SPARCLE: Stream Processing Applications over Dispersed Computing Networks

Parisa Rahimzadeh, Jinsung Lee, Youngbin Im, Siun-Chuon Mau, Eric C. Lee, Bradford O. Smith, Fatemah Al-Duoli, Carlee Joe-Wong and Sangtae Ha

July 18, 2020

SPARCLE: Stream Processing Applications over Dispersed Computing Networks

Parisa Rahimzadeh*, Jinsung Lee*, Youngbin Im†, Siun-Chuon Mau‡, Eric C. Lee‡, Bradford O. Smith ‡, Fatemah Al-Duoli §, Carlee Joe-Wong¶ and Sangtae Ha*
* University of Colorado Boulder, {parisa.rahimzadeh, jinsung.lee, sangtae.ha}@colorado.edu
† UNIST, ybim@unist.ac.kr
‡ CACI, {siun-chuon.mau, eric.lee, bradford.smith}@caci.com
§ System & Technology Research, fatemah.alduoli@stresearch.com
¶ Carnegie Mellon University, cjoewong@andrew.cmu.edu

***Abstract*—In this paper, we propose SPARCLE, a novel scheduling system offering network-aware polynomial-time task assignment and resource allocation algorithms for stream processing applications in dispersed computing networks. In particular, we address two major challenges. The first one concerns the assignment of *both* computation and transport tasks comprising a stream processing application to computing nodes and communication links of the network, respectively, in order to maximize the application's processing rate. The second one concerns the resource allocation of multiple stream processing applications to satisfy their requested QoE. Our experimental results on a real image stream processing application and extensive simulations show that SPARCLE can increase the application's processing rate by 9× and 3×, compared to the cloud computing case and state-of-the-art algorithms, respectively.**

***Index Terms*—Stream processing, Edge computing, Dispersed computing**

I. Introduction

"Big" data is becoming increasingly available in many application domains, driving the widespread adoption of machine learning techniques for data processing and analysis [24]. Such an emerging trend leads to the advent of a new data processing paradigm called *stream processing*, which can collect and process continuous big data streams with distributed computation running on a large cluster of machines. Examples of such data streams are from sensors, mobile devices, and online social media such as Facebook, while examples of such distributed processing engines are Apache Flink [3], Spark [4], Storm [5], etc. Up to now, stream processing applications have been typically deployed on large-scale and centralized cloud environments (e.g., datacenters). However, in addition to the growing amount of data, the enhanced capability in end devices and the deployment of edge services are now moving the computation to the edges of the network.

Dispersed computing is an emerging distributed computing paradigm where heterogeneous network nodes in proximity voluntarily share their resources (e.g., computing power, storage space, and network bandwidth) to carry out substantial amount of computation, storage, communication, and management [23]. This is a suitable computing scenario especially when sending the data to cloud servers is problematic due to the limited bandwidth of the access network, high latency or security/privacy concerns. In such scenarios, applications generally cannot access a single cluster with enough computing resources to process their data, and thus may instead need to integrate resources at multiple nearby devices. However, running these computations over a dispersed network requires new resource allocation and application placement algorithms that can address several unique challenges: constrained network connectivity, heterogeneous available resources, and dynamic network conditions caused by failures or device mobility across discrete nodes in the dispersec computing network, which is in stark contrast to servers co-located in a well-connected datacenter.

It is particularly challenging to run stream processing applications on dispersed computing systems. These applications require computing nodes to analyze a continuous stream of data produced by sources such as sensors and social networks. Such stream processing applications generally consist of multiple smaller computation tasks (CTs) with different resource requirements and dependencies, which can be modeled with a Directed Acyclic Graph (DAG) [22], [26], [29]. To account for transporting data between consecutive CTs, we define a *transport task* (TT) representing the traffic between hosts of two consecutive CTs in the DAG. While the performance effects of TTs in a datacenter environment is negligible due to reliable connectivity, they can significantly impact application QoE in a dispersed computing network.

In order to run stream processing applications on dispersed computing networks, we need a scheduler to place each of their CTs on nodes, or Networked Computing Point(s) (NCP), in the network and each of their TTs on links between these nodes. Existing schedulers used in Apache Spark [4] or Kubernetes [1] rely on heuristics for a straightforward implementation and are not network-aware. Thus, these schedulers do not consider certain important costs for stream processing applications such as those of transporting data between computing nodes. In particular, scheduling those applications over dispersed computing networks faces several new challenges.

- First, this assignment algorithm should not only be task dependency-aware, but it also should be *computing network topology-aware*. The placement of CTs affects the links on which we must place TTs, and the available computation and communication resources of computing nodes and links, respectively, must be considered. The need to consider the network topology considerably complicates the resource allocation problem as it introduces couplings between where different CTs are placed.

