SNR: Network-aware Geo-Distributed Stream Analytics

Habib Mostafaei
TU Berlin

Shafi Afridi
TU Berlin

Jemal H. Abawajy
Deakin University

***Abstract*—Emerging applications such as those running on the Internet of Things (IoT) devices produce constant data streams that need to be processed in real-time. Distributed stream processing systems (DSPs), with geographically distributed cluster networks interconnected via wide area network (WAN) links, have recently gained interest in handling these applications. However, these applications have stringent requirements such as low-latency and high bandwidth that must be guaranteed to ensure the quality of service (QoS). These application requirements raise fundamental DSPs resource management and scheduling challenge. In this paper, we formulate the problem of placement of worker nodes on a geo-distributed DSPs cluster network as a multi-criteria decision-making problem and propose an additive weighting-based approach to solve it. The proposed solution finds the trade-off among different network parameters and allows executing the tasks according to the desired performance metrics. We evaluated the proposed approach using the Yahoo! streaming benchmark on a testbed and compare it against mechanisms deployed in Apache Spark, Apache Storm, and Apache Flink. The results of the evaluation show that our approach improves the performance of Spark up to 2.2x-7.2x, Storm up to 1.2x-3.4x, and Flink up to 1.4x-3.3x compared to other approaches, which makes our approach useful for use in practical environments.**

***Index Terms*—IoT, worker node placement, Geo-distributed analytics, Stream processing, Simple Additive Weighting**

I. Introduction

Emerging applications running on the Internet of Things (IoT) devices, social networks, or user-clicks on websites typically generate a massive amount of continuous stream of raw data from distributed geographical locations. Organizations need to analyze and mine these data to obtain insights and valuable intelligence such as trend detection in social networks in real-time. These applications normally have stringent requirements ranging from low-latency to high bandwidth. Therefore, processing this massive amount of streaming data in a single location within a limited timeframe is not practically possible. As a result, distributed stream processing systems (DSPs) composed of geographically distributed (geo-distributed) networks that communicate via WAN have recently become a popular choice to run these applications [1]. The main idea of geo-distributed stream data processing systems is to push the computations to the edge of the network close to the sources of the streaming data. This approach provides several benefits that include privacy preservation and cost-saving due to the minimization of data transfer overhead.

Although geo-distributed cluster networks have the capacity to handle stream data processing, stringent QoS requirements of the stream data processing applications raise fundamental resource management and scheduling challenge. Currently, various systems such as Apache Spark [2], Apache Storm [3], and Apache Flink [4] are used for processing streaming data. However, these systems are designed to process the queries at a single location on a cluster rather than on geo-distributed cluster networks. There are several attempts to address the stream data processing in geo-distributed cluster networks [1], [5], [6]. The techniques in such solutions are mostly applied for batch scenarios in which the input data is available prior to query execution. Although the approaches proposed in existing works [7]–[10] do not assume that the data size and the rate are not known in advance, these solutions focus on different aspects such as links delay, bandwidth, and cost of running a task in different datacenters. For example, the work in [11] claims that the link cost should have higher priority than the delay and bandwidth in a multi datacenters environment. The reason for such a claim is due to the amount of data transferred using high-cost links. Nevertheless, none of these solutions offer the flexibility in prioritizing various network parameters in placing the workers on a geo-distributed cluster.

In this paper, we address the placement of worker nodes on geo-distributed cluster networks. Specifically, we attempt to answer the following questions: (i) How can we prioritize network metrics to efficiently run a task on geo-distributed cluster networks?; and (ii) What is the trade-off among different network parameters in placing worker nodes in a multi-cloud environment? To answer the above questions, we first formulate the problem of placement of worker nodes on geo-distributed cluster network as a multi-criteria decision-making problem and use the Simple Additive Weighting (SAW) [12] method to solve it. We call the proposed approach a SAW-based Node Ranking (SNR) algorithm. The main goal of SNR is to find the best placement of the worker nodes on geo-distributed cluster networks by considering multiple network criteria. We check the different impacts of SNR on the placing nodes, and the obtained results show that the internode delays of our algorithm on average are 2.4x less than the current default algorithm. To perform the real-world experiments, we use the Yahoo! streaming benchmark [13] on a cluster of 11 VMs running Apache Spark, Apache Storm, and Apache Flink. Our results show the significant performance improvement of SNR compared to the default placement approach.

The contribution of our work can be summarized as follows:

- We develop SNR (worker node placement on geo-

distributed cluster network) that considers the most significant network relevant parameters when placing the workers in a geo-distributed cluster network;
- We study the trade-off among the relevant parameters of worker selection;
- We perform real-world experiments to evaluate the performance of SNR on a set of custom and all networks taken from TopologyZoo [14]. The obtained results show that SNR improves the execution latency of Spark up to 2.2x-7.2x, Storm up to 1.2x-3.4x, and Flink up to 1.4x-3.3x compared to the default placement approach.

The rest of the paper is organized as follows. Section II states the system model and problem formulation. The detail of the SNR algorithm comes in Section III. Section IV details the evaluation of the SNR and reports the obtained results. The related works come in Section V and Section VI concludes the paper.

