HaaS: Cloud-based Real-time Data Analytics with **H**eterogeneity-**a**w**a**re **S**cheduling

Jiong He*, Yao Chen*, Tom Z. J. Fu*, Xin Long‡, Marianne Winslett†, Liang You‡, Zhenjie Zhang*
*Advanced Digital Sciences Center
†University of Illinois at Urbana-Champaign
‡Alibaba Group

***Abstract*—Real-time data analytics has become increasingly important in modern times as many organizations and companies are generating and analyzing high volume of data constantly. Despite of the impressive technical development, it remains a challenging job to analyze the stream data effectively and efficiently because traditional hardware and software lack specific designs and optimizations for those emerging requirements. In this paper, we discuss our experience on real-time data analytics, with our in-house processing framework *HaaS*. *HaaS* is designed to exploit existing data analytics tools and libraries as well as distributed computing technologies to embrace heterogeneous computation resources in the cloud. *HaaS* utilizes hierarchical clustering to partition physical topology of clusters weighted with task topology information into densely connected sub-graphs. *HaaS* is also equipped with a heterogeneity-aware scheduling algorithm to facilitate holistic optimization over multiple running tasks with various service level agreements. To the best of our knowledge, HaaS is the first ever streaming analytical framework providing users with flexible and optimized usage with CPUs, GPUs and FPGAs in the cloud. Users with stream processing tasks can easily enjoy remarkable advantages of CPUs, GPUs and FPGAs in throughput, power consumption and monetary cost over others. In our empirical evaluations with highly diversified workloads, HaaS saves over 18% on power consumption and 24% on monetary cost over existing system design architecture, while the overall throughput of *HaaS* remains no lower than 90% of the theoretical limit.**

***Keywords*-real-time data analytics, large-scale systems, heterogeneous computing, workload scheduling**

I. Introduction

Driven by the explosively growing data and computation power, real-time data analytics has become a hot topic in both academia and industry. e.g., in Alibaba, hundreds of diversified real-time analytical jobs are running at the same time on the cloud platform. Taobao, known as the largest e-commerce web site in the world, continues processing and analyzing users' product browsing behaviors, in order to provide accurate, customized and timely recommendations. Hundreds of thousand surveillance cameras are pushing fresh video feeds to Alibaba's cloud for real-time vehicle recognition, supporting traffic congestion evaluation and prediction. Information from all management departments and infrastructure of big cities, currently Hangzhou and Kuala Lumpur, are endlessly flowing to our cloud to facilitate fast response to emergent events, e.g., terrorist attack and flooding. While existing stream processing frameworks provide preliminary supports to these applications, new technical requirements are now rising from both system and business perspectives. The following is the summary of our observations on these new requirements.

Firstly, the scale of the data stream and analytical workload is growing so fast that the cost of data storage and processing is quickly becoming unaffordable, especially to small companies and organizations on the long tail. It has never been so important to reduce or contain the operational cost of streaming applications when running in cloud under pay-as-you-go pricing scheme.

Secondly, the increasing complexity of analytical tasks raises more request on better extensibility with domain-specific algorithms and knowledge, e.g., professional image processing software and state-of-the-art deep learning framework. The distributed stream analytical engine is supposed to host these analytical modules from their-party or in diverse programming language. Because of the heterogeneous nature of these modules, traditional scheduling algorithms for stream engine may not be able to fulfill the resource optimization target when the codes and binaries are indirectly executed or invoked.

Thirdly, even the user requirements are becoming more and more diverse. Instead of maximizing the throughput, the users may eagerly look for energy saving or monetary cost reduction. The problem gets even more complex when heterogeneous computation resource is introduced into the cloud. Such new hardware shows completely different characteristics when handling various workloads. With the wide adoption of Graphics Processing Unit (GPU) and Field programmable Gate Array (FPGA) instances in the cloud, streaming analytical system is expected to capture their characteristics and enhance the resource utilization rate based on users' customized objectives.

As a response to these challenges, we redesign existing real-time data analytics systems for modern analytical workload, i.e., *scalable* real-time analysis applications with *flexible* objective and *complex* processing logic over *heterogeneous* computation resources. Specifically, the contributions of the paper are summarized as follows.

- We propose a framework HaaS aiming at real-time data analytics by encompassing a collection of existing algorithms and primitive operations. Specifically, we provide an interface for a wide spectrum of data analytics tools, covering popular neural network, computer vision and linear algebra libraries implemented in native languages. Implemented on top of Apache Storm, HaaS provides desirable functionalities, including fault-tolerance and at-least-once guarantee.

- We generally integrate CPUs, GPUs and FPGAs resources for programmers to easily customize their own *objectives* (performance, power consumption and monetary cost). To the best of our knowledge, HaaS is the first ever system providing user-friendly accesses to various processors and accelerators for real-time data analytics in the cloud.
- Our system adapts hierarchical clustering to identify the optimal mappings between tasks and physical hardware resources. A new heterogeneity-aware scheduling algorithm is designed to achieve holistic optimizations for multiple simultaneous tasks.

II. Background and Motivation

We implement HaaS on top of Apache Storm [1] to enable support for heterogeneous resource integration and heterogeneity-aware workload scheduling. Before describing the implementation details of HaaS, we firstly introduce related background knowledge.

A. *Real-time Data Analytics*

There are extensive applications that operate on continuous input data streams such as online shopping stores, video surveillance, sensor networks and geological monitoring. These applications process data input in diverse formats such as texts, audios, images and videos which are either well-structured or unstructured. A plethora of scalable distributed streaming processing systems have been proposed such as Apache Storm [1], S4 [2], Samza [3], MillWheel [4], Photon [5] and Flink [6]. Streaming applications are usually required to deliver results in real-time so that users can obtain up-to-date insights from high volume of data instantly. A typical streaming application can be abstracted as Fig. 1.

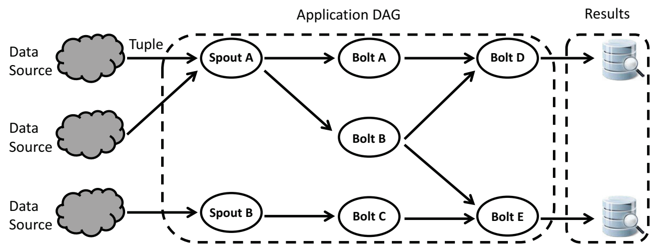


Fig. 1. Structure of typical streaming applications.

As shown in Fig. 1, incoming data from various data sources are input into applications which are abstracted as Directed Acyclic Graphs (DAGs) in streaming processing systems. Each vertex represents an *operator* with user-defined processing logic. Data is wrapped as *tuples* and intermediate tuples will flow to downstream operators for further processing. Results are continuously produced/updated. A typical use case is the recommendation system used in online shopping stores that needs to make quick recommendations to a large number of customer activities based on prediction model which will be constantly tuned with new customer activities at the same time. As shown from such application scenario, real-time streaming system needs to satisfy the following fundamental requirements [1].

- **Scalability.** Workloads change dynamically so that the system needs to adapt to varying workloads to maintain stable performance.
- **Resilience.** Distributed clusters consist of many commodity hardware nodes, resulting in non-negligible hardware failures. Such failures should not affect the running workloads significantly.
- **Low latency.** As the target application scenarios always require quick responses from the system and such responses should be made instantly upon arrival of incoming activities, low latency is one of the most important features for real-time data analytics systems.

Besides, with the growing data diversity and computing complexity, real-time data analytics always requires integration of domain-specific algorithms and primitives to deliver the final results in a collaborative manner. For instance, neural network (NN), or more commonly adopted as Convolution Neural Network (CNN), have become the main solutions to learn *patterns* from massive unstructured data such as images, audios and video automatically. Thus, it has been adopted in many applications such as object classification, detection and movement tracking. Mainstream cloud vendors such as Alibaba ECS [7] have also provided instances equipped with deep learning frameworks.