- Second, in stream processing, a task is repeated continuously on different incoming data units. Thus, in addition to placing the tasks on nodes and links, we must *decide the application processing rate* to ensure that the computing network is not overwhelmed. This is particularly challenging when multiple applications with different priorities and requested QoE are sharing the network. Some applications may need a guaranteed rate, while others are satisfied with any offered rate.
- Finally, the dynamic condition of the computing network affects the QoE of applications. Elements of the computing network may fail, or have intermittent availability due to mobility. The proposed algorithms must consider these dynamics as well.

In this paper, we design SPARCLE, a network-aware scheduler for stream processing applications. This is the first paper to propose and evaluate a system that both assigns tasks to devices (considering both computation and transport tasks) and allocates resources to them, for heterogeneous stream processing applications in dispersed computing networks. Our extensive experiments and simulations show the performance improvement by SPARCLE-based dispersed computing, compared to cloud computing and other state-of-the-art algorithms.

Since changing the placement of existing applications introduces potentially significant task migration costs, an application's task placement cannot be changed when new applications arrive, but the resources allocated to an application may be changed. Thus, we propose to solve the task assignment problem first and then solve for the optimal resource allocation given these assignments. We predict the available resources for each application to be placed, based on its priority and the priorities of previously placed applications, and use them in our proposed task assignment algorithm for that application. Next, given that the placement of applications is known, we solve the resource allocation optimization problem to obtain the exact resource allocation for each application. Our contributions with SPARCLE can be summarized as follows:

- Our polynomial-time network-aware task assignment algorithm to maximize the stable processing rate of an application is both network topology and task dependency-aware and can consider multiple resource types.
- We consider both single and multiple Best-Effort and Guaranteed-Rate stream processing applications.
- To best of our knowledge, this is the first work to optimize multiple task assignment paths in order to guarantee the offered QoE to applications.
- In addition to extensive simulations, we have implemented and evaluated SPARCLE in our experimental testbed with an image stream processing application. Experimental results show that SPARCLE can increase an application's processing rate by $9\times$ and $3\times$, compared to the cloud computing case and state-of-the-art algorithms, respectively. Interestingly, we also show that dispersed computing with SPARCLE could be beneficial even with high bandwidth access links, where cloud computing is expected to perform better.

The rest of the paper is organized as follows. In §II, we present related work. §III describes our models for dispersed computing networks and stream processing applications. In §IV, we present SPARCLE's problem formulation and detailed algorithms. §V evaluates the performance of SPARCLE via experiment and simulations. §VI concludes the paper. All proofs can be found in the extended version [7].

II. Related Work

There is a vast literature on task assignment, scheduling, and resource allocation algorithms for cloud-based stream processing [14], [16], [22], [29], [33], which can be run on platforms like Apache Flink [3], Spark [4], and Storm [5]. Mobile cloud computing can also partition applications with high processing demands into different tasks to be executed on the application initiator mobile device or in the cloud, e.g., [20], [30]. In contrast, we focus on stream processing in dispersed computing networks, where existing cloud-based or mobile cloud computing algorithms may not be able to handle the challenges of the dispersed computing network like heterogeneous computing devices, failure or unavailability of network elements, and limited connectivity.

The task assignment problem that we consider generalizes Virtual Network Embedding (VNE). The VNE problem deals with how a virtual network structure can be embedded on a physical network topology [12], [13], [21]. The virtual nodes and the substrate nodes in this line of work are equivalent to computation tasks and NCPs in our work, respectively. The main difference between VNE and our task assignment work is that in VNE the resource requirements of each virtual network element are given and fixed, while in our work we optimize the input rate of each stream processing application, so the computation and communication requirement of tasks of an application will change with the application's input processing rate. This extra degree of freedom can considerably complicate the already difficult VNE problem.

On the other hand, several works have studied stream processing applications in dispersed computing networks [9], [10], [25], [31]. In particular, [31] investigates general models and architectures of streaming applications, including IoT stream query and analytics, real-time event monitoring, networked control in industrial automation, and mobile crowdsourcing in the fog architecture. [9] formulates the operator placement algorithm as an integer linear programming, considering maximization of different utility functions. However, their algorithm is not network-aware, as the data transport placement between operators on network links is not being optimized. In contrast, SPARCLE is designed to conduct network-aware task assignment and resource allocation for multiple heterogeneous stream processing applications with

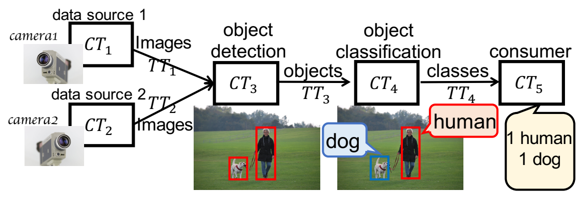


Fig. 1. An example of a stream processing application for multiple viewpoint object classification.

QoE requirements, which can further reflect network element failure and intermittency, limited network connectivity, and task graph dependencies.

