Energy-Aware Streaming Analytics Job Scheduling for Edge Computing

Demetris Trihinas
Department of Computer Science
University of Nicosia
trihinas.d@unic.ac.cy

Moysis Symeonides
Department of Computer Science
University of Cyprus
msymeo03@ucy.ac.cy

Joanna Georgiou
Department of Computer Science
University of Cyprus
jgeorg02@ucy.ac.cy

George Pallis
Department of Computer Science
University of Cyprus
pallis@ucy.ac.cy

Marios D. Dikaiakos
Department of Computer Science
University of Cyprus
mdd@ucy.ac.cy

***Abstract*—Energy profiling and optimization are expected to be crucial factors impacting the realisation of the Internet of Things (IoT) as more intelligence is deployed at the network extremes to achieve better response times in the proximity of where data are harvested. To improve the performance of streaming analytics jobs, several schedulers have been designed to tackle key challenges in edge computing realms, including resource heterogeneity and highly volatile network links. However, energy-aware scheduling for streaming analytic jobs is at best, not adequately examined. In this article, we introduce PowerStorm, a scheduler for streaming analytic jobs that is designed to explore trade-offs between performance and energy consumption in geo-distributed edge computing settings. We implement our scheduler for Apache Storm and show the scheduler's energy saving capabilities over the Yahoo streaming benchmark with worker nodes featuring heterogeneous power and resource capabilities on both a physical and emulated testbed.**

***Index Terms*—Big Data, Internet of Things, Energy Profiling.**

I. Introduction

The Internet of Things is emerging as the dominating service paradigm bridging the physical with the digital world through internet-enabled devices, denoted as *things*, that are capable of sensing their surroundings [1]. With IoT penetrating multiple areas of our everyday lives, new applications are emerging including autonomous robotic swarms, intelligent transportation services and AR-powered wearable technology. These applications are giving birth to an exponential growth of the data generated by more than 25 billion geo-distributed *things* with the International Data Corporation (IDC) estimating that the IoT datasphere will well surpass 79 ZB in volume by 2025 [2]. At the same time, the need for intelligent responses with better response times is apparent.

With IoT hardware vastly improving, Distributed Stream Processing Engines (DSPEs), such as Storm, Spark-Streaming and Flink, are now being explored to foster scalable, online and low-latency IoT applications where data is processed in-proximity of it's origins instead of being moved to the cloud [3]. However, data processing in edge ecosystems has its challenges [4]. In edge computing, resource heterogeneity is the norm, which contradicts with the operating requirements of DSPEs that are optimized for homogeneous machine clusters found in the cloud [5]. In turn, the network capabilities of each processing node can also significantly differ, as well as, the network distance from other nodes. The latter has the potential to significantly impact performance by creating bottlenecks and straggling tasks [6]. To overcome these challenges several streaming job schedulers have been proposed to improve the performance and scalability of stream processing in geo-distributed settings [7] [8] [9]. This is usually achieved by acknowledging heterogeneity, opting for allocating computing tasks to the most powerful worker nodes and reducing the communication overhead for data shuffling tasks [10].

According to recent estimates, the energy footprint for ICT has well surpassed the 1% mark of the global energy demand and is growing at a 4.3% annual rate [11]. Advancements in AI are further accelerating this growth [12]. Hence, more initiatives in the form of policy changes (i.e., EU green deal, UK net-zero) are calling for the migration to sustainable (edge) computing practices [13] [14]. This is further fueled by the fact that, as Gartner states, 75% of enterprise data are expected to be created and processed outside of traditional datacenters by 2025 [15]. Hence, a new problem dimension arises that is the focal point of this work; *striking a balance between energy-efficiency and performance optimization for data stream processing in edge computing settings*.

The contributions of this paper are:

- We provide a generalized problem description and an algorithmic framework that can be exploited by DSPEs to explore trade-offs between performance optimization and energy-efficiency when mapping operators of streaming analytics jobs to worker nodes.
- We introduce PowerStorm, a scheduler implemented for Apache Storm that embeds the designed energy-aware algorithmic framework.
- We evaluate the efficacy of our scheduler using the Yahoo streaming benchmark over a physical and emulated testbed comprised of multiple edge devices with heterogeneous resource, network and energy constraints.

In the experimentation, PowerStorm is compared against the default Storm scheduler and R-Storm, the open-source and most popular resource-aware Storm scheduler.

The rest of this paper is structured as follows: Section 2 provides a brief overview Storm. Section 3 provides an in-depth problem description. Sections 4 and 5 introduce PowerStorm and elaborate on our energy-aware scheduling algorithm. Section 6 provides a comprehensive experimentation analysis. Section 7 presents the related work, while Section 8 concludes the article and outlines future directions.

II. The Storm Ecosystem

Storm (https://storm.apache.org/) is a distributed computation system for processing data streams in real-time and serve results to users with low latency and high throughput. In brief, a Storm job is described as a *Topology* that is the input data structure received by the Storm cluster for continuous execution and analytics insight extraction. An analytics job may contain multiple queries and therefore, multiple Topologies. In its most simplistic form, a Topology is a Directed Acyclic Graph (DAG) comprised of multiple nodes that can take one of two forms and with edges depicting the flow of data between applied operations. Nodes can be *Spouts* or *Bolts* operators, as shown in Fig. 1. A *Spout* is linked to a data source and is in charge of data ingestion by receiving data as a stream of tuples and delegating these tuples to respective *Bolts*, based on the given Topology. Examples of data sources are various databases, distributed file-systems and even high-performance queuing services. In turn, a *Bolt* is a query operator that applies the processing logic to given data (i.e., filter, aggregation, join) and emits data further downstream.