B. *Heterogeneous Cloud*

Heterogeneous cloud [8] has replaced the traditional homogeneous cloud mostly due to the diversified user requirements on performance, power consumption and monetary overhead. For certain applications, heterogeneous processing is likely to improve by an order of magnitude or even more on those performance metrics. Currently, popular processors and accelerators in the cloud include multi-core CPUs, general-purpose GPUs (GPGPUs) and FPGAs. These processors and accelerators show different advantages over others in these metrics.

Architecturally, the CPU processor is composed of few computing units but a lot of control units and hierarchical cache memory. Thus, it is efficient in processing instructions with conditional judgments and execution branches. On the contrary, the GPU accelerator consists of hundreds or even thousands of light-weight processing cores (streaming processors) and much simpler memory hierarchy. Thus, thousands of threads can be launched concurrently and the performance can outperform the counterpart implementation on the CPU by tens or even hundreds of times. The FPGA is a bulk of reconfigurable digital units that can be customized by users. Compared to CPUs and GPUs, the FPGA shows better energy efficiency and flexibility for different applications. Thus, with dedicated optimizations, FPGA can even outperform the CPU/GPU on suitable applications.

As each of them shows advantages over others in certain performance metric(s), cloud vendors provide instances for each of them so that users can tailor the cluster configurations according to the workloads. To fulfill the vision and exploit the power of heterogeneous computing, we must solve a series of technical problems. In the following, we outline these challenges and our corresponding solutions, in order to justify our system design introduced in Section III.

- Firstly, it is difficult to integrate heterogeneous accelerators into systems and provide a unified user-friendly interface for users without sacrificing performance as being used in native ways. The insufficiency of tools and libraries on FPGAs aggravates this issue. We adopt a generic bridging approach [9] to build a native access layer ("NAL" in Subsection III-A) so that any native analytics tools and libraries are accessible in high-level languages (e.g., Java) to ease the adoption in larger programmer communities.
- Secondly, unpredictable dynamics in both workloads and hardware make it difficult to configure the hardware and schedule workloads optimally in a static manner. The requirement of supporting multiple concurrently running tasks makes it even more challenging. We implement a hierarchical clustering algorithm (Subsection III-C) based on weighted matrices (Subsection III-B) generated and updated at runtime so that new optimal scheduling plans can be obtained upon dynamics.
- Thirdly, users may have different performance requirements in cloud environments. The demand for multiple tasks makes it more difficult to achieve holistic optimum. To resolve it, we propose different scheduling algorithms (Subsection III-D) to achieve that objective.

III. System Design

A. System Overview

Fig. 2 shows the architectural design of HaaS. The main functional components (FCs) of HaaS include topology recomposer (TR), topology-aware optimizer (TAO), heterogeneity-aware scheduler (HAS) and native access layer (NAL). TAO contains Metrics Manager (MM) that manages performance metrics placed in Metrics Store which maintains the latest metrics and derived weighted matrices. TAO also implements a group of operators (registrar and updater) to manipulate these data structures.

Topology Recomposer (TR). Topology recomposer receives user-defined topologies as the input. TR will insert necessary functional components that assist in collecting runtime performance statistics for scheduling and optimizations. Specifically, TR will make the following transformations to the original task topology and its environment configurations.

- **Adding metrics.** This utility function performs two tasks: define (built-in and customized) metrics of interest as well as register metric consumer bolt. All instances of each transformed topology will periodically send back registered metrics to the consumer bolt automatically. We set a global metrics store for the consumer bolt to store those metrics.
- **Tagging nodes.** Tagging is used to identify different resources in the cloud. Specifically, tagging utility function adds tags to both physical nodes (supervisors) and logical topology components (Storm spouts and bolts) to identify the type of processors so that the default routing mechanism of Storm can be replaced by optimized scheduling algorithms based on the tagging results.

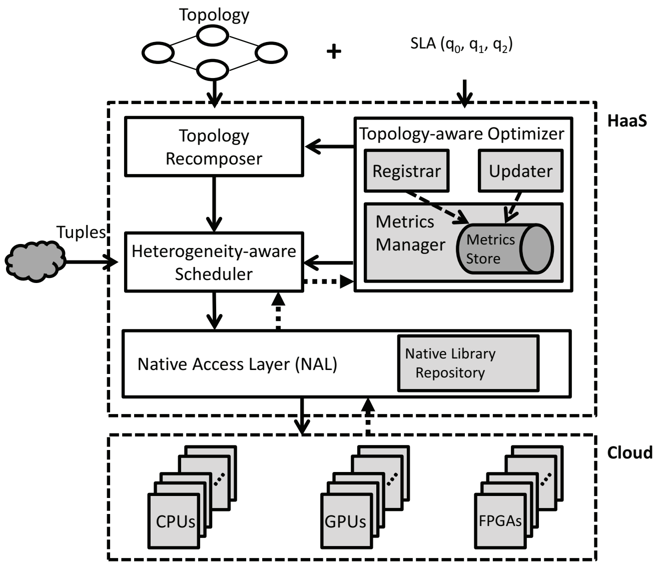


Fig. 2. System overview of HaaS.

Topology-aware Optimizer (TAO). Topology-aware optimizer monitors the hardware configurations and provides necessary information to topology recomposer and heterogeneity-aware scheduler. Specifically, the metrics registrar initializes different metrics. The updater periodically updates the performance matrices according to the latest metrics information. TAO implements the hierarchical clustering based on the latest weighted matrix. Besides, users can input their performance requirements as a Service Level Agreement (SLA) by a vector $SLA(p_0, p_1, p_2)$ where $p_i = 1$ indicates that this topology should be optimized towards throughput, power consumption or monetary cost, respectively.

Metrics Manager (MM). Metrics manager is a class that implements various customized metric definitions and keep performance weighted matrices up-to-date for TAO and HAS to produce the optimal mappings and scheduling plans. The performance weighted matrices will be introduced in Subsection III-B later in details, and these matrices depend on basic metrics which fall in two categories: hardware-related and task-related. For hardware-related metrics such as communication speed among physical nodes, MM adopts a tentative detection approach [10] tailored for heterogeneous environment. For task-related metrics, MM supports the following items.

- **Count of incoming tuples (CIT).** This metric is registered in each component to calculate the effective throughput of topologies in case where “reduction” operations are performed such as word counting.

- **Average processing delay (APD).** The average processing delay metric is specific to those bolts performing actual processing logic. It records the average time between when *execute* is called on one tuple until the tuple is *ack*ed or *fail*ed. The “ack” mechanism adopts a very light-weight implementation with a 64-bit value to track the status of each tuple emitted from spouts. It is *ack*ed only when all relevant bolts complete operations within a specific timeout on the tuple tree produced from that tuple (i.e., anchoring tuples). Otherwise, *fail* method is triggered.
- **Other metrics.** Other built-in metrics are also accessible from Storm’s native interfaces. III-B.

Heterogeneity-aware Scheduler (HAS). Heterogeneity-aware scheduler is the core component that schedules workloads (at both tuple level and task level) to optimal locations. It implements three scheduling algorithms as introduced in section III-D.

Native Access Layer (NAL). HaaS is built on top of Storm with access to many existing data analytics tools and libraries so that users can easily call those functions as like in native languages. Native access layer provides a generic and similar programming interface as of Java and Storm so that traditional programmers can easily use HaaS. To add new native libraries, users just need to expose the native interfaces to NAL via Java Native Interface (JNI) or Java Native Access (JNA). We have also implemented a native library repository by integrating existing implementations [11] such as Caffe, CUDA, OpenCV, MKL and utility functions like optical flow extractions. Specially, as there is a great gap between general-purpose data analytics libraries and their FPGA counterpart implementations due to FPGAs’ unique hardware designs procedure, we experiment with an existing neural networks framework implementated on FPGAs [12] that supports deep learning frameworks like Caffe and MXNet. Enriching the FPGA libraries with common analytics tools is ongoing work.