The problem considered here is a new generalization of the joint-routing-and-rate control of end-to-end data flows, e.g., [28], to that of stream processing applications defined by task graphs. The generalized problem is convex if continuous load-balancing over all multipaths is allowed and only Best-Effort service is considered. In this work, we trade convexity for the accommodation of Guaranteed-Rate applications and for the avoidance of full-multipath tracking complexity. The only decentralized approach to full-multipath load-balancing that does not require explicit tracking of each paths that we are aware of is that of back-pressure, e.g., [15]. However its generalization to task graphs is not yet tackled and that would be complementary to this work.

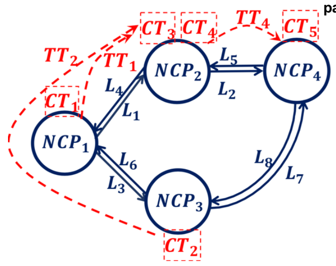
III. Application and Network Models

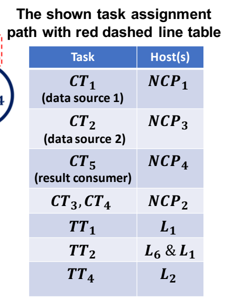
A. Stream Processing Application Model

We consider multiple heterogeneous stream processing applications that arrive over time and need to be placed on the computing network. Each stream processing application is modeled by a DAG (denoted by $\mathbb{G}$), which shows the process steps of the application's computation tasks (CTs). The data transport between *neighbor* CTs is modeled by another type of task, called a transport task (TT). CTs, denoted by the set $\mathbb{C}$, are modeled by vertices in the task graph $\mathbb{G}$, and TTs, denoted by the set $\mathbb{T}$, are the edges. Each task is associated with a resource requirement vector specifying the resources required to process one data unit, e.g., CPU cycles or the average amount of memory required per data unit for a CT and the average number of bits required per data unit for a TT. We use $a_i^{(r)}$ to denote the amount of resource type r required to process one data unit of task i. We suppose that each application has one/multiple originating CT(s) (i.e., unique vertex/vertices with no incoming links) that represent the data source(s). The processing rate of an application j, x_j, measured in data units per second, is the end-to-end rate from all data sources to all destinations.

Figure 1 illustrates the task graph of a stream processing application for multiple-viewpoint object classification. In this example, data sources CT_1 and CT_2[1] are two cameras sending stream of images, taken from different angles, as data units in the source for this application. The transport tasks TT_1 and TT_2 represent the raw image streams sent to the next CT. The

[1]Both data source and result consumers can be assumed as CTs with possibly zero resource requirements with predetermined hosts.




The shown task assignment path with red dashed line table

Task	Host(s)
CT_1 (data source 1)	NCP_1
CT_2 (data source 2)	NCP_3
CT_5 (result consumer)	NCP_4
CT_3, CT_4	NCP_2
TT_1	L_1
TT_2	L_6 & L_1
TT_4	L_2

Fig. 2. An example of the computing network (solid line), and an example of a task assignment path for the example task graph shown in Figure 1 (dashed line): The task assignment in the example (left) is listed in the table (right).

computation task CT_3 detects objects using images of different angles and sends out the found objects through TT_3. In the next computation task, CT_4, a classification algorithm is used to classify each incoming object. Then, the classified results are delivered to the result consumer CT_5 by TT_4.

Our goal is to satisfy the required QoE of the stream processing applications, which we define in terms of their *processing rate* and *availability*, based on which applications can be categorized into two groups as follows:

- Best-Effort (BE) applications do not have a minimum processing rate requirement, and will achieve higher QoE with higher processing rates. They may have an availability (having at least one working task assignment path) requirement. In order to consider their fairness, we associate a priority P_j for each BE application $j \in \mathbb{J}$, which denotes the application's relative importance compared to other BE applications in the system.
- Guaranteed-Rate (GR) applications have a minimum processing rate requirement for a specific portion of time (e.g., 2 images/sec in 90% of the time). Therefore, a *min-rate availability* is defined for GR applications, which shows the portion of time that the processing rate requirement is satisfied.

B. Dispersed Computing Network Model

We model the computing network with a graph[2], where the computing nodes (i.e., NCPs in the set $\mathbb{N}$) are vertices and links (in the set $\mathbb{L}$) are the edges of this graph (cf. Figure 2). The computation capacity of each computing node includes the computation capabilities (e.g., CPU cycles per second or Hz) of that NCP; we denote the capacity as $C_j^{(r)}$ for a given resource type r on NCP j. The communication capacity of each link includes the communication capabilities (e.g., link bandwidth) of that link and is denoted by $C_j^{(\mathrm{b})}$ for link j. Furthermore, in order to take into account the dynamics of the computing network elements, we assume each network entity j may fail or be unavailable independently during its operation with a failure probability P_{f_j}.

One placement of all CTs of an application on NCPs and all TTs on corresponding links of a computing network is called

[2]We can model the computing network with either an undirected or a directed graph, if the bandwidth of the links between two nodes is shared or not shared in different directions, respectively.