II. System Model and Problem Statement

A. System Model

We consider a scenario in which the compute slots of the cluster are geographically distributed around the globe. There is a global manager that is in charge of running the tasks across different locations depending on the enterprise requirements. The compute slots are connected through WAN links that have different features such as bandwidth, delay, and cost. For example, the uplink and downlink bandwidths of a link can be different because different applications share the same links [5]. The global manager can query the network to obtain the information of the underlying infrastructure. Fig. 1 shows an example scenario with 5 datacenters (DCs). Each datacenter has its own compute slots, and the input data comes as a stream from different resources depending on the application scenarios. The data should be processed close to the source for several reasons such as saving the bandwidth and cost or privacy of data. The global manager has the view of the available resources and decides where should a task runs.

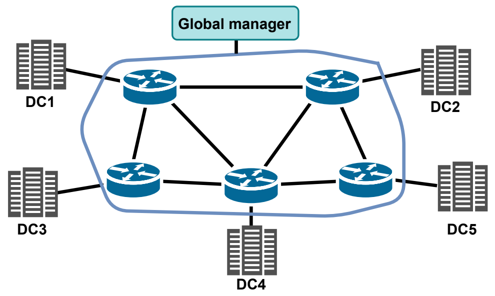


Fig. 1. A geo-distributed stream-processing system spanning over 5 datacenters (DCs).

B. Problem Statement

The main objective is to minimize the execution latency and cost of streaming tasks while obeying the bandwidth constraints in placing the workers of a cluster in a geo-distributed environment. The streaming tasks run for an infinite time. We consider the network relevant parameters such as bandwidth, delay, and cost when placing the tasks for execution.

We consider a network topology of $G = (V, E)$ in which V is the set of worker nodes, and E is the set of edges (links) connecting the worker nodes. Each worker node[1] has a set of available task slots to execute tasks. We assume that the workers already exist and ready for task execution. Each link in G has three different parameters, namely, bandwidth, delay, and cost. The main goal is to enable the users to select a set of worker nodes in the network according to the given priority to the network-related parameters[2].

Delay. The internode delays in a geo-distributed cluster are different from those in a single cluster. The delay can vary from 10s to 100s of milliseconds depending on the locations of the compute slots [15]. Therefore, link delay plays a determinant role in task execution, especially for delay-sensitive applications. For such kinds of applications, every millisecond of latency can have a huge economical impact on enterprises. For example, Akamai in 2017 reported that every 100 milliseconds of delay have a determinant impact in dropping the customers of online businesses [16]. Therefore, one of our goals is to select a subset of worker nodes in such a way that the sum of internode delays is minimized. Mathematically,

$$\begin{aligned} \text{minimize} \quad & \sum_{x=1}^{n} T_x^y \qquad (1) \\ \text{subject to} \quad & T_x^y > 0, \quad \forall (x,y) \in G' \quad \text{and} \quad x \neq y, \end{aligned}$$

where T_x^y is the link delay in milliseconds (ms) crossing from node x to y, and n is the number of chosen worker nodes from graph G. We assume that G' is the sub-graph of the chosen nodes in graph G.

Cost. We model the cost as follows. Let C_x^y be the data transmission cost from worker x to y through the link among them. The total transmission cost can be computed as follows.

$$\begin{aligned} \text{minimize} \quad & \sum_{x=1}^{n} C_x^y \qquad (2) \\ \text{subject to} \quad & C_x^y > 0, \quad \forall (x,y) \in G' \quad \text{and} \quad x \neq y. \end{aligned}$$

We consider the cost as the cost of steering 1GB of data traffic over a link in USD ($). The goal is to minimize the sum of the cost of G'.

Bandwidth. We model the network traffic as follows. Let B_x^y be the available bandwidth of a link in Mbps from worker x and y. The total generated traffic on link $(e_x, e_y) \in E'$ can be computed as follows.

$$\sum B_x^y, \quad x \neq y, \qquad (3)$$

We assume that multiple tasks can be executed on the available task slots on a worker. Therefore, each task generates data

[1]We assume the nodes in the graph are placed in different physical locations.
[2]We use the available task slots on a VM as the same meaning of the worker nodes to avoid any confusion.

traffic and consumes a portion of the available bandwidth between nodes x and y. The goal is to maximize the minimum available bandwidth among the selected nodes in G'. Consider a scenario in which we have several worker nodes distributed on a geo-distributed network. In this case, there are multiple hops between two nodes in the graph with diverse parameters. Therefore, the link with the lowest bandwidth determines the amount of traffic that can be sent through that path.

III. SNR Algorithm

In this section, we state how SNR selects the task slots by considering multiple criteria. We also explain a running example of SNR in placing worker nodes in the cluster network.

A. SNR algorithm

WE use different criteria in the node selection step of the SNR algorithm. This problem is known as the multi-objective problem, and we use the Simple Additive Weighting (SAW) method to transform the problem into a single objective one [17]. The SAW method is a simple and popular technique to make a decision on a set of attributes for each alternative. We explain the detail of SNR as follows.