B. Topology-aware Optimization

To fully utilize the power of heterogeneous clusters, scheduling tasks according to the characteristics of task and hardware topologies is an important factor. Existing studies [13]–[16] mainly focus on tree-based networks and algorithms. However, such designs cannot be applied in our system. Firstly, streaming applications are not organized in tree-based structures resulting in that they cannot be mapped to tree-based network. Secondly, existing work requires to know the connection topology of underlying hardware. However, current cloud infrastructure hides such information from users with resource virtualization. These studies neither consider the heterogeneity in clouds nor the workloads. Besides, in cloud environment, multiple tasks can be concurrently running and users may define different objectives except throughput only. Thus, novel designs should be introduced to resolve these issues.

Inspired by [10], [17], [18], we construct five adjacency matrices derived from basic metrics to model the “cost” between nodes in terms of throughput, energy and monetary efficiency. These matrices can be used to represent the graphs of logical and physical topologies. However, our designs still differ from existing work greatly in the following aspects.

- First, we enhance the data structures and clustering algorithms to support different performance metrics including throughput, energy and monetary efficiency.
- Second, we propose multiple scheduling algorithm to support multiple users and multiple topologies with different requirements in terms of throughput, energy and monetary efficiency. Simple greedy algorithm would deteriorate the overall performance in cloud environments.
- Third, our data structures and scheduling algorithms are adapted to streaming and cloud environment. Therefore, it is able to capture the dynamics in both logical and physical topologies as well as changes in users’ requirements.

Among these matrices, physical data communication matrix (PM), each element of which represents the fixed communication overhead between two nodes of physical topologies, and logical data transfer matrix (LM), which captures the data transfer sizes between two nodes of logical topologies, are combined for throughput-oriented optimization.

- **Physical data transfer speed matrix (PM).** Given a cluster in the cloud consisting of a group of virtual machine nodes N_i where $i \in \{1, 2, \ldots, n\}$ and n is the cardinality of cluster, we construct an adjacency matrix PM to record the effective data transfer speed. To achieve that, we break the overhead into two parts, fixed network latency and bandwidth [10]. Each element α_{ij} represents the effective data transfer speed from physical node i to j, and the value can be obtained from the tentative detection approach [10]. The derived matrix shown as follows can be further used in performance-oriented matrices as well as clustering algorithms.

$$\mathbf{PM} = \begin{array}{c} \\ N_a \\ N_b \\ \vdots \\ N_n \end{array} \begin{array}{c} \begin{array}{cccc} N_a & N_b & \ldots & N_n \end{array} \\ \begin{pmatrix} 0 & \alpha_{ab} & \ldots & \alpha_{an} \\ \alpha_{ba} & 0 & \ldots & \alpha_{bn} \\ \ldots & \ldots & \ddots & \vdots \\ \alpha_{na} & \alpha_{nb} & \ldots & 0 \end{pmatrix} \end{array}$$

- **Logical data transfer matrix (LM).** Similarly, LM is a matrix that tracks the actual data transfer size between any two components in task topology τ (Eq. 1) with the number of tuples ($\#tuple$) and tuple size (ts). This matrix is constructed dynamically at runtime according to different workloads. To have a swift response to the latest changes in data transfer amount, LM only accumulates tuples transferred during the past ΔT time. $||LM||$ is the total size of tuples flowing through this task topology during ΔT.

$$\beta_{ij} = \frac{\#tuple * ts}{\Delta T} \tag{1}$$

- **Throughput and energy/monetary efficiency matrix (TM/EEM/MEM).** To calculate the throughput-oriented matrix, we need to combine the costs of data transfer and execution together for each topology, the latter of which can be obtained directly from metrics in MM. Thus, the value of each element (t_{ij}) in TM can be calculated as Eq. 2.

$$t_{ij} = \frac{\sum_{k}^{N_k \in \tau} CIT_k}{\frac{||LM||}{\alpha_{ij}} + \sum_{k}^{N_k \in \tau}(CIT_k * APD_k)} \tag{2}$$

For EEM and MEM, we assume a constant power for physical nodes and this assumption holds for streaming processing as streaming applications usually run for a long time and the workload on each node will not change dramatically. Besides, most cloud vendors apply time-based pricing schemes as shown in Table I. We assume that the instance price reflects the monetary cost including both hardware purchase as well as other associated cost such as electricity.

TABLE I
HARDWARE SPECIFICATIONS OF EC2 INSTANCES.

Instance types	vCPU	Instance (type)	Peak power (Watts)	Price ($/hour)
m4.xlarge	4	CPU	145	0.2
p2.xlarge	4	GPU	300	0.9
f1.2xlarge	8	FPGA	45*	1.65

* It is the upper boundary based on FPGA architectural designs.

Based on these assumptions, we further derive the energy and monetary efficiency matrices so that clustering can partition the derived graphs into different sub-graphs to achieve energy-/monetary-oriented optimization objectives. For EEM and MEM, we consider each element as the weight of edge connecting two arbitrary nodes representing the energy and monetary efficiency for unit of workload of topology τ. To unify the unit of measures across processors and accelerators, we use the ϕ_e^P and ϕ_m^P ($P \in \{CPU, GPU, FPGA\}$) as the conversion factors for energy and monetary efficiency comparisons among different processors, respectively. Similar with existing work [19], [20], the conversion factors are calculated based on the theoretical peak GFLOPS for fair comparison among different processors [21]. It is noteworthy that the performance counters calculated based on conversion factors may not produce accurate values. However, the clustering and scheduling algorithms are insensitive to moderately deviated values. Specifically, each element of EEM and MEM can be calculated as Eq. 3. ρ_j and ς_j stand for the power and price of the instance at physical node j.

$$\begin{cases} \gamma_{ij} = \dfrac{\rho_j}{\phi_e^P} \times t_{ij} \\ \theta_{ij} = \dfrac{\varsigma_j}{\phi_m^P} \times t_{ij} \end{cases} \tag{3}$$

We notice that both γ_{ij} and θ_{ij} are weighted by the throughput metric which will be constantly updated in ΔT time interval to keep track of dynamics in workloads and hardware as the energy and monetary efficiency depends not only on the underlying hardware characteristics, but also the elapsed time (longer execution time results in higher energy consumption and monetary cost) under the previous assumptions of constant power and pricing.

C. Heterogeneity-aware Multi-objective Clustering

The performance matrices can be converted to weighted undirected acyclic graphs. We denote it as (G, W) where $G = (V, E)$ and $W : w \rightarrow \mathbb{IR}$ ($\mathbb{IR}$ denotes non-negative real number set). $V = (v_0, v_1, ..., v_N)$ represents the set of all N vertices and $E = \{e_{ij} \mid v_i \in V, v_j \in V\}$ contains edges connecting two nodes v_i and v_j. A *sub-graph* G_i^* is defined as a subset of V (i.e., V_i) with associate edges connecting vertices within the subset ($V_i \subset V$). Obviously, we can easily derive the following equations (Eq. 4 to Eq. 6). Eq. 4 is the total weights of all edges connected to vertex v_i. Eq. 5 calculates the total weights of sub-graph G_i^* and Eq. 6 represents the total weights of edges connecting vertices in G_i^* and vertices in G_j^*.