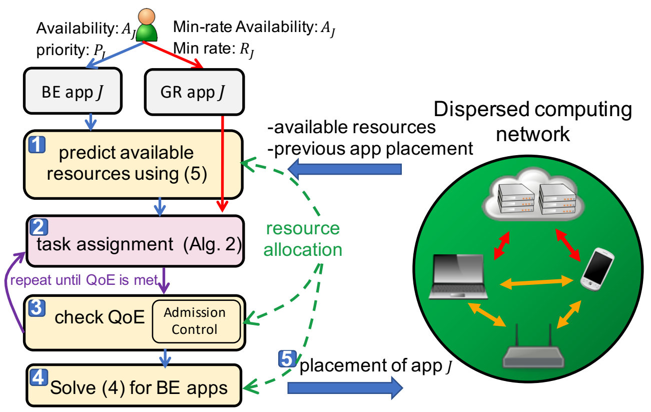


Fig. 3. An overview of SPARCLE system, which supports both BE applications with priorities and GR applications.

one task assignment "path". A stream processing application may have more than one path. Figure 2 captures one example path with dashed lines on the computing network, given the application described in Figure 1. In what follows, we consider the presence of multiple stream processing applications. Applications arrive over time and should be provided one/multiple task assignment paths, in case their QoE requirements can be met and be rejected otherwise.

IV. SPARCLE Design and Algorithm

In this section, we explain a general problem formulation for stream processing applications in a dispersed computing network. Then, we present SPARCLE, a new system offering polynomial-time task assignment and resource allocation algorithms. Figure 3 shows how SPARCLE operates to support both BE applications and GR applications together.

- In the "task assignment", we deal with the placement of CTs and TTs of an application on NCPs and links to find **one** "task assignment path" for **one** application.
- In the "resource allocation", we deal with **multiple** applications. We find the required number of task assignment paths for each application and the amounts of computation/communication resources allocated to each application, to satisfy its QoE, considering its priority.

A. Task Assignment Problem

Assume that the tasks of an application are already assigned to NCPs and links of the computing network. We must then constrain the input rates (which determine the resources allocated for each application) so that no NCP or link exceeds its resource capacity. The processing time of one data unit in the CT i hosted on NCP j is $\max_r \frac{a_i^{(r)}}{C_j^{(r)}}$, i.e., the resources required to process one data unit divided by the resource capacity of the host NCP, where the maximum is taken over all resource types $r \in \mathbb{R}$. Similarly, the transmission time of a data unit for a TT i' hosted on link j' is $\frac{a_{i'}^{(\mathrm{b})}}{C_{j'}^{(\mathrm{b})}}$, where $a_{i'}^{(\mathrm{b})}$ is the number of bits of each data unit of TT i' and $C_{j'}^{(\mathrm{b})}$ is the bandwidth of the link j'.

For an incoming data unit from the data source, all CTs and TTs are executed one-by-one based on the application's task graph. For example, in the task graph shown in Figure 1, object detection is performed on raw images and then recognized objects are classified in the object classification task. We can model this process by a queueing network, where each incoming data unit is an incoming customer and the queueing nodes are CTs and TTs. If task i is the only task placed on the computing network element (NCP or link) j, the service rate of each customer at the queueing node corresponding to task i is $\min_{r\in\mathbb{R}} \frac{C_j^{(r)}}{a_i^{(r)}}$, i.e., the inverse of the completion time defined above. The customers are routed in this queueing network based on the orders imposed by the task graph $\mathbb{G}$. A feasible task assignment and rate allocation will then ensure that this network is stable (i.e., with bounded queue lengths and hence limited delays). The input data rate (processing rate) of this application should then be less than the service rate of the slowest server (the bottleneck NCP or link) [8]. In other words, we constrain $x \le \min_{\substack{j\in\mathbb{N}\cup\mathbb{L}\\ r\in\mathbb{R}}} \frac{C_j^{(r)}}{\sum_{i\in\text{tasks placed on j}} a_i^{(r)}}$. This holds for all task assignment paths of an application and the total processing rate of an application is the summation of the rate of all task assignment paths.

For instance, for the task assignment shown in Figure 2, if there is only one resource type (e.g., CPU cycles per second for NCPs and bandwidth for links) the processing rate of this application should satisfy $x \le \min(\frac{C_{NCP_2}}{a_{CT_3}+a_{CT_4}}, \frac{C_{L_2}}{a_{TT_4}}, \frac{C_{L_6}}{a_{TT_2}}, \frac{C_{L_1}}{a_{TT_1}+a_{TT_2}})$. This constraint can be written as $Rx \le C$, where R is a vector with size $|\mathbb{N}|+|\mathbb{L}|$, which includes the sum of the loads on different computing network components. Thus, $R = [0, a_{CT_3}+a_{CT_4}, 0, 0, a_{TT_1}+a_{TT_2}, a_{TT_4}, 0, 0, 0, a_{TT_2}, 0, 0]$. Also, C is network elements' capacity vector, $C = [C_{NCP_1}, C_{NCP_2}, C_{NCP_3}, C_{NCP_4}, C_{L_1}, C_{L_2}, C_{L_3}, C_{L_4}, C_{L_5}, C_{L_6}, C_{L_7}, C_{L_8}]$. In order to find one task assignment path which maximizes the processing rate of an application, we need to solve