Algorithm 1: The SNR algorithm

```
input : network graph $G$, links bandwidth, delay, cost , number
        of worker nodes , priority of each parameter
output: a set of nodes $W$
1  $W = \emptyset$                /* The set of worker nodes */
2  for each $x \in G$ do
3      N=G.getNeighbors(x)
4      $L_{sum}$ = 0, $BW_{sum}$ = 0, $C_{sum}$ = 0
5      for $y \in N$ do
6          $BW_{sum}$ += $B_x^y$
7          $L_{sum}$ += $T_x^y$
8          $C_{sum}$ += $C_x^y$
9      end
10     normalize the parameters of each node using Eq. 4
11 end
12 x = firstNode($G$)
13 $W = W \bigcup x$
14 while $|W| < t$ do
       /* We look for the neighbors of x to add
          the next node to $W$ */
15     for $y \in x.Neighbors$ do
16         find maximum value of each parameter
17     end
18     for $y \in N$ do
19         apply Eq. 6
20     end
21     find rank using Eq. 5
22     $\beta$=get the neighbor with the highest rank
23     $W = W \bigcup \beta$
24     $x = \beta$
25 end
```

Let E be a set of links and m be the set of decision criteria for each link in G. Here, we consider the available bandwidth, delay, and cost as the decision criteria, i.e., $|m| = 3$. We prefer the highest value for the available bandwidth. While for the delay, and cost the lowest values are better. Let assume that w_k is the weight of importance for each criterion.

Algorithm 1 presents the pseudo-code of the SNR algorithm. In this algorithm, we first normalize the network-relevant parameters of each connected link to a node in the graph. Then, we compute the rank of each node to select a node in each step of the algorithm. The algorithm is general-purpose and can be used to select a subset of task slots available on the worker nodes.

First node selection. The SNR algorithm starts by selecting the first node on a graph. Therefore, we first normalize the criteria based on the network-related parameters of neighbors of each node using Eq. 4. To do so, we take the average of the available bandwidth of all connected links to a node and divide it over the maximum available bandwidth. Similar to the bandwidth, SNR computes the average link delays and costs. Then, it divides the averaged values into their minimum values. This procedure executes once when SNR selects the first node. We normalize the values of attributes of each node x in graph G as follows.

$$\psi_x = \begin{cases} \dfrac{\frac{\sum_{e=1}^{n} e_{xy}}{n}}{max(e_{xy})} & \text{, bandwidth} \\ \dfrac{min(e_{xy})}{\frac{\sum_{e=1}^{n} e_{xy}}{n}} & \text{, delay and cost,} \end{cases} \tag{4}$$

where n is the number of neighbors of a node and e_{xy} is the edge from node x to y. This equation returns the normalized value ψ_x for the entire nodes of the graph. Now, we assign the weight for each criterion of a link connected to node x according to the user needs. Mathematically,

$$R_{node} = \sum \psi_x w_k, \quad \text{for} \quad x = 1, \ldots, n \quad \text{and} \quad k = 1, 2, 3, \tag{5}$$

where R_{node} is the rank of each node in the graph G, n is the number of neighbors of node x, and w_k is the priority weight of each attribute, i.e., bandwidth, latency, and cost. Each w_k has a value in the range (0,1), and the sum of them is equal to 1. We select the node with the highest rank as the first node. We colocate the master and the first worker node of the cluster on the first node. We use the master node to submit the query to a set of workers/clients. Algorithm 2 shows the pseudo-code of the first node selection of the SNR approach. **Node selection.** After selecting the first node, we use the neighbor nodes of this node to add the next node to our subset of worker nodes. To do so, we normalize the available bandwidth, delay, and cost of the links that are connected to the current node using Eq. 6. Then, we apply Eq. 5 by replacing the ψ_x with π_x to rank the neighbors of the current node. The neighbor with the highest rank will be selected as the next node, and this procedure continues until the desired number of worker

Algorithm 2: Find the first node

```
input : Network graph $G$ , bandwidth, delay, and cost of each
        link
output: A node with highest rank ($x_{best}$)
Function firstNode($G$):
    tmp=$\emptyset$
    for each $x \in G$ do
        for $y \in N$ do
            $BW_{max}$ = maxBandwidth(x,y)
            $L_{min}$ = minLatency(x,y)
            $C_{min}$ = minCost(x,y)
        end
        for $y \in N$ do
            $E_{BW} = w_1 \cdot \frac{B_x^y}{BW_{max}}$
            $E_L = w_2 \cdot \frac{L_{min}}{T_x^y}$
            $E_C = w_3 \cdot \frac{C_{min}}{C_x^y}$
        end
        find rank using Eq. 5
        tmp= tmp $\bigcup$ x
    end
    $x_{best} = \varnothing$
    $R_{tmp} = 0$
    for $x \in tmp$ do
        if $R_x > R_{tmp}$ then
            $R_{tmp} = R_x$
            $x_{best} = x$
        end
    end
    return $x_{best}$
End Function
```

nodes has been chosen.

$$\pi_x = \begin{cases} \dfrac{\sum_{e=1}^{n} e_{xy}}{max(e_{xy})} & \text{, bandwidth} \\ \dfrac{min(e_{xy})}{\sum_{e=1}^{n} e_{xy}} & \text{, delay and cost.} \end{cases} \tag{6}$$

Note thta we remove the selected node from the available nodes list while using Eq. 5. This mechanism speeds up the node selection step of the SNR algorithm due to the reduction in the search space.