$$\delta(v_i) = \sum\nolimits_{v_j \in V} w_{ji} \tag{4}$$

$$W(G_i^*) = \sum\nolimits_{v_i \in V_i} \delta(v_i) \tag{5}$$

$$cut(G_i^*, G_j^*) = \sum\nolimits_{v_i \in V_i, v_j \in V_j} w_{ij} \tag{6}$$

Modularity [22] is widely adopted to measure the closeness among all vertices of a "community" in a graph. Good modularity indicates that vertices within "communities" should be more densely connected (higher weights or degrees), and loosely connected with nodes in any other "communities". We adopt the similar modularity definition [23] and adapt it for the weighted graphs. For a division (G_0^*, G_1^*, ..., G_k-1^*), we define $k \times k$ matrix **e** where $\mathbf{e}_{ij} = cut(G_i^*, G_j^*)/W(G)$ is the fraction of total weight of edges, each of which connects one vertex in G_i^* and the other one in G_j^*. **Tr(e)** is trace of matrix **e** that represents the fraction of total weight of edges, each of which connects two vertices in the same community, to the total weights of G. Similarly, we define $a_i = \sum_{j=0}^{k} \mathbf{e}_{ij}$ as the sum of each row in **e** representing the fraction of total weight of edges that connect vertices of G_i^* and those in other sub-graphs to the total weight in graph G. Thus, the modularity of weighted graph can be defined as shown in Eq. 7, where $||\mathbf{e}^2||$ is the sum of all elements in matrix $\mathbf{e}^2$.

$$Q(G_0^*, G_1^*, \ldots, G_{k-1}^*) = \sum_{i=0}^{k-1}(e_{ii} - a_i^2) = Tr(\mathbf{e}) - ||\mathbf{e}^2|| \tag{7}$$

In Eq. 7, $Tr(\mathbf{e})$ is the fraction of weights of edges connecting vertices in the same communities, and $||\mathbf{e}^2||$ is the expected value as the first part in a *null model* [22], which is defined as a random graph with the same number of nodes, the same number of edges and the same degree distribution as in the original graph, but links among vertices are randomly distributed. To guide the clustering process, we need to further calculate the modularity change after merging two communities. We treat single vertex as singleton community before it has been assigned to any "communities". Thus, we define $\Delta Q(G^*_{mn} \Leftarrow G^*_m, G^*_n)$ to denote the difference of modularity of divisions before and after merging G^*_m and G^*_n. It is used as the judgment condition to determine whether clustering process should proceed to terminate upon optimum has been achieved.

Though modularity can be used to determine the quality of clustering results, it is necessary to define *distance* to indicate the clustering priority between two pairs of "communities". Generally, single linkage, complete linkage and average linkage are adopted as distance in hierarchical clustering. However, single linkage and complete linkage suffer from chaining or crowding problems. Thus, we adopt average weighted linkage (Eq. 8) as this calculation will not impose significant overhead to system.

$$D(G^*_i, G^*_j) = \frac{\sum_{v_i \in V^*_i, v_j \in V^*_j} w_{ij}}{W(G^*_i) \times W(G^*_j)} = \frac{cut(G^*_i, G^*_j)}{W(G^*_i) \times W(G^*_j)} \quad (8)$$

Based on the distance definition, we explore the distance space among all vertices given a graph and at each iteration, each sub-graph will be merged with the one that is closest to it (Algorithm 1). The result is a dendrogram tree as shown in Fig. 3.

Algorithm 1 Dendrogram tree generation algorithm.

```
Input:  M = {TM, EEM & MEM}
Output: Dendrogram trees $DGT_{M_i}$ for $M_i \in M$
 1: $G \Leftarrow \{G^*_0, G^*_1, \ldots, G^*_{N-1}\}$ where $N = |M_i|$
 2: $L = 0$
 3: $DGT^L_{M_i} = (V_0, V_1, \ldots, V_{N-1})$
 4: while true do
 5:     for $i = 0$ to $|DGT^L_{M_i}| - 1$ do
 6:         $idx = i + 1$
 7:         for $j = i + 1$ to $|DGT^L_{M_i}|$ do
 8:             if $D(G^*_i, G^*_j) < D(G^*_i, G^*_{idx})$ then
 9:                 $idx = j$
10:         Pair $G^*_{idx}$ with $G^*_i$ and mark it incomparable
11:     $L = L + 1$
12:     Merge vertices of paired sub-graphs and assign them to $DGT^L_{M_i}$
13:     if All vertices are in one tree node then
14:         break
15: return $DGT_{M_i}$, $L$
```

For each performance matrices, we can obtain their dendrogram trees (Fig. 3) based on Algorithm 1. At the bottom level l_0, each vertex composes a singleton community. At upper levels, these sub-graphs will be merged to other sub-graphs according to the distance (Eq. 8).

Based on the generated dendrogram tree by Algorithm 1, we can iterate over the tree and obtain different clustering

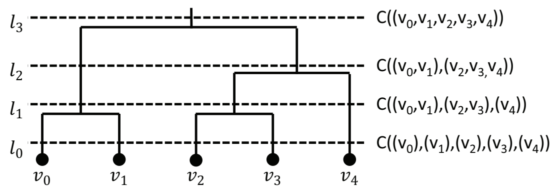


Fig. 3. Dendrogram structure for each performance matrix.

results flexibly [17]. The algorithm determines whether current clustering result will increase or decrease the modularity of the whole graph (according to ΔQ) from bottom up. Once the modularity reaches maximum, clustering process is terminated and the results at level l are output.

D. Multi-topology Coordinated Scheduling

With the clustering methods introduced in Section III-B, the physical topologies can be combined with task topologies and clustered into a group of sub-graphs according to user performance requirements. Given an objective such as throughput-oriented optimization, sub-graphs can be sorted from the most throughput-optimal to the least throughput-optimal according to the sorted total weights of sub-graphs of TM. For example, given an physical topology τ, the corresponding combined topologies for two logical topologies are τ_A and τ_B, respectively. The physical nodes fall into two different partitions for two topologies, one is ((N_0, N_1), (N_2), (N_3, N_4)), the other one is ((N_0), (N_1, N_2), (N_3, N_4)). Each of them consists of three sub-graphs. The clustering can be towards any performance metrics. We propose three different scheduling schemes (Fig. 4) to map task topologies to the physical resources.

- **Round-robin (RR).** With round-robin scheduling, HaaS ignores the heterogeneity of underlying hardware as well as the logical topologies. Instead, it schedules tasks in a round-robin manner to all available physical nodes evenly. We omit it in Fig. 4 due to space constraint.
- **Greedy.** In greedy scheduling scheme, early coming topologies would be scheduled to the physical sub-clusters from the most suitable to the least suitable sub-clusters in descending order.
- **Multi-Opt.** Greedy works well in cases where only few topologies are running concurrently without intense resource race condition. However, with increasing number of users submitting topologies to the cloud, support for multi-topology-oriented optimizations becomes crucial in improving quality of service. To that end, we further propose a multi-topology optimized (denoted as "Multi-Opt") scheduling approach. In this approach, we define "crowdedness" (Eq. 9) to describe the hardware resource utilization within

each sub-graph.

$$\Psi(G_i^*) = \sum_{v_j \in V_i} (\epsilon \times \Upsilon(\Omega(v_j)))$$
$$s.t. \begin{cases} \Omega(v_j) \in \{CPU, GPU, FPGA\} \\ \Upsilon(GPU) = \Upsilon(FPGA) = 1 \\ \Upsilon(CPU) = \#vCPU \end{cases} \quad (9)$$

Here ϵ is a regulating factor for fair comparison among processors and accelerators. Its value is associated with the resource type as well as its conversion factors. Note that $vCPU$ should be equal to the number of supported concurrently running threads in multi-threading CPUs instead of the number of cores. For GPUs and FPGAs, though the device can be shared by multiple tasks with complicated designs, we treat them as indivisible because this will add unpredictable performance downgradation and unaffordable design complexity. Thus, GPUs and FPGAs can be only exclusively occupied by any one instance.

