terms of scalability by minimizing end-to-end latency and avoiding node overloading. Specifically, our results demonstrate that NEMO scales DAF operator computation to topologies with millions of nodes and significantly reduces network communication. This establishes NEMO as a foundation for placing DAFs in osmotic computing environments to efficiently process continuous data streams on large, heterogeneous, and geo-distributed topologies.

ACKNOWLEDGMENTS

This work was funded by the German Ministry for Education and Research as BIFOLD – Berlin Institute for the Foundations of Learning and Data (ref. 01IS18025A and ref. 01IS18037A).

REFERENCES

[1] 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.

[2] Shadi Al-Sarawi, Mohammed Anbar, Kamal Alieyan, and Mahmood Alzubaidi. 2017. Internet of Things (IoT) communication protocols. In *8th International Conference on Information Technology, ICIT*. 685–690.

[3] Leonardo Aniello, Roberto Baldoni, and Leonardo Querzoni. 2013. Adaptive online scheduling in storm. In *The 7th ACM International Conference on Distributed Event-Based Systems, DEBS*. 207–218.

[4] Tanapat Anusas-Amornkul and Sirasit Sangrat. 2017. Linux Server Monitoring and Self-healing System Using Nagios. In *Mobile Web and Intelligent Information Systems - 14th International Conference, MobiWIS*, Vol. 10486. 290–302.

[5] Lawrence Benson, Philipp M. Grulich, Steffen Zeuch, Volker Markl, and Tilmann Rabl. 2020. Disco: Efficient Distributed Window Aggregation. In *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT*. 423–426.

[6] Robert L. Cannon, Jitendra V. Dave, and James C. Bezdek. 1986. Efficient Implementation of the Fuzzy c-Means Clustering Algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.* 8, 2 (1986), 248–255.

[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] Ramaswamy Chandrasekaran and Arie Tamir. 1990. Algebraic Optimization: The Fermat-Weber Location Problem. *Math. Program.* 46 (1990), 219–224.

[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, 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.

[12] Ankit Chaudhary, Steffen Zeuch, and Volker Markl. 2020. Governor: Operator Placement for a Unified Fog-Cloud Environment. In *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT*. 631–634.

[13] Ankit Chaudhary, Steffen Zeuch, Volker Markl, and Jeyhun Karimov. 2023. Incremental Stream Query Merging. In *Proceedings 26th International Conference on Extending Database Technology, EDBT*. 604–617.

[14] 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.

[15] Ruosi Cheng and Yu Wang. 2018. A Survey on Network Coordinate Systems. In *MATEC Web of Conferences*.

[16] 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.

[17] Manuel Costa, Miguel Castro, Antony I. T. Rowstron, and Peter B. Key. 2004. PIC: Practical Internet Coordinates for Distance Estimation. In *24th International Conference on Distributed Computing Systems, ICDCS*. 178–187.

[18] James A. Cowling, Dan R. K. Ports, Barbara Liskov, Raluca Ada Popa, and Abhijeet Gaikwad. 2009. Census: Location-Aware Membership Management for Large-Scale Distributed Systems. In *USENIX Annual Technical Conference*.

[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] Marcos Dias de Assunção, Alexandre Da Silva Veith, and Rajkumar Buyya. 2018. Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. *J. Netw. Comput. Appl.* 103 (2018), 1–17.

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

[22] Raphael Eidenbenz and Thomas Locher. 2016. Task allocation for distributed stream processing. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM*. 1–9.

[23] 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.

[24] Stefano Forti, Marco Gaglianese, and Antonio Brogi. 2021. Lightweight self-organising distributed monitoring of Fog infrastructures. *Future Gener. Comput. Syst.* 114 (2021), 605–618.

[25] Thomas M. J. Fruchterman and Edward M. Reingold. 1991. Graph Drawing by Force-directed Placement. *Softw. Pract. Exp.* 21, 11 (1991), 1129–1164.

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

[27] 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.

[28] 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.

[29] 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.

[30] Jeong-Hyon Hwang, Ugur Çetintemel, and Stanley B. Zdonik. 2008. Fast and Highly-Available Stream Processing over Wide Area Networks. In *Proceedings of the 24th International Conference on Data Engineering, ICDE*. 804–813.

[31] Richard M. Karp. 2010. Reducibility Among Combinatorial Problems. In *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*. Springer, 219–241.

[32] Geetika T. Lakshmanan, Ying Li, and Robert E. Strom. 2008. Placement Strategies for Internet-Scale Data Stream Systems. *IEEE Internet Comput.* 12, 6 (2008), 50–60.

[33] 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.

[34] Yongjun Liao, Wei Du, Pierre Geurts, and Guy Leduc. 2013. DMFSGD: A Decentralized Matrix Factorization Algorithm for Network Distance Prediction. *IEEE/ACM Trans. Netw.* 21, 5 (2013), 1511–1524.

[35] 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.

[36] Stuart P. Lloyd. 1982. Least squares quantization in PCM. *IEEE Trans. Inf. Theory* 28, 2 (1982), 129–136.

[37] Björn Lohrmann, Peter Janacik, and Odej Kao. 2015. Elastic Stream Processing with Latency Guarantees. In *35th IEEE International Conference on Distributed Computing Systems, ICDCS*. 399–410.