SNR orchestrator. We develop an orchestrator to integrate SNR with the current DSPs, i.e., Storm, Spark, and Flink. It executes the given streaming tasks on the chosen worker nodes. To do so, it gets an input file including the directory of each DSP on all worker nodes, the number of worker nodes, and the node selection mechanism. The orchestrator uses the daemons provided by each DSP to start either the master or the worker nodes. Then, it submits the received streaming tasks to the chosen worker nodes for execution.

B. Running Example for SNR

In this section, we explain our SNR method in selecting the worker nodes by using an example. Fig. 2 depicts a network graph of 6 nodes. Each edge in this graph has three network-related parameters, namely, the link bandwidth in Mbps, the link latency in milliseconds, and the link cost as the cost of transferring 1GB of data over that link in USD($). We consider the Round-Trip Time (RTT) delay as the link delay in this graph.

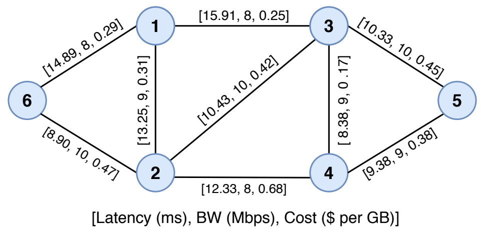


Fig. 2. An example graph with six nodes.

First node selection. To select the first node from the graph in SNR, we compute the normalized value of all nodes in the graph. In this example, we have 6 nodes, and Table I indicates the normalized value for the bandwidth, latency, and cost of each edge connected to that node. We compute these values using the same parameters on each edge in Eq. 4. SNR computes the weight of each edge in the graph using the normalized values. We apply Eq. 5 and select the maximum value as the highest rank. SNR selects the corresponding node to that highest rank link as the first node. According to the example graph in Fig. 2, the SNR selects node number 3 as the first node.

TABLE I
THE NORMALIZED VALUES FOR EACH NODE AND THEIR CORRESPONDING RANK IN FIRST NODE SELECTION STEP OF SNR.

Node	Latency	BW	Cost	R_x
Node 1	0.90	1.00	0.68	2.57
Node 2	1.00	0.60	0.88	2.48
Node 3	1.00	0.88	0.87	**2.75**
Node 4	0.90	0.69	0.98	2.57
Node 5	0.97	0.68	1.00	2.65
Node 6	0.97	0.75	0.83	2.55

Node selection. We assume that our goal is to select three nodes for the cluster and all three networking parameters have the same priority for the placement. After choosing the first node, we apply a similar mechanism for the remaining nodes in the graph until we reach the desired number of nodes for the cluster. According to our example, SNR selects node number 3 as the first node.

Table II shows the rank of each node in the example graph. Now, we check the neighbors of this node and their rank to choose the next node. Among the neighbors of node number 3, node number 4 has the highest rank value, and we add this node as the next node to our list of selected nodes. We need to pick another node according to our needs, and we check the neighbors of node number 4 to choose the next node because this is the last selected node in the cluster. Among the neighbors of this node, SNR picks node number 5. Therefore, the final selected nodes are 3, 4, and 5.

Tradeoff among the parameters. We now show the tradeoff among different parameters. Table III shows the impact of

TABLE II
THE NORMALIZED VALUES FOR EACH NODE WITH THEIR NEIGHBORS RANK IN SNR.

Node	Normalized values BW	Latency	Cost	R_x	To
Node 1	1.00	1.00	0.81	2.81	Node 2
	0.89	0.83	1.00	2.72	Node 3
	0.89	0.89	0.86	2.64	Node 6
Node 2	0.90	0.67	1.00	2.57	Node 1
	1.00	0.85	0.74	2.59	Node 3
	0.80	0.72	0.46	1.98	Node 4
	1.00	1.00	0.66	2.66	Node 6
Node 3	0.80	0.53	0.68	2.01	Node 1
	1.00	0.80	0.40	2.21	Node 2
	0.90	1.00	1.00	**2.90**	**Node 4**
	1.00	0.81	0.38	2.19	Node 5
Node 4	0.89	0.68	0.25	1.82	Node 2
	1.00	1.00	1.00	3.00	Node 3
	1.00	0.89	0.45	**2.34**	**Node 5**
Node 5	1.00	1.00	1.00	3.00	Node 3
	0.89	0.52	0.59	1.99	Node 4
Node 6	0.80	0.60	1.00	2.40	Node 1
	1.00	1.00	0.62	2.62	Node 2

TABLE III
THE TRADE-OFF AMONG THE METRICS IN EXAMPLE GRAPH OF SNR.

Prioritized metric	Nodes	BW	Latency	Cost
Bandwidth	{3,2,6}	10,10,10	10.43,8.9,19.33	0.42,.47,0.89
Latency	{5,4,3}	8,10,9	9.38,10.33,8.38	0.38,0.45,0.17
Cost	{1,3,4}	8,9,8	15.91,8.38,24.29	0.25,0.17,0.42

giving a higher priority to the bandwidth, delay, and cost. The nodes column in this table shows the selected nodes by SNR for the corresponding prioritized metric. When we give the highest priority to the bandwidth, SNR picks the nodes with the highest available bandwidth. For example, by prioritizing the bandwidth, SNR selects nodes 3, 2, and 6. There are 3 links among these nodes, and thus, we have three values for each row in Table III that shows the chosen parameter value for that link. While for the latency and cost, SNR selects the nodes with the lowest latency and cost. The corresponding values for each prioritized parameter are highlighted in red in Table III. The results show that the priority of each network parameter dictates the selection of the nodes. The obtained results confirm that SNR selects the worker nodes based on the applications' demand.