$$\underset{Y}{\text{maximize}} \qquad \min_{\substack{j\in\mathbb{N}\cup\mathbb{L}\\ r\in\mathbb{R}}} \frac{C_j^{(r)}}{\sum_{i\in\mathbb{C}\cup\mathbb{T}} y_{i,j} a_i^{(r)}} \tag{1a}$$

subject to

$$\sum_{j\in\mathbb{N}} y_{i,j} = 1, \forall i \in \mathbb{C}, \tag{1b}$$

$$\begin{cases} y_{TT_k,l} = 1, \forall l \in P \\ y_{TT_k,l'} = 0, \forall l' \notin P \end{cases}, \text{ if } \begin{cases} y_{i,j} = 1 \\ y_{i',j'} = 1 \\ \mathbb{G}(i,i') = \{TT_k\} \\ P \in \mathbb{P}(j,j') \end{cases}, \forall i, i' \in \mathbb{C}, \tag{1c}$$

$$y_{i,j} \in \{0,1\}, \forall i \in \mathbb{C}\cup\mathbb{T}, \forall j \in \mathbb{N}\cup\mathbb{L}, \tag{1d}$$

where Y is the decision variable vector and $y_{i,j} = 1$ if the computation/transport task i is placed on NCP/link j and 0 otherwise. The objective function in (1a) is the bottleneck processing rate. Based on the constraint (1b), each CT should be assigned to exactly one NCP. The constraint in (1c) considers the dependency of tasks in the task graph $\mathbb{G}$: if

CT i is placed on NCP j and CT i' is placed on NCP j', where CTs i and i' are neighbors in the task graph $\mathbb{G}$, which are connected by only the TT k ($\mathbb{G}(i,i') = \{TT_k\}$), the TT k should be placed on all the links on the selected path (P) among all paths between two host NCPs j and j' ($\mathbb{P}(j,j')$).

Theorem 1. *The task assignment problem in (1) is NP-hard.*

We use the intuition behind the greedy algorithm proposed for optimizing placement of independent operations on homogeneous machines [17], [18], to derive a heuristic algorithm for our task assignment problem with dependent tasks and heterogeneous NCPs/links to maximize the processing rate. Our polynomial-time greedy dynamic ranking algorithm to solve the general task assignment problem with limited-bandwidth links and heterogeneous NCPs are detailed below.

B. Task Assignment Algorithm

The request for different heterogeneous applications is submitted into the system and their tasks are placed on the computing network, by SPARCLE. The complete dynamic ranking task assignment algorithm of SPARCLE is presented in Algorithm 2. In our task assignment algorithm, tasks of an application are placed one at a time and a dynamic ranking algorithm is used to select the CT to be placed next. For each unplaced CT (denoted by the set $\mathbb{C}^u$) (line 7), we consider placing the task at different host NCPs (line 9) and examine the effect of the CT and corresponding TTs placements on the processing rate bottleneck. To check the placement of CT i on NCP j, the new processing rate imposed by NCP j would be $\min_{r\in\mathbb{R}} \frac{C_j^{(r)}}{a_i^{(r)}+\sum_{i''} y_{i'',j} a_{i''}^{(r)}}$. In addition, if the input/output of CT i is received from/sent to another CT i' (i.e., CTs i and i' are neighbors based on the task graph $\mathbb{G}$) via the TT k ($\mathbb{G}(i,i') = \{TT_k\}$), and the CT i' is previously placed on another NCP j' ($h(i') = j'$), then the TT k will be placed on all the links on the best path between NCP j and NCP j' ($P^{*k}(j,j')$), and the processing rate imposed by these links would be $\min_{l\in P^{*k}(j,j')} \frac{C_l^{(b)}}{a_k^{(b)}+\sum_{i''} y_{i'',l} a_{i''}^{(b)}}$. This is also the case for all the already placed reachable CTs of CT i. That is, if CTs i and i' are reachable CTs with the set of TTs $\mathbb{G}(i,i')$ between them, we know that at least one TT in the set $\mathbb{G}(i,i')$ will be placed on a link in the path between NCP j and NCP j'. Therefore, the new processing rate $\gamma_{i,j}$ imposed by placing CT i on NCP j is

$$\gamma_{i,j} = \min\Big(\min_{r\in\mathbb{R}} \frac{C_j^{(r)}}{a_i^{(r)} + \sum_{i''} y_{i'',j} a_{i''}^{(r)}}, \min_{\substack{i'\in\nu_i \\ l\in P^{*k}(j,j')}} \frac{C_l^{(b)}}{a_k^{(b)} + \sum_{i''} y_{i'',l} a_{i''}^{(b)}}\Big), \quad (2)$$

where ν_i is the set of placed reachable CTs of CT i and $k \in \mathbb{G}(i,i')$. We next describe how to compute the best path $P^{*k}(j,j')$ in (2) and how to use this new bottleneck processing rate in the task assignment algorithm.