A Storm cluster is comprised of two basic components: a leader node, denoted as *Nimbus*, and worker nodes, denoted as *Supervisors*. Nimbus, quite similar to the JobTracker in a MapReduce cluster, is the entity responsible for the job coordination that includes the placement of operators to Supervisors and the overall overview of the cluster lifecycle management (e.g., handling failures). In turn, Supervisors are the nodes that accept analytic tasks from Nimbus and coordinate their execution on the local environment they have access to. In a geo-distributed IoT environment Supervisor's may be deployed on Edge, Fog or Cloud nodes. It is worth mentioning that a third component is also required for the successful deployment of a Storm cluster, although not considered part of Storm per se. This third component is ZooKeeper, which handles the cluster communication overlay between Nimbus and the Supervisor nodes along with some additional functionality including worker health reporting.

III. Problem Description

A. Distributed Stream Processing

In distributed stream processing the scheduling problem boils down to finding a valid mapping of the operators comprising an analytics job ($O = o_1, o_2, \dots$) to the worker nodes participating in the computing cluster ($W = w_1, w_2, \dots$). For the default Storm scheduler, denoted henceforth as the Baseline, a *valid* mapping simply adopts a pseudo-random round-robin placement of operators to workers making a broad assumption that worker nodes present both homogeneous resource profiles and network distance [16]. However, not acknowledging these problem dimensions, especially in geo-distributed settings, results in some of the nodes being extremely strained and over-utilized while other nodes with resource availability not efficiently utilized.

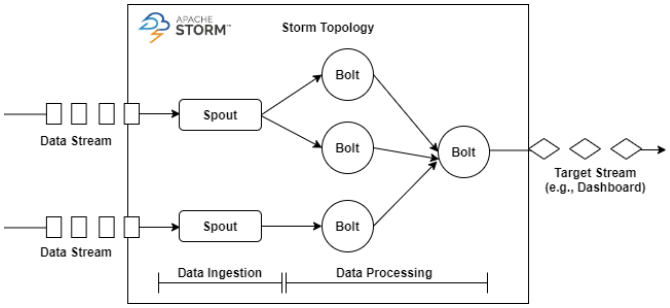

Fig. 1. An Exemplary Storm Topology

While the Baseline mapping is valid it can be far from optimal. Assuming c resources of interest (i.e., cpu, memory), the definition can be extended to any schedule that meets the resource constraints ($R = r_1, r_2, .., r_c$) of the job's operators when placed on workers with respect to their resource availability ($A = a_1, a_2, .., a_c$) based on an optimization strategy where the mapping is steered towards the maximization of one or more metrics. When optimizing stream processing performance the metric of interest is usually throughput, measured at sink nodes and defined as the number of tuples processed in a pre-defined time window (i.e., 10min). Throughput is directly impacted by the computational latency of the operations executed on the workers and the communication latency for data transfers among the operators in the path from source to sink [10]. Hence, the efficient execution of a job is constrained by the workers' resource capacities (i.e., compute, memory), as well as the communication latency between workers during the exchange of data and intermediate results. Assuming that workers can meet the resource constraints of the job's operators, the communication latency is what dominates and affects the overall throughput of a geo-distributed job [17].

B. Energy-Aware Stream Processing

Energy-awareness, and subsequently green computing, is an important aspect for IoT that can critically affect the liveness of the underlying processing infrastructure [18] as well as the overall carbon footprint of the deployment [12]. In a geo-distributed environment, edge nodes may present not only different resource and network capacity but also different operating power levels. Power, denoted as P directly impacts energy consumption ($E = P \cdot \tau$) as the amount of energy required by a computing system to execute a specific task is calculated by multiplying the power drawn with the time (τ) required for the task to finish. Power usage is reported as the sum of $P_{idle} + P_{dyn}$, where P_{idle} denotes the load-independent power drawn by the computing system, even if no

task is under execution, and P_{dyn} is load dependent. Assuming the task is a software service (i.e., data stream processing) the key components contributing to P_{dyn} are the use of processors and memory. Hence, power is an important attribute as many worker nodes may be candidates for the placement of an operator in terms of their computational and memory availability. However, when they feature different power levels then the energy consumption contributed to performing the analytic task will incur a higher energy footprint if the more power-hungry nodes are selected.

To give an example, a Raspberry Pi 4 model B presents a $P_{max} = 8W$, while the max power drawn by a Nvidia Jetson AGX Orin is $40W$ and for a Dell PowerEdge R610 $P_{max} = 330W$. All these can satisfactory fit an operator applying a simple data filter. Hence, if the task can run on the Raspberry Pi, then there will be significant energy savings. Moreover, the problem can become even worse if the topology is battery-powered, as the worker selection can severely impact an analytic job in the near future with nodes deemed unavailable very early due to battery exhaustion.