IV. EVALUATIONS

In this section, we first study the tradeoff among different network-related parameters of the worker node placement by assigning priorities to bandwidth, latency, and cost of WAN links. Then, we measure the performance of the SNR algorithm on three custom and all topologies of TopologyZoo [14]. The goal is to assess the impact of placement on various network-related performance metrics such as average delay among the nodes and the number of hops. Finally, we measure the impact of placement on the execution latency of streaming queries using Yahoo! streaming benchmark [18].

TABLE IV
CUSTOM GRAPH PROPERTIES.

Graph	Nodes	Links	BW [Mbps]	Delay [ms]	Cost [$ per GB]
Small	20	35	[7, 14]	[1, 25]	[0.02, 0.25]
Medium	30	65	[7, 14]	[26, 75]	[0.02, 0.25]
Large	50	140	[7, 14]	[76, 125]	[0.02, 0.25]

We use the real networks' delay such as the ones in [15], [19], cost [20]–[22], and bandwidth [23] among the worker nodes and set the locations of worker nodes using the real-world datacenter locations [24]–[26]. We create three custom random graphs, namely small, medium, and large, to simulate diverse networks in terms of size and other parameters. Table IV presents the corresponding network parameters with their values in each graph. The link delay information for TopologyZoo networks is obtained using the coordination information.

A. *Illustrating Tradeoff*

We check the tradeoff among different network-relevant parameters by assigning different priorities among them for scenarios with 8 worker nodes. In this experiment, we first give the highest priority to the available bandwidth while keeping the priority among the delay and cost fixed and the same in Eq. 5. Then, we do the same measurements for the delay and cost in all three custom topologies. The values for the bandwidth are in [Mbps], latency in [ms], and cost [$ per GB].

TABLE V
THE TRADE-OFF AMONG THE METRICS IN SMALL TOPOLOGY.

Prioritized metric	min(BW)	$\overline{\text{Latency}}$	$\sum$Cost
Bandwidth	9.4	34.29	17.12
Latency	7.3	20.55	22.58
Cost	7.3	26.29	15.47

TABLE VI
THE TRADE-OFF AMONG THE METRICS IN MEDIUM TOPOLOGY.

Prioritized metric	min(BW)	$\overline{\text{Latency}}$	$\sum$Cost
Bandwidth	10.1	107.35	16.44
Latency	7.5	92.18	24.03
Cost	8.3	96.16	11.28

TABLE VII
THE TRADE-OFF AMONG THE METRICS IN LARGE TOPOLOGY.

Prioritized metric	min(BW)	$\overline{\text{Latency}}$	$\sum$Cost
Bandwidth	8.7	223.89	16.98
Latency	7.0	132.95	11.53
Cost	7.2	155.95	10.4

Tables V, VI, and VII reports the tradeoff among bandwidth, delay, and cost of running SNR on small, medium, and large topologies for the links among the chosen nodes. In these tables, we report the minimum available bandwidth ($\min(BW)$) among the chosen nodes because it will impact the data transfer rate. We put the average delays ($\overline{Latency}$) among the nodes, while for the cost, we sum the cost ($\sum$Cost) of each among the selected nodes. We highlight the corresponding values from the output of SNR for each prioritized parameter in red. The obtained results confirm that SNR can select the worker nodes according to their priority. For example, it selects worker nodes with minimum delays among them when link delay has the highest priority in Tables V, VI, and VII.

B. Topology-aware Results

Custom topology. First, we use the same priority among the parameters and measure the average link delay among the chosen worker nodes by the SNR and default algorithms. In the default approach, each DSP selects the worker nodes in an ordered fashion starting from node number 1. We run the SNR and default methods to choose 4 to 8 worker nodes by assuming that all the worker nodes have the same number of available task slots to execute the streaming tasks. Fig. 3 shows that by increasing the number of worker nodes in a cluster, the average delay among them also increases in all three topologies. The obtained results show that the average delays in SNR are 1.35x, 1.42x, and 1.61x less than the default method in small, medium, and large topologies, respectively.

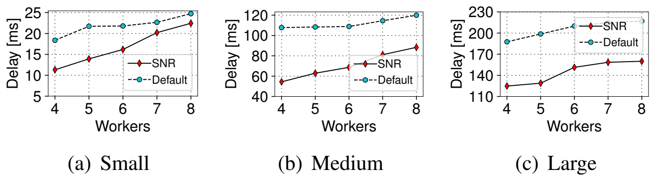


Fig. 3. The average delay among the worker nodes on custom topologies for different number of workers in a cluster network.

TopologyZoo results. We check the impact of placement in SNR on all network graphs of TopologyZoo. We use the coordination information of the nodes to compute the link delay among the nodes for the networks with such data. Fig. 4(a) shows that the SNR selects the worker nodes on average 1.44x less link delay than the default approach. Furthermore, Fig. 4(b) presents that the traffic among the nodes should cross 1.41x times more hops in the default approach than SNR. Using a fewer number of hops decreases the routing overhead among the nodes, and it also better utilizes the available capacity of the links.