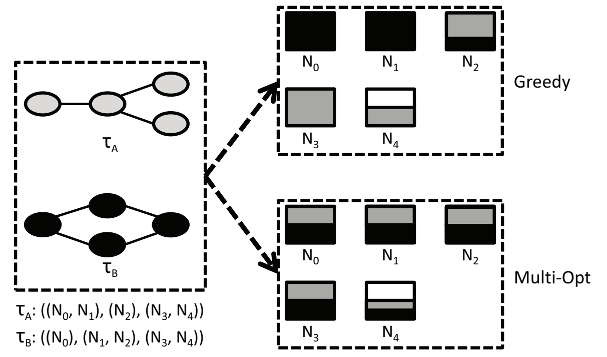

τA
τB
τA: ((N0, N1), (N2), (N3, N4))
τB: ((N0), (N1, N2), (N3, N4))
N0
N1
N2
N3
N4
Greedy
Multi-Opt

Fig. 4. Different scheduling schemes (greedy and multi-topology optimized).

To avoid the crowdedness being excessively high in some sub-graphs but low in the remaining ones, we set a fixed threshold value $\Bbbk \in \{0, 1\}$ to control the maximum percentage of hardware resources that can be occupied by instances from each topology. It is noteworthy that fully covered color in each sub-graph in Fig. 4 does not indicate that no more instances can not be scheduled to that sub-graph. In fact, any number of instances can be launched on any sub-clusters but not all of them can be simultaneously executed. Thus, the hardware utilization will not be further improved after exceeding the limit.

IV. Experimental Evaluations

In this section, we perform extensive experimental evaluations to analyze the efficiency and effectiveness of HaaS.

A. System Configurations

Experiments are conducted on a commercial cloud platform that supports CPU, GPU and FPGA instances. HaaS is built on top of Apache Storm v1.0.3. Table I lists three instance types and their hardware specifications used in our experiments. Unless otherwise stated, HaaS is evaluated on a 32-node cluster (denoted as "default") consisting of 16 CPU instances (m4.xlarge), 8 GPU instances (p2.xlarge) and 8 FPGA instances (f1.2xlarge). Each CPU instance includes one Intel Xeon E5-2676 v3 (Haswell) processor at 2.4 GHz and 16GB RAM. Each GPU instance consists one NVIDIA GRID GPU with 4GB device memory. For FPGA instances, the maximum memory capacity can be configured up to 8GB with FPGA and 16 GB corresponding to CPU. To simplify the deployment of HaaS, we export an HaaS-based instance image so that it can be easily re-deployed on any compatible new instances.

We build a benchmark with existing and hand-crafted applications. The hand-crafted applications are abstracted from more complex real-world use cases. Thus, it can be treated as a group of primitive operations to provide primitive functions in various domains including neural network (NN), computer vision (CV), text/log processing and bioinformatics. Specifically, it consists of the following applications.

- **Real-time object classification (OC).** Real-time classification tasks are important in security surveillance and traffic monitoring at a large scale [24]. Cloud-based real-time video analytics provides intelligence for users benefiting from high volume storage and computing capability. We implement a Caffe-based real-time object classification on the cloud with pre-trained models. OC can be scheduled arbitrarily for execution on any instances including FPGAs. For simplicity, we feed OC topology on the cloud with prepared video frames at controlled feeding frequency.
- **Optical flow based vehicle tracking (OF).** Optical flow algorithms track the movements of pixels of interest among consecutive frames. Typical optical flow algorithms include 1) sparse feature based Lucas-Kanade optical flow algorithm by detecting "corners" with Shi-Tomasi algorithm, and 2) dense optical flow implementation based on Gunner Farneback's algorithm. Note that for the sparse optical flow, the computation overhead may vary dramatically between frames as the number of "corners" may change significantly with time. We adopt the first algorithm to introduce computation turbulence among frames into workloads.
- **Stateful Word Count (WC).** The stateful word-count calculates the frequency of each arriving words. It is a chain-based topology that consists of a sentence splitter and a word counter components, the former of which parses sentences into separate words and the latter accumulates the number of occurrences for each word starting from the beginning. We use words from a subset of compositions of William Shakespeare [25] with more than 500,000 words in total.
- **Spike Sorting (SS).** We integrate linear algebra libraries such as Intel MKL and OpenBLAS due to the importance of basic algebraic operations in many domains such as bioinformatics. According to a real-time spike sorting implementation [26], we ignore the domain-specific knowledge and only extract the core workflow that involves matrix manipulations. SS contains a spout receiving raw data, a bolt producing an original matrix and two bolts performing Double-precision General Matrix Multiply (DGEMM) and

Singular Value Decomposition (SVD), respectively. The results will be post-processed by a sink bolt and output to the downstream (we omit those downstream operations in SS). Without losing generality, we apply SS on a 10-second hybrid data set with about 12MB size iteratively.

B. Performance Comparisons between Processors

Firstly, we evaluate each individual application on three homogeneous clusters with 16 CPUs, 8 GPUs and 8 FPGAs, respectively. So far only OC can be arbitrarily scheduled to all types of instances. For fair comparison, we normalize the values of throughput on different processors with different cluster configurations as $\widehat{T}$ (Eq. 10). Furthermore, for each application, we choose the CPU-cluster as the baseline configuration and present the relative speedups of GPU- and FPGA-cluster over it.

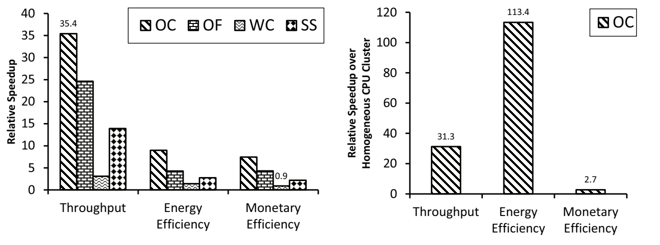
$$\widehat{T} = \frac{\sum_{k}^{N_k \in \tau} CIT_k}{time * \#nodes} \tag{10}$$

The results are calculated from a 10-minute period ($time = 10min$). As seen in Fig. 5, with default round-robin scheduling, the throughput of all four task topologies on the GPU and FPGA instances can always outperform the CPU baseline by an order of magnitude (up to 35.4 times on the GPU and 31.3 times on the FPGA) except WC (3.1 times) as for inference and numerical computations, the abundant data-level parallelism fully exploits the architectural strengths of GPUs and FPGAs. As word count consists of buffering operations and a lot of condition judgments in parsing sentences, the Single Program Multiple Data (SPMD) execution pattern of GPUs can not be fully exploited. Compared to GPUs, CPUs demonstrate good performance in processing complex logical judgments (branching instruction paths). Thus, they can execute WC comparably to GPUs.

For energy efficiency, FPGAs show dominant advantage over CPUs for the unique device characteristics. GPUs can still outperform the CPU counterparts in OC, OF and SS. However, it is only 1.4 times higher than the CPU in WC. For monetary efficiency, it is even lower on GPU than on CPU for WC (0.89 times). For OC, the advantage of FPGAs over CPUs dramatically downgrades to 2.7 times in monetary efficiency. Thus, though the performance and energy efficiency of CPUs are lower than those of GPUs and FPGAs, it is still affordable to obtain great computing capability by deploying a larger-scale CPU cluster. On the one hand, purchasing and configuring CPU instances are much easier than other instances. On the other hand, it is lacking in techniques that can efficiently virtualize and manage those emerging accelerators, raising the cost to deploy them in the cloud. The GPUs and FPGAs can provide multi-dimensional tuning knobs as they show advantages in performance and energy efficiency, respectively.

C. Performance Comparisons between Scheduling Schemes

We have proposed three scheduling algorithms, "RR" (default), "Greedy" and "Multi-Opt". We apply each of them to



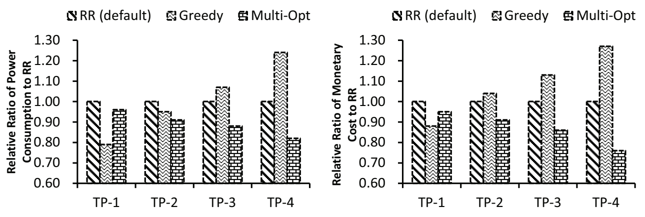
(a) GPU vs. CPU (b) FPGA vs. CPU

Fig. 5. Performance comparisons of different types of processors in homogeneous environment. The scheduling algorithm adopts the default round-robin.

the default cluster configuration and show the performance in throughput (omitted), power consumption and monetary cost, respectively. It is noteworthy that the power consumption and monetary cost are performance counters for the entire workload set instead of individual topology. As the throughput counter can be only applied to each individual topology, we omit it in this paper due to space limit.