Therefore, minimizing energy consumption for data stream processing in geo-distributed settings can be formalized as an optimization problem and summarized as follows:

$$minimize \sum_{j}^{|W'|} P_j \qquad W' \subseteq W \tag{1}$$

subject to:

$$\sum_{i}^{|O|} R_i \leq A_w \qquad \forall w_j \in W' \tag{2}$$

$$and \quad \sum_{j}^{|W|} S_{i,j} = 1 \qquad \forall o_i \in O \tag{3}$$

$$and \quad \sum_{b}^{|B|} q_b > Q \qquad B_{sinknodes} \subset O \tag{4}$$

Equation 1 highlights our optimization goal that is to minimize the power drawn by the subset of workers ($W' \subseteq W$) selected for operator placement. We use power and not energy since, as mentioned, energy is dependent to power multiplied by the time enforced. The limitation expressed in equation 2 ensures that resource availability of the selected workers meets the requirements of the job's operators. In turn, the limitation in equation 3 ensures that each operator is placed on a single worker with the placement vector S expressed as follows:

$$Si,j = \begin{cases} 1, & \text{if } o_i \text{ mapped to } w_j \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

However, energy-efficiency is an optimization objective where trade-offs with performance guarantees must be explored. We emphasize *trade-off* as simply minimizing energy consumption can severely penalize performance and affect QoS when network distance among workers is far from homogeneous. Towards this, an energy-aware scheduler must

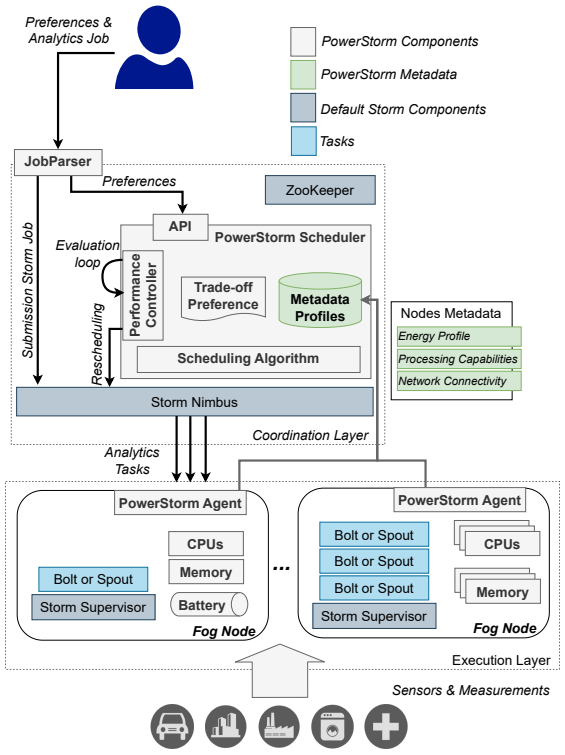

Fig. 2. PowerStorm in the Storm Ecosystem

employ an optimization process that outputs a valid mapping assigning the set of operators $o_i \in O$ to the worker nodes $w_j \in W$ with resource availability so that the overall energy consumption is reduced by making use of the least energy consuming workers when more than one candidate are available. At the same time though, the limitation expressed in equation 4 must be respected so that the overall throughput measured across the B sink nodes of the operator graph ($B \subset O$) does not drop below a user-defined threshold.

This ends up being a constraint optimization problem. Such problems are NP-complete [19], however many schedulers have been proposed to optimize the throughput of data stream processing by adopting various heuristics so that the complexity of the decision-making does not create an additional overhead to the system real-time responsiveness [7] [8]. Approximations through heuristics are required as the longer the (re-) scheduling takes to compute, the longer the downtime/delay faced by the actual data stream processing [8]. In Section V we show our heuristic-based energy-aware scheduling process for geo-distributed data stream processing jobs.

IV. PowerStorm in the Storm Ecosystem

A high-level overview of how PowerStorm components fit within the Storm ecosystem is showcased in Fig. 2. In particular, PowerStorm introduces three components, namely a *JobParser*, *PowerStorm Agents*, and the *PowerStorm Controller*, with these running alongside Storm and without requiring any adaptions to the vanilla Storm package release.

A typical analytics job starts with the user submitting via the JobParser the required artifacts along with a description of

any preferences. These, may include weights for the Scheduler's optimization function that denote the significance the Scheduler should give to performance, network overhead and energy prioritization. In turn, the user may (optionally) denote the minimum acceptable throughput that can be tolerated. This tells the Scheduler that low-power worker nodes can be preferred over power-hungry ones but only if the current throughput can be maintained above the threshold so that QoS is not penalized for energy-efficiency to a non-desired point. When this threshold is violated, the Scheduler will overwrite the energy optimization weighting leaving the algorithm to only consider resource and network-aware optimization.

With the configurations set, the job is passed to the PowerStorm Controller. The Controller is tasked with feeding Nimbus with the customized Scheduler configuration and next, the coordination of the monitoring data collection. In addition, the Controller can request the operator placement to be re-executed. Hence, the Controller does not interfere with Nimbus (cluster Leader) that handles cluster supervision, health checks and faults. Rescheduling the placement process is something PowerStorm offers as in a loosely-coupled edge deployment the worker nodes may come and go with the available resource pool substantially changing. This is available in two modes: (i) periodically, with the placement run every X minutes; or (ii) deviation-based, with the process re-executed when the resource pool deviates from the last run.

During the continuous job execution, monitoring data is collected by PowerStorm Agents with one Agent residing on each worker. Monitoring entails resource utilization, network link latency and power consumption. For the power consumption, PowerStorm has two modes of operation. The first supports the extraction of real energy consumption with data obtained through an Agent monitoring plugin that exposes the relevant interface for extension to support various energy metering libraries and devices. PowerStorm also comes with several ready-to-use templates for various smart meters (i.e., Meross). The second mode supports inference of energy data from energy models that take advantage of infrastructure utilization. This mode is ideal when embracing an IoT testing tool such as Fogify [20] with which PowerStorm can be integrated. Nonetheless, users are free to adopt their own energy models by, again, extending the PowerStorm Agent plugin interface.