C. Evaluation on Real Systems

In this section, we report the testbed used to run the Yahoo! streaming benchmark. The Yahoo! streaming benchmark [13] is a popular streaming benchmark that has been used in other research studies related to the performance evaluation of big data analytics platforms [27]. The Yahoo! streaming benchmark measures the performance of DSPs, e.g., Apache Storm, Apache Spark, and Apache Flink. The benchmark emulates an advertisement analytics pipeline in the DSPs and

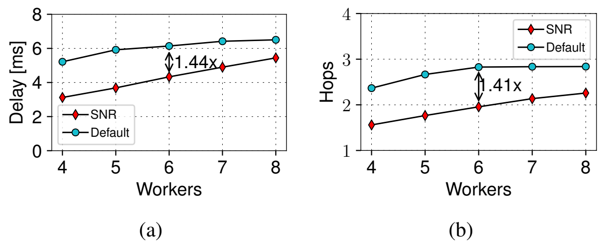
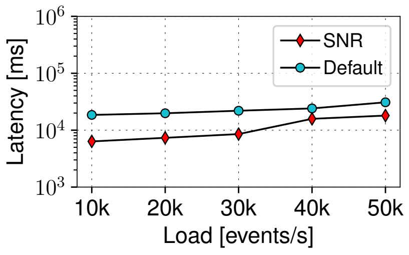


Fig. 4. SNR vs. default. (a) Average delay among the chosen nodes in TopologyZoo networks (b) Average hops among the chosen nodes in TopologyZoo networks

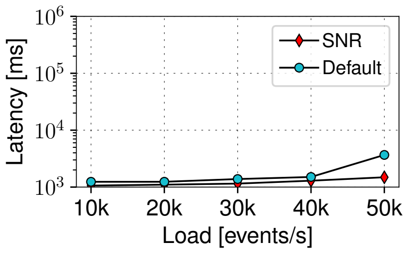
measures the performance of the various systems. There is a number of advertising campaigns in the query in which each one gets a set of advertisements. The producer of the benchmark generates events with a timestamp. Then, it truncates them to a specific digit that determines the campaign it belongs to. Each event also carries the last update timestamp along with the event information. The benchmark uses Apache Kafka [28] for event generation. After processing the event by each DSP, the benchmark calculates the event latency by deducting the window timestamp and duration from the last updated timestamp. The benchmark uses Redis [29] as the sink of query and it was the bottleneck for the benchmark. The bottleneck has been removed from the benchmark in [18]. Finally, the obtained latency value along with the number of processed events are written into appropriate files. More detailed information on the benchmark could be found in [13].

The testbed has 11 VMs with 16 CPU cores and 8 GB of RAM running Debian 10. Each DSP has a server-client architecture, where the master node executes the tasks on the set of slaves or worker nodes. We assign a VM for the master node, and 8 VMs for the worker nodes in each system. Therefore, the cluster executes the query with 8 worker nodes emulating 8 different locations in the network. Each worker node has 6 task slots to execute the streaming query and we use 6700MB of memory in each worker to use for task execution. We dedicate a VM for Kafka and a VM for the Redis database. We use Kafka and Redis VMs to generate the required input rate.

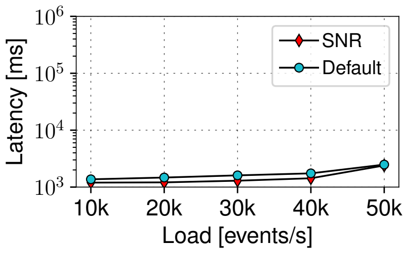
We measure the impact of the worker node placement on three custom topologies on Spark, Storm, and Flink. To do so, we apply SNR on the network graphs, i.e., small, medium, and large, to select the place of the worker nodes and obtain the network-related parameter values. Then, we use the `tc` tool to set artificial delays among the worker nodes given from running SNR. Each node is connected to other nodes via a link avoiding the routing overhead. We also apply the recommended settings of the Yahoo! streaming benchmark to set the configuration parameters in each DSP. We set the Spark batch interval to 10k milliseconds, and also the event acknowledgment of Storm to 0. We connect the Kafka and Redis VMs directly to the worker nodes. This setting allows us to simulate the scenarios in which the query execution of the system is performed close to the source. Furthermore, we use the parallelism parameter of Flink, i.e.,



(a) Spark

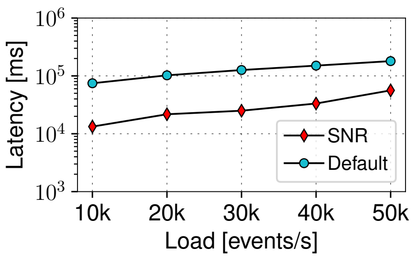


(b) Storm

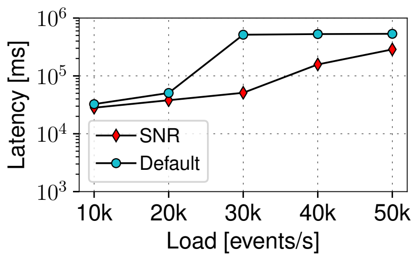


(c) Flink

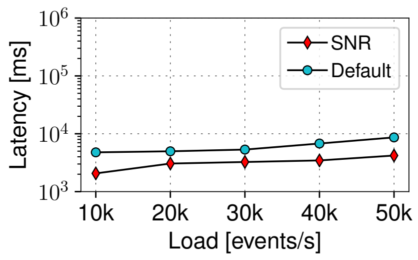
Fig. 5. The 99-percentile execution latency of Spark, Storm, and Flink on small topology by varying the input data rate.