[38] Cristian Lumezanu, Randolph Baden, Dave Levin, Neil Spring, and Bobby Bhattacharjee. 2009. Symbiotic Relationships in Internet Routing Overlays. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI*. 467–480.

[39] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. 2005. TinyDB: An acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.* 30, 1 (2005), 122–173.

[40] Kasper Grud Skat Madsen, Yongluan Zhou, and Jianneng Cao. 2017. Integrative Dynamic Reconfiguration in a Parallel Stream Processing Engine. In *33rd IEEE International Conference on Data Engineering, ICDE*. 227–230.

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

[42] Yun Mao, Lawrence K. Saul, and Jonathan M. Smith. 2006. IDES: An Internet Distance Estimation Service for Large Networks. *IEEE J. Sel. Areas Commun.* 24, 12 (2006), 2273–2284.

[43] Matthew L. Massie, Brent N. Chun, and David E. Culler. 2004. The Ganglia distributed monitoring system: Design, Implementation, and Experience. *Parallel Comput.* 30, 5-6 (2004), 817–840.

[44] Christopher Mutschler, Holger Ziekow, and Zbigniew Jerzak. 2013. The DEBS 2013 grand challenge. In *The 7th ACM International Conference on Distributed Event-Based Systems, DEBS*. 289–294.

[45] T. S. Eugene Ng and Hui Zhang. 2002. Predicting Internet Network Distance with Coordinates-Based Approaches. In *The 21st Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM*. 170–179.

[46] T. S. Eugene Ng and Hui Zhang. 2004. A Network Positioning System for the Internet. In *Proceedings of the General Track: USENIX Annual Technical Conference*. 141–154.

[47] Xia Pan, Xia Zhang, Hongyi Yu, and Chao Zhang. 2009. Study on routing protocol for WSNs based on the improved Prim algorithm. In *2009 International Conference on Wireless Communications & Signal Processing*. 1–4.

[48] 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.

[49] 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.

[50] Sean C. Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. 2004. Handling Churn in a DHT (Awarded Best Paper!). In *Proceedings of the General Track: USENIX Annual Technical Conference*. 127–140.

[51] 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.

[52] Peter J. Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* 20 (1987), 53–65.

[53] Atul Sandur, ChanHo Park, Stavros Volos, Gul Agha, and Myeongjae Jeon. 2022. Jarvis: Large-scale Server Monitoring with Adaptive Near-data Processing. In *38th IEEE International Conference on Data Engineering, ICDE*. 1408–1422.

[54] Zhitao Shen, Vikram Kumaran, Michael J. Franklin, Sailesh Krishnamurthy, Amit Bhat, Madhu Kumar, Robert Lerche, and Kim Macpherson. 2015. CSA: Streaming Engine for Internet of Things. *IEEE Data Eng. Bull.* 38, 4 (2015), 39–50.

[55] Mohammad Shokouhifar and Ali Jalali. 2017. Optimized sugeno fuzzy clustering algorithm for wireless sensor networks. *Eng. Appl. Artif. Intell.* 60 (2017), 16–25.

[56] RIPE NCC Staff. 2015. Ripe Atlas: A global internet measurement network. *Internet Protocol Journal* 18, 3 (2015), 2–26.

[57] Moritz Steiner and Ernst W. Biersack. 2009. Where Is My Peer? Evaluation of the Vivaldi Network Coordinate System in Azureus. In *8th International Networking Conference, IFIP-TC*. 145–156.

[58] Nitin Sukhija and Elizabeth Bautista. 2019. Towards a Framework for Monitoring and Analyzing High Performance Computing Environments Using Kubernetes and Prometheus. In *IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation*. 257–262.

[59] Hüseyin Özgür Tan and Ibrahim Korpeoglu. 2003. Power efficient data gathering and aggregation in wireless sensor networks. *SIGMOD Rec.* 32, 4 (2003), 66–71.

[60] Cory Thoma, Alexandros Labrinidis, and Adam J. Lee. 2014. Automated operator placement in distributed Data Stream Management Systems subject to user constraints. In *Workshops Proceedings of the 30th International Conference on Data Engineering Workshops, ICDE*. 310–316.

[61] 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.

[62] Shivaram Venkataraman, Aurojit Panda, Kay Ousterhout, Michael Armbrust, Ali Ghodsi, Michael J. Franklin, Benjamin Recht, and Ion Stoica. 2017. Drizzle: Fast and Adaptable Stream Processing at Scale. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 374–389.

[63] 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.

[64] Guohui Wang, Bo Zhang, and T. S. Eugene Ng. 2007. Towards network triangle inequality violation aware distributed systems. In *Proceedings of the 7th ACM SIGCOMM Internet Measurement Conference, IMC*. 175–188.

[65] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. 2008. Wireless sensor network survey. *Computer networks* 52, 12 (2008), 2292–2330.

[66] 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.

[67] Wang Yue, Lawrence Benson, and Tilmann Rabl. 2023. Desis: Efficient Window Aggregation in Decentralized Networks. In *Proceedings 26th International Conference on Extending Database Technology, EDBT*. 618–631.

[68] 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*.

[69] 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.