V. Energy-Aware Scheduling

Algorithm 1 provides a high-level overview of the energy-aware scheduling to output a valid mapping of operators to the current workers comprising the computing environment. The algorithm starts by considering a heuristic (line 2) where topological ordering is applied on the Storm job graph. Topological ordering is feasible under the assumptions that the graph presents no cycles and there is at least one node with zero in-degree. These are met, as Storm jobs are directed acyclic graphs and for actual data processing at least one node must assume the role of a Spout for data ingestion. Hence, topological ordering is applicable and utilized to obtain a natural ordering of the operators so that operators exchanging data (adjacent graph nodes) are scheduled in (near) succession to exploit data locality. For example, suppose o_i and o_k are adjacent operators and that o_i is placed on worker w_j. If the resource availability of this worker permits $R_{o_k} <= (A_{w_j} - R_{o_i})$ then it would benefit the most to prioritize the placement of o_k on the same node rather than some other operator.

Now, if adjacent operators cannot be placed together then they should be placed close to each other as functioning in geo-distributed settings introduces, as mentioned in Section III, a significant communication overhead. Latency is directly linked to the improvement of the overall throughput as it defines the highest frequency at which analytic computations can be outputted by the given topology. At this point we consider a second heuristic to avoid measurements for all network links between workers and for all operator pairs. Specifically, to reduce the communication overhead one must minimize the latency aggregated from source to sinks. In an unrealistic scenario where all worker nodes feature infinite resource availability this boils down to a *critical path* problem with a greedy solution and a $O(V+E)$ complexity. A critical path is denoted as the slowest path from source to sink in the operator graph. However, operators have different resource constraints and each worker features different availability. In turn, when more than one workers are candidates for operator placement, we want to opt for a worker providing energy savings.

Algorithm 1 Energy-Aware Scheduling

Input: Graph G, Workers W, UserConfigs C
Output: Placement S of the graph operators to worker nodes
Ensure: PowerStorm `Scheduler` is running
1: `S` $\leftarrow \{\}$ `//the placement holder`
2: $T \leftarrow$ `VerticesTopologicalOrdering(`G`)`
3: `Z` $\leftarrow$ `Z.weights()`
4: $q \leftarrow$ `Scheduler.getSinkThroughput()`
5: **if** $q <$ `C.throughputThreshold` **then**
6: `Z["pow"]` $\leftarrow 0$
7: **end if**
8: **for each** o **in** T **do**
9: $S \leftarrow$ `OperatorPlacement(`o, W, Z`)`
10: `S`$\leftarrow$ `S` $\cup\ s$
11: **end for**
12: **return** `S`

Algorithm 2 Operator Placement

Input: Operator o, Workers W, Weights Z
Output: Selected worker node for operator given operator
Ensure: PowerStorm `Scheduler` is running
1: `penalties` $\leftarrow \{\}$
2: $\delta \leftarrow$ `o.getParentCriticalPath()`
3: **for each** w **in** W **do**
4: $q \leftarrow$ `Scheduler.getLatency(w)`
5: `netdist` $\leftarrow\ \delta +$ `Scheduler.getLatency(w)`
6: $z_1 \leftarrow$ `Z["res"]`, $z_2 \leftarrow$ `Z["net"]`, $z_3 \leftarrow$ `Z["pow"]`
7: $p \leftarrow z_1 \cdot \sum_{k=1}^{N} |r_k - a_k|$
$+\ z_2 \cdot$ `netdist`
$+\ z_3 \cdot$ `Scheduler.getPowerLevel(w)`
8: `penalties`$\leftarrow$ `penalties` $\cup\ p$
9: **end for**
10: $w_{sel} \leftarrow$ `min(penalties)`
11: `o.updCriticalPath(q)`
12: **return** w_{sel}

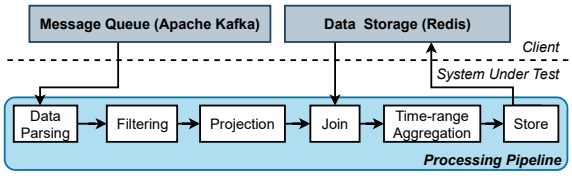

Fig. 3. High-Level Overview of the Yahoo Streaming Benchmark Workload

Hence, in Algorithm 2 which deals with the selection of a worker node for the current operator, we introduce a third heuristic. Specifically, we loosen the requirement of always opting for the worker with the smallest network distance from its predecessor and instead adopt a penalty function that considers in a normalized form; resource suitability, network distance and the worker's operating power level (lines 7-8). With this function, during operator placement we assess the suitability of each worker node and in the end take the worker with the smallest penalty (line 10).

Finally, we note that the problem dimensions feature weights for two reasons (algo. 2 - line 6). First, so that users can adjust the importance of each dimension to their pleasing. Second, so that when the user-given minimum acceptable throughput cannot be achieved, the energy saving dimension is reduced to zero with the scheduling process rolling back to optimizing solely for performance and guaranteeing QoS (algo. 1 - lines 4-7). Through this algorithmic process, PowerStorm shows preference to low-power workers, if throughput permits, and operators are placed in proximity which reduces the network overhead so that throughput is not penalized by the data exchange latency.