(a) Spark



(b) Storm



(c) Flink

Fig. 6. The 99-percentile execution latency of Spark, Storm, and Flink on medium topology by varying the input data rate.

`parallelism.default`, to 48 and the settings for Spark by assigning `spark.default.parallelism` to 48. We also use 6 task executors per worker in Storm. Finally, we assume that latency has the highest priority in selecting worker nodes and assign $w_2 = 0.9$ in Eq. 5.

Small topology results. In the small topology, the nodes are close to each other, which results in having low link delay among them. However, the available bandwidth among the nodes stays unchanged in a public cloud. We vary the input rate in the range of 10k to 50k events per second. Figs. 5(a), 5(b), 5(c) show the 99-percentile execution latency for each DSP. The general trend in these three figures can be summarized as follows. By increasing the input rate of each system, the execution latency of the events also increases. However, the performance of Storm and Flink are similar, while Spark has the highest execution latency. Furthermore, the SNR improves the execution latency of Spark up to 1.5x-2.9x, Storm up to 1.1x-2.5x, and Flink up to 1.1x-1.45x compared to the default approach, respectively. The reason for such improvements is due to a low inter-node latency among the worker nodes in the DSPs. We also checked 95th- and 90th percentile latency of the task execution for Spark, Storm, and Flink, and found similar behavior. The obtained results confirm that Spark, Storm, and Flink can process the events irrespective of WAN delays when they are in the range of a few 10s of milliseconds.

Medium topology results. We do the same experiments on medium topology that has higher inter-node delays compared to those of small topology. Fig. 6 shows the execution latency in the medium topology is longer than the small topology since the internode delays are longer. However, SNR improves the execution latency of Spark up to 2.2x-7.2x, Storm up to 1.2x-3.4x, and Flink 1.4x-3.3x compared to the default approach. One of the main reasons for such results lies in Transmission Control Protocol (TCP) that suffers from high RTT among the worker nodes. The second reason for the difference among DSPs comes from the different architectures of each one. Spark streaming is not a pure stream processing system and processes the incoming stream of events in a micro-batch fashion. Therefore, the batch interval of Spark plays a determinant role here. Furthermore, Storm topology uses a different number of task executors using Spout and Bolts operators. Therefore, running a Storm topology in a geo-distributed environment needs further attention. Even though if all DSPs use the Netty framework [30] for internode communications, but the way they exchange data is different.

Large topology results. We also see similar trends in the results of all DSPs for large topology, but the SNR has less improvement since the DPSs cannot tolerate such inter-node delays in processing the incoming events. However, we exclude the relevant figures in the paper due to space limitations.

V. Related Work

This section briefly reports the current state-of-the-art placement in the geo-distributed analytics systems.

Wide-area Data Analytics. Several works have been proposed to execute queries in wide-area scenarios [1], [5], [6], [11], [31], [32]. These works consider different aspects of job execution in a wide-area such as minimizing the bandwidth usage or handling the WAN delays. For example, Kimchi [11] studies the impact of the network cost on the task placement by giving the priority to the link cost rather than bandwidth and delay. Additionally, the paper considers the dynamic of link cost in the selection procedure by applying the proposed approach on Apache Spark. RTSATD [33] minimizes the completion time and monetary cost of processing big data workflows in clouds without delaying the completion of workflows. These solutions mostly tackle the problems in scenarios in which the input data size known prior to the query execution.

Understanding the tradeoff. There are some attempts that find the tradeoff between one of the network-relevant parameters and performance in wide-area analytics. For example, some works [5], [11], [31] consider the tradeoff between the query execution and WAN usage. Nevertheless, non of these works focus on finding the tradeoff among all network-relevant parameters while SNR does.

Wide-area Stream Analytics. There are numerous works that have been considered the applications for the streaming scenarios [7]–[9], [34], [35]. In this case, there is no ending for the job execution and the input data rate can change due to several reasons like the number of users generating the data for the system. A WAN-aware scheduling algorithm for Apache Flink in multi-query scenarios has been proposed in [35]. The system checks the common parts of the input queries and schedules them to run once. The work in [34] considers WAN bandwidth limitations of a geo-distributed streaming cluster to schedule the streaming task in Apache Spark streaming using Amazon EC2. CONA [36] addresses the congestion problem in the inter-datacenter transfer methods that use the bandwidth allocation for high utilization. However, the contributions of these works consider some of the network-relevant parameters for the task execution in wide-area scenarios.

VI. Conclusion

This paper presents a simple-additive weighting-based approach for worker node placement in a geo-distributed cluster. Our algorithm SNR allows the users to place the workers according to their needs. The users can prioritize the task execution according to the network-relevant parameters such as the available bandwidth, delay, or cost. Furthermore, SNR finds the tradeoff among the available bandwidth, latency, and cost allowing the users to plan the job execution. We apply SNR on three custom topologies by changing all the relevant parameters in deciding for the placement. Furthermore, the real-world experiments using the Yahoo! streaming benchmark show that SNR improves the performance of the current distributed stream processing systems. We plan to extend our contribution to include the physical resource information of the workers in node selection. Furthermore, we plan to use the concept of Software-Defined Networking (SDN) to monitor the network for the dynamics of WAN links.