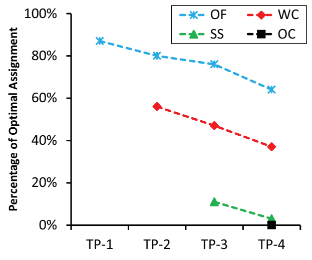
For simplicity, we set the RR as the baseline and present the relative ratios of three performance values to those of RR instead of absolute performance. Besides, as HaaS is designed for cloud environment with multiple users and task topologies, we vary the number of concurrently running topologies from 1 to 4 so that the scarcity of resources increases with the number of topologies. In Fig. 6, we use "TP-1", "TP-2", "TP-3" and "TP-4" to denote (OF), (OF, WC), (OF, WC, SS) and (OF, WC, SS, OC), respectively. Thus, ony TP-4 workloads can fully exploit the hardware heterogeneity.

When there is only one topology (TP-1) running in the cluster, all resources are accessible to it. However, when more topologies are submitted to the cluster, the default RR scheduling ignores hardware and task characteristics. All instances from each topology will be scheduled to all available nodes in a round-robin manner unless specified by users. Thus, each topology will not obtain the optimal performance. We set it as the baseline ("1" across all workloads for RR). To make it fair to compare the power consumption and monetary cost of Greedy and Multi-Opt to the baseline, we run each workload group with RR for 10 minutes and record the number of fully processed tuples of each application. Then we run them with Greedy and Multi-Opt until all topologies have fully processed the same number of tuples as that in the baseline counterpart.

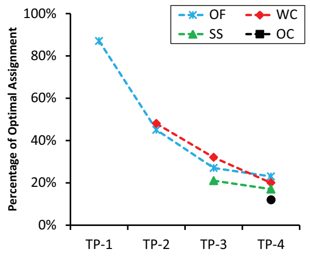


(a) Power Consumption (b) Monetary Cost

Fig. 6. Relative changes of power consumption and monetary cost to those of the default scheduling algorithm (RR) in multi-task environment.



(a) Greedy



(b) Multi-Opt

Fig. 7. The ratio of successful assignments of optimal resource to each task in multi-task environment.

As Fig. 6 shows, when the number of concurrently running topologies increases from "TP-1" to "TP-4", the Greedy scheduling algorithm can save both power consumption and monetary cost at the beginning because few topologies can be always scheduled to the most optimal sub-clusters with low possibility of race conditions. As shown in Fig. 7(a), 87% of resources of optimal sub-cluster can be assigned to the task topology (OF) in TP-1. Thus, the power consumption and monetary cost are controlled lower than those of RR due to higher hardware utilization and shorter execution time. The same cause applies to Multi-Opt. However, as the number of concurrently running topologies increases, the power consumption and monetary cost increase significantly (1.24x and 1.27x in TP-4 for power consumption and monetary cost, respectively) compared to the default scheduling algorithm. As shown in Fig. 7(a), for TP-4, though OF and WC can still obtain resources from optimal sub-cluster, almost none can be obtained by remaining topologies. Furthermore, they even can not obtain sufficient resources from sub-optimal sub-cluster as well, resulting in very long execution time (low throughput) and high cost in both power consumption and monetary cost. On the contrary, Multi-Opt can guarantee a predefined threshold fraction (it is usually set as $\frac{1}{||\text{TP-X}||}$, where $X \in \{1,2,3,4\}$, and $||\text{TP-X}||$ is the number of applications in a workload set) of resources of each sub-cluster are assignable to individual topology (Fig. 7(b)). Therefore, though the throughput of individual topology is affected, but the performance of entire workload set can be significantly improved (82% and 76% of total power consumption and monetary cost compared to RR, respectively).

D. Quality of Service (QoS)

In the cloud, tasks always need to meet certain performance requirements upon resource limit. For instance, users with limited budget may set budget limit and throughput can be sacrificed. Instead, for those throughput-sensitive tasks such as real-time product recommendation, higher monetary cost can be taken by users. For this purpose, we set various "performance requirements" in terms of throughput, power consumption and monetary cost to different topologies and launch them concurrently in HaaS. To quantitatively analyze QoS, we define an indicator QoS_i that $QoS_i = P^i_{real}/P^i_{opt}$ where P^i_{real} is the measured performance counter values for each topology in TP-4, and P_{opt} is the measured values when only one the same topology running in the cloud with "Greedy" scheduling scheme. As shown in Fig. 8, almost all performance counters can achieve more than 79% of the theoretically optimal performance, and OC and WC can even keep above 90% on some or all performance counters.

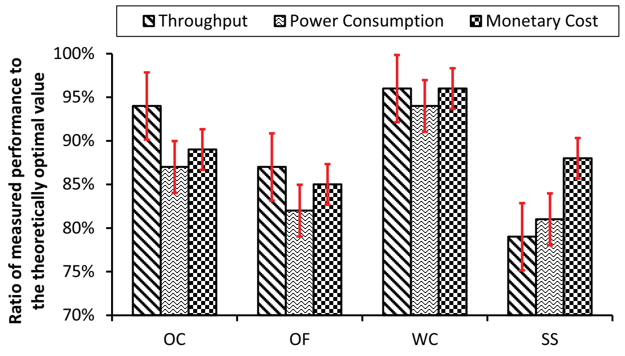


Fig. 8. Quality of Service of each application in mixed workloads compared the theoretically optimal value.

E. Adaptivity upon Dynamics

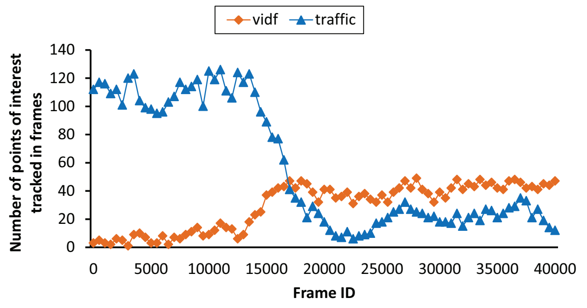
Unpredictable dynamics can occur either from data changes which may impact the evenness of workload deployment or amount of computations, or environmental changes such as hardware failures and network condition changes. Sparse optical flow tracks only those points of interest (i.e., "corners") among frames that usually represent the most obvious features of moving objects. In flowing frames, the number of corners may vary significantly and the amount of computations is affected accordingly [27]. With HaaS, such dynamics are captured and reported to the metrics manager to update the corresponding metric matrices periodically (updating frequency can be tuned flexibly as it runs independently and will not impose additional overhead to scheduling).

We apply OF to data sets that are synthesized from two existing data sets, moving pedestrians data set [28] (denoted as "vidf") and vehicle data set [29] (denoted as "traffic"), with sizes of 787MB and 60MB, respectively.

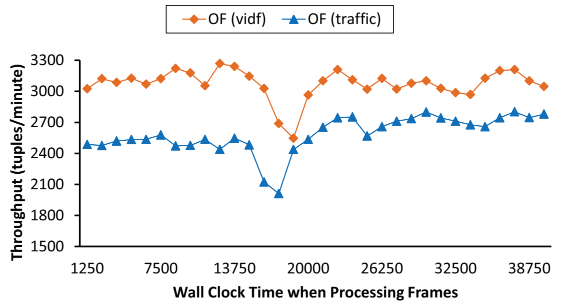
Fig. 9(a) presents the counts of points being tracked in each frame, indicating the quantity of computations. The curves change dramatically between frame #14000 to #18000. As shown in Fig. 9(b), the throughput of OF at the time when processing each frame between frame #14000 and frame #18000 will drop largely at the beginning, but returns to the original throughput as that before dynamics occur. There is slight delay between dynamics occur and the scheduler takes action due to the frequency set to update the metrics.