VI. Evaluation

This section introduces an evaluation of the PowerStorm scheduler utilizing an open and publicly accessible benchmark suite over two experimental multi-worker setups.

A. Schedulers Under Comparison

We compare the efficacy of PowerStorm against:

- the Baseline Storm scheduler, which performs a pseudo-random round-robin distribution of operators among workers nodes;
- R-Storm, a popular resource-aware scheduler favoring operator assignment to the most powerful workers to increase both performance and data locality so that the the network overhead is reduced and throughput is improved.

B. Benchmarking Suite

We adopt the open and popular Yahoo Streaming Benchmark [21] designed specifically to evaluate the performance of distributed stream processing engines. In a nutshell, the benchmark is an ad campaign analytics pipeline (Fig. 3), run on Storm, performing the following: (i) reading and parsing incoming ad traffic data, (ii) filtering, (iii) projection of unnecessary values, (iv) combining with campaign data, and (v) updating stats and storing the results. All raw streaming data are extracted from a Kafka queue, while an in-memory database (Redis) is used to store intermediate data and the

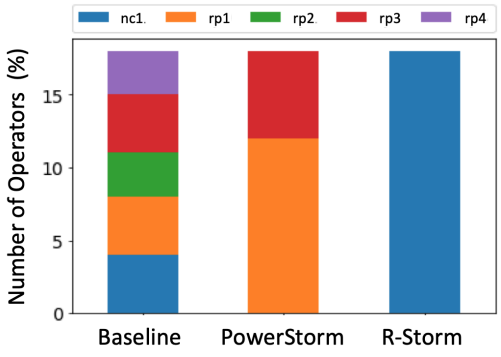

Fig. 4. Operator Placement per Worker Node

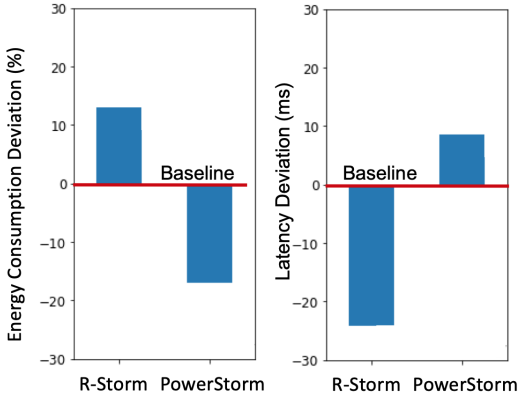

Fig. 5. Scheduler Energy and Stream Processing Latency Comparison

produced results. Through the benchmark suite one can configure many parameters of the workload (i.e., ad campaigns) which we leave to the default values and set the load to 1000 tuples/s and the benchmark duration to 30min. We adopt the vanilla implementation of Apache Storm (v2.3) and extend the deployment to include the PowerStorm JobParser and Controller on the Nimbus master node and the PowerStorm Agents on each worker node so that monitoring data for resource utilization, QoS and energy profiling are extracted.

C. Experimenting with a Physical Deployment

The first experiment run is a physical testbed with the Storm workers comprised of a power-hungry Dell PowerEdge server, denoted as *nc1*, and 4 Raspberry Pis (*rpi1*-*rpi4*). The specifications of these devices are depicted in Table I. An additional RPI is dedicated for hosting Nimbus and the Yahoo workload generator. The *nc1* is connected to a smart power distribution unit and the RPIs are connected to Meross smart meters, with PowerStorm Agents extracting energy consumption data every 5 seconds. We note that for this experiment all nodes are placed in the same network with the intra-network latency among all nodes relatively stable (measured with $< 1.2ms$ difference) so focus is driven on resource heterogeneity and the difference in operating power levels. Hence, for PowerStorm we adopt a balanced weighting (algo 2) for performance and energy prioritization during the worker selection process.

Fig. 4 and 5 depict the experiment results to draw comparisons among the under-evaluation schedulers. Fig. 4 shows the

Device Type	Quantity	Processor	Memory (GB)	Power Range (W)	Experiment Scenario
DELL PowerEdge R610 (nc1)	1	12 core@2.4GHz	12	175-330	1 and 2
Nvidia Jetson Orin Nano (nano1, nano2)	2	6 core@1.5GHz	8	7-14	2
Nvidia AGX Orin (agx1, agx2)	2	8 core@2.2GHz	32	18-40	2
Raspberry Pi 4 model B (rpi1, rpi2, rpi3, pri4)	4	4 core@1.5GHz	4	4-8	1 and 2

TABLE I
WORKER NODE RESOURCE AND POWER CONFIGURATIONS

Network	Latency (ms)	Worker Nodes Exp-2A	Worker Nodes Exp-2B
low-range	5	nc1, nano1, rpi2	nano1, nano2, rpi3
mid-range	10	rpi1, rpi4, nano2	rpi1, rpi2, agx2
far-range	15	agx1, agx2, rpi3	agx1, nc1, rpi4

TABLE II
NETWORK CONFIGURATIONS FOR EMULATED TESTBEDS

percentage of operators mapped to each worker in the deployment. From this, we immediately observe that in the case of the Baseline scheduler, operators are indeed allocated fairly to all worker nodes irrespective of their resource capabilities. Now, when embracing R-Storm, all operators are allocated to the powerful *nc1* worker as the resource availability of *nc1* is large enough and hence, there is no need to activate any of the other workers. However, when embracing PowerStorm, we observe that the operators are shared among 2 RPIs avoiding the activation of the power-hungry server.