Algorithm 1: The modified Dijkstra algorithm to find the best path from NCP j to NCP j' for TT k ($P^{*k}(j,j')$) in $\mathbb{G}_W$.

```
1  $\mathbb{N}^u \leftarrow \mathbb{N}$
2  $\phi[v] \leftarrow -\infty$ , $\forall v \in \mathbb{G}_W$
3  $\phi[j] \leftarrow +\infty$
4  while $\mathbb{N}^u \neq \emptyset$ do
5      $v \leftarrow \operatorname{argmax}_{i\in\mathbb{N}^u} \phi[i]$
6      if $v == j'$ then
7          return $prev$          /* $prev$ is the $P^{*k}(j,j')$ path */
8          break
9      Remove $v$ from $\mathbb{N}^u$
10     for $u \in$ neighbors of $v$ do
11         if $\min(\phi[v], weight[v,u]) > \phi[u]$ then
12             $\phi[u]=\min(\phi[v],weight[v,u])$
13             $prev[u] = v$
```

Finding the best path. Algorithm 1 summarizes how to find this best path. The best path[3] to place the TT k is defined based on the already placed TTs on links, which is the path with maximum newly imposed bottleneck on the processing rate by its links, so

$$P^{*k}(j,j') = \operatorname*{argmax}_{P(j,j')} \min_{l\in P(j,j')} \frac{C_l^{(b)}}{a_k^{(b)} + \sum_{i''} y_{i'',l} a_{i''}^{(b)}}. \quad (3)$$

We use a modified Dijkstra algorithm to find this best path in polynomial time. First, we make a weighted graph ($\mathbb{G}_W$) with the computing network graph topology, where NCPs in $\mathbb{N}$ are the vertices and links in $\mathbb{L}$ are the edges. The weight of the link l from NCP v to NCP u is weight$[v,u]$=$\frac{C_l^{(b)}}{a_k^{(b)}+\sum_{i''} y_{i'',l} a_{i''}^{(b)}}$. $\phi(v)$ is the processing rate bottleneck on the path from node j to node v, which is initialized as $-\infty$ at first (line 2), and updated inside the algorithm (line 12). The best path $P^{*k}(j,j')$ is then the path from NCP j to NCP j' with maximum minimum weight on the links along the path.

Ranking the CTs. After finding $\gamma_{i,j}$ for all $j \in \mathbb{N}$ using (2), the best host for the CT i is the NCP which imposes the maximum new processing rate, $j_i^* = \operatorname*{argmax}_j \gamma_{i,j}$ (line 15 in Algorithm 2). We use the metric γ_{i,j_i^*}, in order to rank CTs, to decide which CT should be placed first. This metric shows the new processing rate, if CT i is now placed on its best host. Therefore, the CT with the minimum imposed bottleneck on the processing rate (i.e., $i^* = \operatorname*{argmax}_i \gamma_{i,j_i^*}$) is selected to be placed first on its best host NCP j^* (line 16 in Algorithm 2). It is noteworthy that as the parameter $\gamma_{i,j}$ depends on the previously placed neighbor CTs of CT i, the ranking of CTs may change every time a CT is placed; hence, this algorithm is a dynamic ranking algorithm.

The time complexity of the proposed algorithm is polynomial in terms of the sizes of both computing network and task graph as follows.

[3]Note that the best path between any two NCPs depends on the TT that we are planning to place (by $a_k^{(b)}$ in (3)) and also it may be different in different steps of the task assignment algorithm, since it depends on the already placed TTs on links (by $y_{i'',l}$ in (3)).

Algorithm 2: SPARCLE's dynamic ranking task assignment algorithm with heterogeneous NCP and limited-bandwidth links.

```
1  $\mathbb{C}^{\text{u}} \leftarrow \mathbb{C}$
2  $\mathbb{C}^{\text{p}} \leftarrow \emptyset$
3  $y_{CT_{\text{src}},\text{src}} \leftarrow 1$    /* place all source CTs on data sources */
4  $y_{CT_{\text{snk}},\text{consumer}} \leftarrow 1$  /* place all sink CTs on result consumers */
5  add $CT_{\text{src}}$ and $CT_{\text{snk}}$ to $\mathbb{C}^{\text{p}}$ and remove from $\mathbb{C}^{\text{u}}$
6  while $\mathbb{C}^{u} \neq \emptyset$ do
7      for $i \in \mathbb{C}^{u}$ do
8          $\nu_i \leftarrow$ the set of reachable CTs of CT $i$ in $\mathbb{C}^{\text{p}}$
9          for $j \in \mathbb{N}$ do
10             for $i' \in \nu_i$ do
11                 $j' \leftarrow h(i')$
12                 $k \leftarrow \text{argmin}_y a_y^{(\text{b})}, y \in \mathbb{G}(i, i')$
13                 Use Algorithm 1 to find $P^{*k}(j, j')$
14             find $\gamma_{i,j}$ using (2)
15         $j_i^* \leftarrow \underset{j}{\text{argmax}} \gamma_{i,j}$    /* find the best host for CT $i$ */
16     $i^* \leftarrow \underset{i}{\text{argmin}}\ \gamma_{i,j_i^*}$        /* find the highest rank CT */
17     $y_{i^*,j^*} \leftarrow 1$  /* place the highest rank CT on its best host */
18     add $i^*$ to $\mathbb{C}^{\text{p}}$ and remove from $\mathbb{C}^{\text{u}}$
```