F. Scalability

We vary the number of instances for each type and set up five cluster configurations, i.e., *Cat-(1,0,0)*, *Cat-base(2,1,1)*, *Cat-(4,2,2)*, *Cat-(8,4,4)* and *Cat-(16,8,8)* where the integers represent the number of instances in type of CPU, GPU



(a) Number of points of interests tracked in OF.



(b) Throughput of individual topology in mixed workloads with unpredictable dynamics.

Fig. 9. Adaptivity to unpredictable dynamics in workloads.

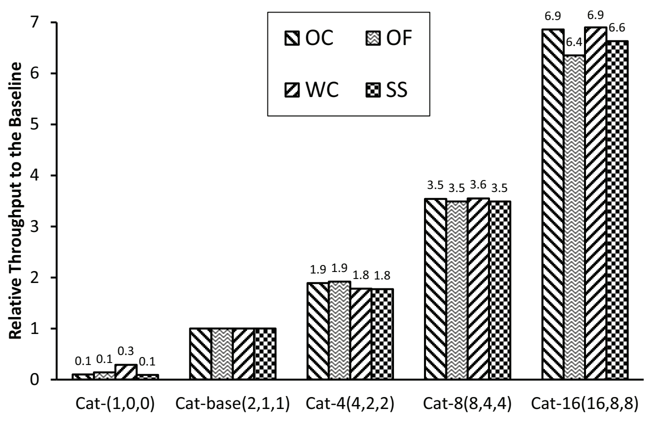


Fig. 10. Scalability: relative throughput of individual topology in mixed workloads to the baseline.

and FPGA, respectively. *Cat-base(2,1,1)* is the baseline configuration and we only present the relative throughput to the baseline. Scalability results for power consumption and monetary cost are similar with throughput and are omitted here.

As shown in Fig. 10, we present the relative throughput of each application to that in the baseline configuration. For *Cat-(1,0,0)*, though there are no GPUs or FPGAs contained, the WC can still show reasonably high performance because CPUs can process it efficiently enough. With the integration of heterogeneous resources into configurations and amount of them increases linearly, the throughput of all four applications can obtain near-linearly improved throughput improvements with heterogeneity-aware optimization and Multi-Opt scheduling algorithm. The result implies promising application prospect in increasingly larger cloud environment to accommodate larger-scale and more complex applications.

V. Related Work

With data explosion in recent years, real-time data analytics is important in analyzing continuously generated data like user-interaction events and machine logs. Streaming processing is a popular processing paradigm for such application scenarios and has been widely adopted in areas such as fraud detection, click-stream analytics, and online recommendations. Researchers have proposed different streaming data processing systems [1]–[6] that are capable of providing sub-millisecond response time and extremely high throughput on millions of events or messages per second. During a long time, streaming-based data analytics only needs to handle well-structured data in simple format. However, the increasing complexity, volume in data and diversity in user requirements impose new challenges in modern streaming-based data analytics systems.

On the one hand, streaming processing systems need to incorporate tools of multiple domains as the final results will no longer depend on single data source. For instance, building a smart city/nation [24] needs to collect and analyze data from diverse sources such as traffic cameras, audio sensors, temperature sensors, subway ticket gates and so on. Thus, a central management and scheduling system can monitor the real-time operations of all sides to make holistically optimal and efficient scheduling plans to provide benefits (security, convenience, etc.) to the public.

On the other hand, researchers also focus on approaches to improve the performance of streaming data analytics in the followings ways.

Optimizing Resource Scheduling. Streaming systems are usually deployed on clusters or cloud where multiple tasks or users may share resources. Optimal provision of resources that could minimize the resource usage without sacrificing performance and quality of service is important. Mesos [30] is a general fine-grained resource sharing system in data centers that could allow multiple cloud computing frameworks such as Hadoop, Spark, MPI and Kafka. Tom et al. [31] propose a dynamic resource scheduler for cloud-based data stream management systems. Furthermore, Dhalion [32] is proposed to achieve service level objectives (SLO) by automatically scaling resource consumption as needed. Li et al. [33] come up with a topology-aware method to accurately predict the performance according to the topology of the application graph and runtime statistics. Threads are assigned to machines under the guidance of prediction results to optimize the performance.

Adoption of Heterogeneous Accelerators. Except the efforts in optimizing scheduling algorithms, the emerging of new hardware such as GPUs [34] and FPGAs exposes larger optimization spaces for accelerating application performance. General-purpose GPUs and FPGAs have been widely adopted as accelerators in data analytics applications. GPUs have massively parallel computing capability making it suitable for applications with abundant data-level parallelism (DLP) and intensive computations such as query processing [35]–[37]. With the popularity of deep learning in tasks such as object classification, face detection, tracking and dehazing, GPUs become the standard hardware configuration for these applications as they involve a high volume of data and computations. FPGAs have also been widely adopted as an resource to accelerate general-purpose data analytic workloads. Microsoft has adopted FPGA resources to accelerate their Bing search engine in a data center scale [38]. Work in [39] adopts FPGAs to accelerates graph processing. Applications such as page-rank is also accelerated with FPGA resources [40].

With the adoption of heterogeneous accelerators in clusters and cloud, existing resource management and scheduling should also be improved with heterogeneity-aware designs. Steve et al. [8] design a preliminary cloud-based access model to clusters to incorporate heterogeneous architectures and accelerators. Chris et al. propose hStream [41] targeting heterogeneous systems with a unified interface to heterogeneous platforms so that it hides the complexity of management of task concurrency and pipeline parallelism among hardware components. Other work [42], [43] have also revisited the resource management and workload scheduling issues in the presence of heterogeneity in the cloud.

VI. Conclusions

In this paper, we propose an in-house real-time data processing framework *HaaS* by leveraging existing data analytics libraries and distributed data processing systems on the cloud to optimally utilize the heterogeneity of underlying platforms and workloads. Our empirical evaluations with highly diversified workloads show that HaaS can save saves over 18% on power consumption and 24% on monetary cost over existing system design architecture, while the overall throughput of HaaS remains no lower than 90% of the theoretical limit.

Acknowledgments

This research is supported by the National Research Foundation, Prime Ministers Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme. Besides, it is also supported by Alibaba Group through Alibaba Innovative Research (AIR) programme. Tom Fu is also supported by National Science Foundation of China (NSFC, No. 61702113) and China Postdoctoral Science Foundation (CPSF, No. 2017M612613). The cloud platform is supported by SingAREN/AWS Cloud Credits for Research Program 2016.

References

[1] 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, 2014, pp. 147–156.

[2] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *2010 IEEE International Conference on Data Mining Workshops*, 2010, pp. 170–177.

[3] S. A. Noghabi, K. Paramasivam, Y. Pan, N. Ramesh, J. Bringhurst, I. Gupta, and R. H. Campbell, "Samza: Stateful scalable stream processing at linkedin," *Proc. VLDB Endow.*, vol. 10, no. 12, pp. 1634–1645, Aug. 2017.

[4] T. Akidau, A. Balikov, K. Bekiroğlu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle, "Millwheel: Fault-tolerant stream processing at internet scale," *Proc. VLDB Endow.*, vol. 6, no. 11, pp. 1033–1044, Aug. 2013.

[5] R. Ananthanarayanan, V. Basker, S. Das, A. Gupta, H. Jiang, T. Qiu, A. Reznichenko, D. Ryabkov, M. Singh, and S. Venkataraman, "Photon: Fault-tolerant and scalable joining of continuous data streams," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13, 2013, pp. 577–588.