Fig. 5 extends the evaluation to the effect (and trade-off) in energy consumption and overall latency. To draw a better understanding, the results are shown for R-Storm and PowerStorm as percentage deviations from the Baseline scheduler. From this, we first observe that the scheduling process of R-Storm has a direct positive effect in performance with the overall latency reduced by 24ms in comparison to the Baseline. This effect is of course due to the fact that all operators end up on the same worker eliminating the need for data to move across the network. However, to sufficiently achieve the required computations, *nc1* operates at high power levels that result in a 14% increment in the overall energy consumption, again in comparison to the Baseline. On the other hand, PowerStorm is able to take advantage of the low-power RPIs and hence energy consumption is reduced by 19% in comparison to the Baseline and 43% in comparison to R-Storm. In turn, the energy savings of PowerStorm come with only a slight performance penalty with the overall latency for stream processing only increasing by 9ms.

D. Experimenting with Emulated Deployments

The next two experiments are run on an Openstack computing cluster employing Fogify [20] to carve emulated edge devices with heterogeneous resource and power features, while also establishing network connections among the devices by shaping the connectivity to the requirements of each experiment. Table I presents the resource configurations for all worker nodes comprising the Storm cluster where 4 completely different emulated device profiles are embraced. In particular, an additional Nvidia Jetson Nano is dedicated for

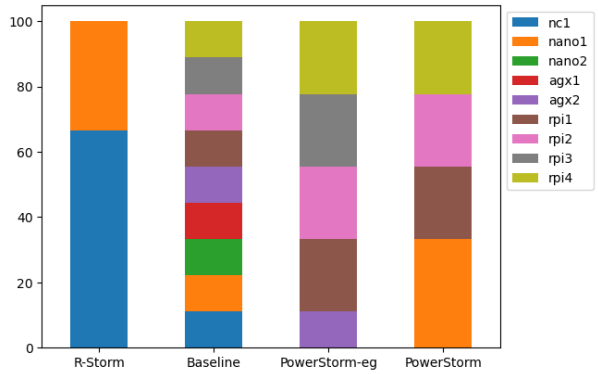

Fig. 6. Operator Placement - Exp A

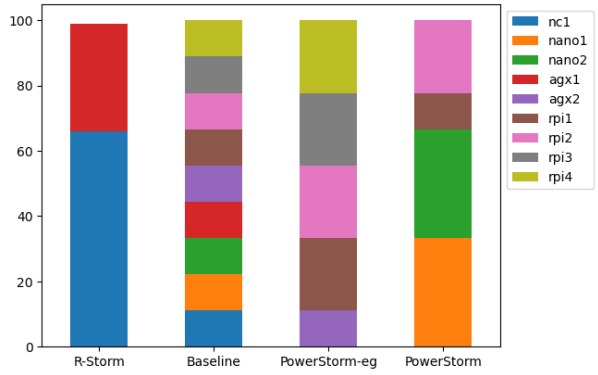

Fig. 7. Operator Placement - Exp B

hosting Nimbus and the Yahoo workload generator. Next, we request from Fogify to create 3 wide-area networks with varying latency from the workload generator (Storm Spout) and to randomly place 3 worker nodes in each network as shown in Table II. For thoroughness, this random assignment is performed twice so that we can show two experiment runs with obvious (detrimental) scenarios avoided (i.e., 3 rpi at low-range and nc1 at far-range). For PowerStorm we will adopt two modes of operation where one embraces a balanced weighting for performance, network overhead and energy prioritization during the worker selection process and the other completely favors energy optimization denoted as `PowerStorm-eg`.

We follow the fashion of the previous experiment. Fig. 6 and 7 show the percentage of operators allocated per worker depending on the scheduling strategy. We confirm, again, that the Baseline fairly allocates the operators. R-Storm, in both runs, takes advantage of *nc1* as the most powerful node and then takes the next powerful node closest to it, completely

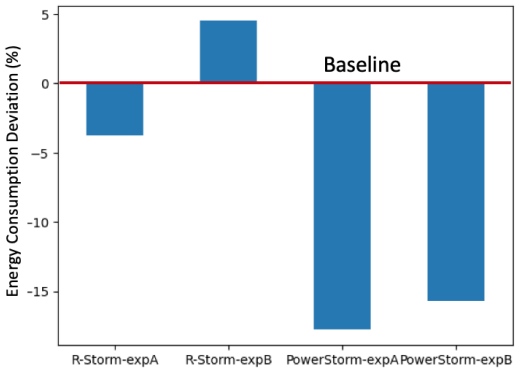

Fig. 8. Scheduler Energy Comparison

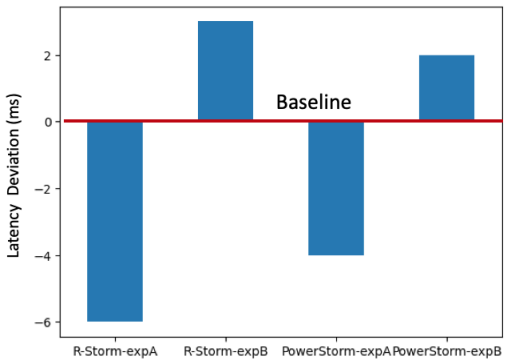

Fig. 9. Scheduler Latency Comparison

ignoring the physical distance from the data source. The fully energy-aware strategy of PowerStorm allocates the operators to the low-power nodes, in this case all 4 RPIs and 1 more node (either an *agx* or *nano*) irrespective of network distance. On the other hand, when PowerStorm embraces all three optimization dimensions, a balance is striked among energy-efficiency, performance and network overhead. In exp-2A, the *nano-1* and *rpi-2* are embraced from the low-range WAN and the two rpi's from the mid-range WAN, while in exp-2B the 2 *nano* workers and *rpi-3* are embraced from the low-range WAN and *rpi-1* from the mid-range WAN.