Theorem 2. *The task assignment algorithm in Algorithm 2 will run in cubic time in terms of the network and task graph size, in the worst case ($O(|\mathbb{N}|^3|\mathbb{C}|^3)$).*

C. Resource Allocation Problem

We determine the amount of resources each application receives in the computing network to satisfy the requested QoE of accepted applications. This is done by determining two variables: the number of task assignment paths of an application and the amount of allocated NCP/link resources to each application (controlled by its processing rate).

Relation with task assignment problem. We solve the task assignment and the resource allocation problems separately for BE and GR applications. However, it is important to note that in order to make sure that the answers to these problems are compatible with the joint solution of the general problem, they cannot be treated as completely independent. In particular, when the assignment algorithm is considering a potential placement of an application's tasks, the potential service rate for this application must be evaluated considering the resources that this application will receive on different NCPs/links in presence of other applications. Therefore, we should predict the assigned resources for different BE applications with different priorities, before solving the task assignment problem for them. Note that this should be done after subtracting all the resources occupied by all previously placed GR applications.

Problem for BE applications. To reflect relative importance of different BE applications in the set $\mathbb{J}$, we formulate the resource allocation optimization problem that achieves weighted proportional fairness between them as:

$$\begin{aligned} \underset{X}{\text{maximize}} \quad & \sum_{i \in \mathbb{J}} P_i \log(x_i) \\ \text{subject to} \quad & RX \leq C, \end{aligned} \tag{4}$$

where x_i is the processing rate obtained by the BE application i. This convex optimization problem can be easily solved.

In order to satisfy the requested availability of BE applications, given the failure probabilities of NCPs/links, one/multiple task assignment paths can be designated for each application. We need to find the required number of task assignment paths for each BE application (the loop in steps 2 & 3 in Figure 3) to ensure that at least one of these task assignments is functional for the target ratio of time (i.e., availability). After this, the rate of each task assignment path is determined in step 4 of Figure 3.

Problem for GR applications. GR applications have a minimum rate requirement for a specific portion of time. Since the processing rate of GR applications should be guaranteed, the resources used by the GR applications won't be shared with other later-coming applications. Thus, for a GR application J, we need to find the required number of task assignment paths N_{path_J} to guarantee the availability A_J of this minimum rate R_J (Min-rate availability), which can be formulated as

$$\begin{aligned} \text{minimize} \quad & N_{\text{path}_J} \\ \text{subject to} \quad & \mathbb{P}(r \geq R_J) \geq A_J, \end{aligned} \tag{5}$$

where r is the effective aggregate processing rate of all N_{path_J} paths. For instance, if paths do not have any intersection, $r = \sum_i r_i p_i$, where p_i is the probability of path i working and r_i is the rate of a path i.

D. Resource Allocation Algorithm

We explain how we incorporate the output of the aforementioned task assignment algorithm into our general solution for BE applications. Then, we describe resource allocation and admission control for GR applications.

Algorithm for BE applications. We first consider the problem of allocating resources to BE applications, assuming that the task assignment of applications has been done by Algorithm 2. Referring back to our original optimization problem in (4), in order for taking into account the presence of multiple applications and allocation of computing network resources to heterogeneous BE applications (e.g., in terms of task graph or priority), we need to predict the amount of resources a BE application will receive after it is placed.

Theorem 3. *After the task assignment, the minimum allocated computation/communication resources assigned to an application placed on an NCP/link by solving (4) is proportional to its priority.*

We use Theorem 3 to predict the approximate available resources of all NCPs/links for the BE application to be placed, to avoid the complexity of solving (4) with both task assignment (R) and resource allocation (X) variables. The complete task assignment and resource allocation steps for BE applications are shown in Figure 3.