Acknowledgment

This work was partially 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] R. Viswanathan, G. Ananthanarayanan, and A. Akella, "CLARINET: Wan-aware optimization for analytics queries," in *OSDI 16*, 2016, pp. 435–450.
[2] Apache Spark, 2020, https://spark.apache.org/.
[3] A. Storm, 2020, https://storm.apache.org/.
[4] Apache Flink, 2020, https://flink.apache.org/.
[5] Q. Pu, G. Ananthanarayanan, P. Bodík, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," in *SIGCOMM'2015*, 2015, pp. 421–434.
[6] A. Vulimiri, C. Curino, B. Godfrey, K. Karanasos, and G. Varghese, "Wanalytics: Analytics for a geo-distributed data-intensive world," *CIDR 2015*, January 2015.
[7] D. Kumar, J. Li, A. Chandra, and R. Sitaraman, "A ttl-based approach for data aggregation in geo-distributed streaming analytics," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 2, Jun. 2019.
[8] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, "Awstream: Adaptive wide-area streaming analytics," in *SIGCOMM '18*, 2018, p. 236–252.
[9] F. Lai, J. You, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "Sol: Fast distributed computation over slow networks," in *NSDI'20*, 2020, pp. 273–288.
[10] B. Heintz, A. Chandra, and R. K. Sitaraman, "Optimizing timeliness and cost in geo-distributed streaming analytics," *IEEE T CLOUD COMPUT*, pp. 1–1, 2017.
[11] K. Oh, A. Chandra, and J. Weissman, "A network cost-aware geo-distributed data analytics system," in *CCGRID'20*, 2020, pp. 649–658.
[12] T. Evangelos, "Multi-criteria decision making methods: a comparative study," *Netherland: Kluwer Academic Publication*, vol. 4, 2000.
[13] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, and P. Poulosky, "Benchmarking streaming computation engines: Storm, flink and spark streaming," in *IPDPSW'16*, May 2016, pp. 1789–1792.
[14] The Internet Topology Zoo, 2020, http://bit.ly/3uQMH8O.
[15] G. P. S. P. times between WonderNetwork servers, 2020, https://wondernetwork.com/pings.
[16] J. Young and T. Barth, "Web performance analytics show even 100-millisecond delays can impact customer engagement and online revenue," 2017, Akamai Online Retail Performance Report.
[17] E. Triantaphyllou, *Multi-Criteria Decision Making Methods*. Springer, 2000, pp. 5–21.
[18] Extending the Yahoo Streaming Benchmarks, 2020, https://github.com/dataArtisans/yahoo-streaming-benchmark.
[19] ATT network delay, 2020. [Online]. Available: http://soc.att.com/30cKc2m
[20] M. A. B. P. Details, 2020, http://bit.ly/3e5PkxF.
[21] G. cloud: pricing, 2020, https://cloud.google.com/pubsub/pricing.
[22] A. E. O.-D. Pricing, 2020, http://amzn.to/3beAFOJ.
[23] akamai's [state of the internet]: Q1 2017 report, 2017, https://www.bit.ly/3jPSKEP.
[24] Microsoft Azure, https://bit.ly/3qdWimV, 2020.
[25] Google Datacenters, https://about.google/locations/, 2020.
[26] Amazon, https://amzn.to/38zsFq4, 2020.
[27] S. Zeuch, B. D. M., J. Karimov, C. Lutz, M. Renz, J. Traub, S. Breß, T. Rabl, and V. Markl, "Analyzing efficient stream processing on modern hardware," *VLDB*, vol. 12, no. 5, 2019.
[28] Apache Kafka, 2020, https://kafka.apache.org/.
[29] Redis, 2020, https://redis.io/.
[30] Netty framework, 2020, https://netty.io/.
[31] C.-C. Hung, G. Ananthanarayanan, L. Golubchik, M. Yu, and M. Zhang, "Wide-area analytics with multiple resources," in *EuroSys '18*, 2018.
[32] W. Xiao, W. Bao, X. Zhu, and L. Liu, "Cost-aware big data processing across geo-distributed datacenters," *IEEE T PARALL DISTR*, vol. 28, no. 11, pp. 3114–3127, Nov 2017.
[33] H. Chen, J. Wen, W. Pedrycz, and G. Wu, "Big data processing workflows oriented real-time scheduling algorithm using task-duplication in geo-distributed clouds," *IEEE TBDATA*, vol. 6, no. 1, pp. 131–144, 2020.
[34] W. Li, D. Niu, Y. Liu, S. Liu, and B. Li, "Wide-area spark streaming: Automated routing and batch sizing," *IEEE T PARALL DISTR*, vol. 30, no. 6, pp. 1434–1448, June 2019.
[35] A. Jonathan, A. Chandra, and J. Weissman, "Multi-query optimization in wide-area streaming analytics," in *SoCC '18*, 2018, p. 412–425.
[36] X. Tao, K. Ota, M. Dong, W. Borjigin, H. Qi, and K. Li, "Congestion-aware traffic allocation for geo-distributed data centers," *IEEE T CLOUD COMPUT*, pp. 1–1, 2020.