Next, we compare PowerStorm to R-Storm by using the Baseline scheduler as a reference point with Fig. 8 and 9 showing the results in terms of energy consumption and stream processing latency. From these we immediately observe the significant energy savings that PowerStorm can achieve. Specifically, PowerStorm reduces energy consumption by 18% and 16% respectively in comparison to the Baseline Storm scheduler. In terms of latency, now that we have a larger deployment, we observe that by utilizing less devices and placing operators close to each other and near the data source, that PowerStorm in exp-2A actually reduces latency by 4ms and in exp-2B latency is increased but only by 2ms. On the other hand, R-Storm is able to reduce energy consumption in exp-2A by almost 3% in comparison to the Baseline as only two nodes are employed (*nc1*, *nano1*) and *nc1* is operated at a lower power level since it is not fully utilized. However, in exp-2B the energy consumption increases by 4.5% in comparison to the Baseline. In terms of latency, the advertised strength of R-Storm, we see that R-Storm reduces latency by almost 6ms in exp-2A. However, for exp-2B we observe that latency increases by 2.5ms which is due to the selection process favoring two powerful nodes (*nc1*, *agx1*) that although close to each other, these nodes are in the far-range network that features high latency from the data source and hence, end up penalizing any performance gains that could be gained.

Finally, for thoroughness we provide an overview of PowerStorm behavior configured with different weights. Fig. 10 depicts for exp-2A the PowerStorm energy and latency compared to the Baseline when the weight prioritization of the power dimension varies in the range from 0 to 1. Specifically, when the weight is configured to 0 then PowerStorm completely

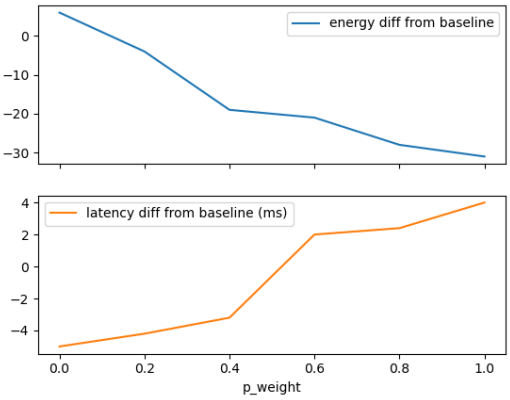

Fig. 10. PowerStorm Configured with Different Weights for Power Dimension

ignores the power levels of the workers becoming a solely performance-aware scheduler and when the weighting is fixed at 1, PowerStorm completely ignores the resource and network dimensions fixating only on worker power levels. From the results we observe that the most utility gained (elbow point) is over a weighting of 0.4 which is close to the configuration (0.33) presented in exp-2A and exp-2B.

VII. Related Work

The distributed processing landscape is large and diverse, covering schedulers, query languages and fault-tolerance among others. For brevity, this section focuses on schedulers designed for optimizing geo-distributed stream processing in Storm and how task scheduling grasps on various requirements for edge and fog computing. We reiterate that the default Storm Scheduler adopts a pseudo-random round-robin operator placement strategy to the worker nodes without exploiting data locality and worker node resource heterogeneity [16].

To tackle resource heterogeneity, R-Storm [8] shapes the placement challenge as a multi-dimensional knapsack problem in an attempt to optimize throughput through task placement when acknowledging the heterogeneity of worker nodes in terms of compute and memory. The operator placement process is performed offline making the broad assumption that the underlying execution environment will not change for the placement to remain relevant, where users must input CPU, memory and bandwidth requirements which is not always straightforward for a large and diverse set of worker nodes.

In turn, the T-Storm Scheduler [9] supports the application of query operators over streaming settings by considering the inter-node and inter-process traffic to assign workload to the nodes, rather than the default approach adopted by the Storm engine. Moreover, T-Storm does not require the use of all worker nodes on the cluster and some may end up not being used at all. Similarly, the TS-Storm Scheduler [22] attempts to solve the inter-node imbalance problem by adopting a constraint-based optimization algorithm to dynamically eliminate the performance bottleneck of the topology. Next, the T3-Scheduler for Storm [7] puts focus on placing the job's tasks that communicate with each other on nodes that are closer in terms of network distance. On a different note, EQUALITY [10] is a framework that adopts a bi-objective optimization scheduling process that explores trade-offs between performance and data quality to optimize streaming analytic jobs in constraint edge computing settings.

The following can complement our work to further improve energy efficiency for stream processing. Er-Stream [5] provides dynamic voltage scaling when the workload does not overwhelm distributed processing to reduce the power level of the workers and save energy without impacting performance. Finally, SpanEdge [23] provides a programming toolkit that enables users to specify parts of their streaming jobs that need to be scheduled close to the data sources to reduce data movement across wide-area networks.