[6] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.

[7] Alibaba Group , https://www.alibabacloud.com/press-room/alibaba-cloud-announces-machine-learning-platform-pai.

[8] S. Crago, K. Dunn, P. Eads, L. Hochstein, D. I. Kang, M. Kang, D. Modium, K. Singh, J. Suh, and J. P. Walters, "Heterogeneous cloud computing," in *2011 IEEE International Conference on Cluster Computing*, 2011, pp. 378–385.

[9] JavaCPP , https://github.com/bytedeco/javacpp.

[10] Y. Gong, B. He, and D. Li, "Network performance aware optimizations on iaas clouds," *IEEE Trans. Comput.*, vol. 66, no. 4, pp. 672–687, Apr. 2017.

[11] JavaCPP-Presets , https://github.com/bytedeco/javacpp-presets.

[12] Mipsology , https://aws.amazon.com/marketplace/pp/B0719156K8.

[13] K. Kandalla, H. Subramoni, A. Vishnu, and D. K. Panda, "Designing topology-aware collective communication algorithms for large scale infiniband clusters: Case studies with scatter and gather," in *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, 2010, pp. 1–8.

[14] N. T. Karonis, B. R. de Supinski, I. Foster, W. Gropp, E. Lusk, and J. Bresnahan, "Exploiting hierarchy in parallel computer networks to optimize collective operation performance," in *Proceedings 14th International Parallel and Distributed Processing Symposium. IPDPS 2000*, 2000, pp. 377–384.

[15] S. Sistare, R. v. Vaart, and E. Loh, "Optimization of mpi collectives on clusters of large-scale smp' s," in *Supercomputing, ACM/IEEE 1999 Conference*, 1999, pp. 23–23.

[16] H. Subramoni, K. Kandalla, J. Vienne, S. Sur, B. Barth, K. Tomko, R. Mclay, K. Schulz, and D. K. Panda, "Design and evaluation of network topology-/speed- aware broadcast algorithms for infiniband clusters," in *2011 IEEE International Conference on Cluster Computing*, 2011, pp. 317–325.

[17] P. Fan, Z. Chen, J. Wang, Z. Zheng, and M. R. Lyu, "Topology-aware deployment of scientific applications in cloud computing," in *2012 IEEE Fifth International Conference on Cloud Computing*, 2012, pp. 319–326.

[18] S. E. Schaeffer, "Survey: Graph clustering," *Comput. Sci. Rev.*, vol. 1, no. 1, pp. 27–64, Aug. 2007.

[19] V. Adhinarayanan, T. Koehn, K. Kepa, W. c. Feng, and P. Athanas, "On the performance and energy efficiency of fpgas and gpus for polyphase channelization," in *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*, 2014, pp. 1–7.

[20] S. Mittal and J. S. Vetter, "A survey of methods for analyzing and improving gpu energy efficiency," *ACM Comput. Surv.*, vol. 47, no. 2, pp. 19:1–19:23, Aug. 2014.

[21] Altera, https://www.altera.com/en_US/pdfs/literature/wp/wp-01222-understanding-peak-floating-point-performance-claims.pdf.

[22] M. E. J. Newman, "Analysis of weighted networks," *Phys. Rev. E*, vol. 70, p. 056131, Nov 2004.

[23] A. Noack, "An energy model for visual graph clustering," in *Graph Drawing*, G. Liotta, Ed., 2004, pp. 425–436.

[24] B. Cheng, S. Longo, F. Cirillo, M. Bauer, and E. Kovacs, "Building a big data platform for smart cities: Experience and lessons from santander," in *2015 IEEE International Congress on Big Data*, 2015, pp. 592–599.

[25] The Tech , http://shakespeare.mit.edu/.

[26] C. Rossant, S. N. Kadir, D. F. M. Goodman, J. Schulman, M. L. D. Hunter, A. B. Saleem, A. Grosmark, M. Belluscio, G. H. Denfield, and A. S. e. a. Ecker, "Spike sorting for large, dense electrode arrays," *Nature Neuroscience*, vol. 19, no. 4, pp. 634–641, 2016.

[27] T. Z. Fu, J. Ding, R. T. Ma, M. Winslett, Y. Yang, Z. Zhang, Y. Pei, and B. Ni, "Livetraj: Real-time trajectory tracking over live video streams," in *Proceedings of the 23rd ACM International Conference on Multimedia*, ser. MM '15, 2015, pp. 777–780.

[28] A. B. Chan and N. Vasconcelos, "Counting people with low-level features and bayesian regression," *IEEE Transactions on Image Processing*, vol. 21, no. 4, pp. 2160–2177, 2012.

[29] A. Chan and N. Vasconcelos, "Probabilistic kernels for the classification of auto-regressive visual processes," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005, pp. 846–851 vol. 1.

[30] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11, 2011, pp. 295–308.

[31] T. Z. J. Fu, J. Ding, R. T. B. Ma, M. Winslett, Y. Yang, and Z. Zhang, "DRS: Auto-scaling for real-time stream analytics," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3338–3352, 2017.

[32] A. Floratou, A. Agrawal, B. Graham, S. Rao, and K. Ramasamy, "Dhalion: Self-regulating stream processing in heron," *Proc. VLDB Endow.*, vol. 10, no. 12, pp. 1825–1836, 2017.

[33] T. Li, J. Tang, and J. Xu, "Performance modeling and predictive scheduling for distributed stream data processing," *IEEE Transactions on Big Data*, pp. 353–364, 2016.

[34] O. John, L. David, G. Naga, H. Mark, K. Jens, L. Aaron, and P. Timothy, "A survey of general-purpose computation on graphics hardware," *Eurographics*, pp. 21–51, 2005.

[35] J. He, M. Lu, and B. He, "Revisiting co-processing for hash joins on the coupled cpu-gpu architecture," *Proc. VLDB Endow.*, vol. 6, no. 10, pp. 889–900, Aug. 2013.

[36] S. Zhang, J. He, B. He, and M. Lu, "Omnidb: Towards portable and efficient query processing on parallel cpu/gpu architectures," *Proc. VLDB Endow.*, vol. 6, no. 12, pp. 1374–1377, Aug. 2013.

[37] J. Paul, J. He, and B. He, "Gpl: A gpu-based pipelined query processing engine," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16, 2016, pp. 1935–1950.

[38] A. Caulfield, E. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A cloud-scale acceleration architecture," in *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*, October 2016.

[39] S. Zhou, C. Chelmis, and V. K. Prasanna, "High-throughput and energy-efficient graph processing on fpga," in *Field-Programmable Custom Computing Machines (FCCM), 2016 IEEE 24th Annual International Symposium on*. IEEE, 2016, pp. 103–110.

[40] Z. Shijie, C. Charalampos, and P. Viktor, "Optimizing memory performance for fpga implementation of pagerank," in *ReConFigurable Computing and FPGAs (ReConFig), 2015 International Conference on*. IEEE, 2015, pp. 1–6.

[41] C. J. Newburn, G. Bansal, M. Wood, L. Crivelli, J. Planas, A. Duran, P. Souza, L. Borges, P. Luszczek, S. Tomov, J. Dongarra, H. Anzt, M. Gates, A. Haidar, Y. Jia, K. Kabir, I. Yamazaki, and J. Labarta, "Heterogeneous streaming," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2016, pp. 611–620.

[42] M. Rychly, P. Škoda, and P. Šmr&#158;, "Scheduling decisions in stream processing on heterogeneous clusters," in *Proceedings of the 2014 Eighth International Conference on Complex, Intelligent and Software Intensive Systems*, ser. CISIS '14, 2014, pp. 614–619.

[43] M. Amaral, J. Polo, D. Carrera, S. Seelam, and M. Steinder, "Topology-aware gpu scheduling for learning workloads in cloud environments," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17, 2017, pp. 17:1–17:12.