VIII. Conclusions and Future Directions

With the pressing need to move towards sustainable practices for large-scale computing, stream processing cannot be the exception. Our work tackles energy-aware scheduling for distributed stream processing so that a balance between performance and energy-efficiency can be exploited. During the placement, PowerStorm attempts to place adjacent operators on the same worker and if not possible, on a worker close by to reduce the network overhead. At the same time, if the user-desired throughput settings permit, PowerStorm will favor low-power workers so that energy savings can be exploited at a fraction of a latency increment. Through a real-world use-case scenario and the Yahoo benchmark suite run on physical and emulated testbeds comprised of multiple edge devices with different resource, network and energy constraints, we show that PowerStorm can strike balance between performance and energy savings. In our experimentation, energy consumption is reduced by 16-43% with a slight increase in latency ranging between 2-9ms when compared against the popular R-Storm scheduler. Our future directions focus on supporting edge micro-datacenters with multiple power sources (i.e., photovoltaic panels, energy grid) and the factoring of carbon emissions in the decision-making and post-analysis.

References

[1] M. Symeonides, D. Trihinas, G. Pallis, M. D. Dikaiakos, C. Psomas, and I. Krikidis, "5g-slicer: An emulator for mobile iot applications deployed over 5g network slices," in *2022 IEEE/ACM Seventh International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 2022, pp. 115–127.

[2] D. Reinsel, "How you contribute to today's growing datasphere and its enterprise impact," *IDC*, 2019.

[3] C. Jiang, T. Fan, H. Gao, W. Shi, L. Liu, C. Cerin, and J. Wan, "Energy aware edge computing: A survey," *Computer Communications*, vol. 151, pp. 556–580, 2020.

[4] Z. Georgiou, M. Symeonides, D. Trihinas, G. Pallis, and M. D. Dikaiakos, "Streamsight: A query-driven framework for streaming analytics in edge computing," in *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, 2018, pp. 143–152.

[5] D. Sun, Y. Cui, M. Wu, S. Gao, and R. Buyya, "An energy efficient and runtime-aware framework for distributed stream computing systems," *Future Generation Computer Systems*, vol. 136, pp. 252–269, 2022.

[6] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-Driven Streaming Network Telemetry," in *Proceedings of SIGCOMM'18*, Aug 2018.

[7] L. Eskandari, J. Mair, Z. Huang, and D. Eyers, "T3-scheduler: A topology and traffic aware two-level scheduler for stream processing systems in a heterogeneous cluster," *Future Generation Computer Systems*, vol. 89, pp. 617–632, 2018.

[8] B. Peng, M. Hosseini, Z. Hong, R. Farivar, and R. Campbell, "R-storm: Resource-aware scheduling in storm," in *Proceedings of the 16th Annual Middleware Conference*, ser. Middleware '15. ACM, 2015, p. 149–161.

[9] J. Xu, Z. Chen, J. Tang, and S. Su, "T-storm: Traffic-aware online scheduling in storm," in *2014 IEEE 34th International Conference on Distributed Computing Systems*, 2014, pp. 535–544.

[10] A.-V. Michailidou, A. Gounaris, M. Symeonides, and D. Trihinas, "Equality: Quality-aware intensive analytics on the edge," *Information Systems*, vol. 105, p. 101953, 2022.

[11] E. Masanet, A. Shehabi, N. Lei, S. Smith, and J. Koomey, "Recalibrating global data center energy-use estimates," *Science*, vol. 367, no. 6481, pp. 984–986, 2020.

[12] D. A. Patterson, J. Gonzalez, Q. V. Le, C. Liang, L. Munguia, D. Rothchild, D. R. So, M. Texier, and J. Dean, "Carbon emissions and large neural network training," *CoRR*, vol. abs/2104.10350, 2021.

[13] M. Dikaiakos, N. Chatzigeorgiou, A. Tryfonos, A. Andreou, N. Loulloudes, G. Pallis, and G. Georgiou, "A cyber-physical management system for medium-scale solar-powered data centers," *Concurrency and Computation: Practice and Experience*, 2023.

[14] D. Trihinas, L. Thamsen, J. Beilharz, and M. Symeonides, "Towards energy consumption and carbon footprint testing for ai-driven iot services," in *2022 IEEE International Conference on Cloud Engineering (IC2E)*, 2022, pp. 29–35.

[15] Gartner, "What Edge Computing Means for Infrastructure and Operations Leaders."

[16] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy, "Storm@twitter," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. Association for Computing Machinery, 2014.

[17] J. Xu, B. Palanisamy, Q. Wang, H. Ludwig, and S. Gopisetty, "Amnis: Optimized stream processing for edge computing," *Journal of Parallel and Distributed Computing*, vol. 160, pp. 49–64, 2022.

[18] D. Trihinas, G. Pallis, and M. Dikaiakos, "Low-cost adaptive monitoring techniques for the internet of things," *IEEE Transactions on Services Computing*, 2018.

[19] R. Eidenbenz and T. Locher, "Task allocation for distributed stream processing," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, 2016, pp. 1–9.

[20] M. Symeonides, Z. Georgiou, D. Trihinas, G. Pallis, and M. D. Dikaiakos, "Fogify: A fog computing emulation framework," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, 2020, pp. 42–54.

[21] 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 *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2016, pp. 1789–1792.

[22] C. Li, J. Zhang, and Y. Luo, "Real-time scheduling based on optimized topology and communication traffic in distributed real-time computation platform of storm," *Journal of Network and Computer Applications*, vol. 87, pp. 100–115, 2017.

[23] H. P. Sajjad, K. Danniswara, A. Al-Shishtawy, and V. Vlassov, "Spanedge: Towards unifying stream processing over central and near-the-edge data centers," in *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, 2016, pp. 168–178.