For each application J with priority P_J, we first predict the available resources of NCPs and links considering their hosted tasks of previously placed applications based on

$$C_n^{\text{pred}} \leftarrow \frac{P_J}{\sum_{J' \in \mathbb{J}_n} P_{J'}} C_n, \tag{6}$$

where $\mathbb{J}_n$ is the set of all placed applications on NCP/link n, and then we use Algorithm 2 to find the task assignment for each application J with the new C_n^{pred} values for NCP/link n. For example, assume an application a has tasks placed on an NCP n with capacity C_n. When placing application b with priority twice the priority of application a (i.e., $P_b = 2P_a$), the resources of NCP n available to application b will be $C_n^{\text{pred}} = \frac{2}{3} C_n$ in the task assignment algorithm (Algorithm 2). Finally, getting the placement of all present BE applications as input, the optimization problem in (4) with rate variables is solved to get the exact processing rates for all BE applications. Importantly, using this prediction, we alleviate the effect of the arrival order of different applications and each application gets resources based on its priority independent of its arrival time.

Moreover, in the resource allocation, we need to find the required number of task assignment paths for each BE application so that it can satisfy the availability requirement. The number of paths for each BE application is increased until the requested availability is met. The availability estimation is obtained using the probabilistic analysis of the case that one of the found paths is working (BE application availability definition), considering the possible overlap between paths. For example, for the task assignment path shown in Figure 2, the availability of the application is $\prod_{j \in \{\text{used NCPs/links}\}} (1 - P_{f_j})$. If we add another path, the availability estimation will be done considering the possible overlap between paths (done in step 3 shown in Figure 3).

Algorithm for GR applications. Unlike BE applications, GR applications request for a target processing rate for specific portion of time. For example, a GR image processing application can request the processing rate of $R_J = 10$ images/sec in at least $A_J = 90\%$ of the time. At each iteration, we find one path for a GR application using Algorithm 2. In the ith iteration, found paths in the set ϕ has rates $\{r_1, r_2, r_3, ..., r_i\}$. In this case, the subset-sum problem is solved to find all the subsets of this set which sum up to at least the required processing rate of the GR application (set ϕ_R). After finding all subsets, we need to find the probability that the paths in the set is working, while all other paths are failed, which is not straightforward when considering the overlaps of paths. Therefore, the min-rate availability of the GR application is

$$\sum_{s \in \phi_R} \mathbb{P}\big(p \in s \text{ working } \& \ p \in (\phi \setminus s) \text{ failed}\big). \tag{7}$$

If this value is less than the requested application availability, we need to add more paths and repeat this process until the required availability of the GR application is satisfied. If the requested QoE of the application cannot be met, the application is rejected.

Note that to obtain more than one task assignment path for an application, Algorithm 2 should be repeated with updated available capacities for NCPs and links. For instance, suppose that for an application J Algorithm 2 is used to find the first task assignment path ($\{y_{i,j}\}_{i \in \mathbb{C} \cup \mathbb{T}, j \in \mathbb{N} \cup \mathbb{L}}$) with the processing rate $r_1 = \min_{\substack{r \in \mathbb{R} \\ j \in \mathbb{N} \cup \mathbb{L}}} \frac{C_j^{(r)}}{\sum_{i'} y_{i',j} a_{i'}^{(r)}}$. Then, to find the second task assignment path for this application, we update the available capacities of NCPs and links by subtracting the resources that the first assignment path took. Thus, the resource type r available capacity of NCP or link j would be $C_j^{(r)} - r_1 \sum_{i'} y_{i',j} a_{i'}^{(r)}$, which is used in Algorithm 2 in the next iteration. This process can be repeated to find an arbitrary number of task assignment paths for each application, until the requested QoE is met or the application is rejected.

It is worth noting that considering polynomial-time task assignment algorithm, limited maximum number of task assignment paths for applications, and convexity of (4), the whole complexity of SPARCLE algorithm is polynomial-time.

V. Performance Evaluation

In this section, we present both experimental and simulation results of deploying SPARCLE in dispersed computing networks with various topologies and subtask graphs. For performance comparison with SPARCLE, we implement the following state-of-the-art algorithms.

(i) T-Storm [29]. This algorithm places tasks on NCPs so as to minimize the added inter-node traffic, but unlike SPARCLE, it does not consider heterogeneous resource capacities.

(ii) VNE [12]. This uses a topology-aware node ranking algorithm to rank the nodes in the computing network and CTs of the application. Unlike our problem, the resource requirements of each virtual network element in VNE is fixed.

(iii) GS/GR. In the Greedy Sorted (GS) and Greedy Random (GR) algorithms, we use a similar placement algorithm as SPARCLE, but the CTs' placement is based on their resource requirements and randomly, respectively, not considering the connecting TTs' resource requirements.

(iv) HEFT [27]. This algorithm uses the heuristic greedy algorithm to place CTs on NCPs. Tasks are given a priority based on their upward rank and then placed on their host NCPs, which will result in the earliest finish time of each task.

(v) Random. In this task assignment, the CTs of application are assigned randomly on NCPs of the network.

A. Experimental Results

Our testbed models the dispersed computing network shown in Figure 4. The assumed network parameters are reported in Table I. We have used a real image stream processing application using OpenCV [6] in our experimental evaluation. This proof-of-concept experiment demonstrates the improved performance of dispersed computing using SPARCLE compared to cloud computing as well as the superiority of SPARCLE compared to other state-of-the-art algorithms.

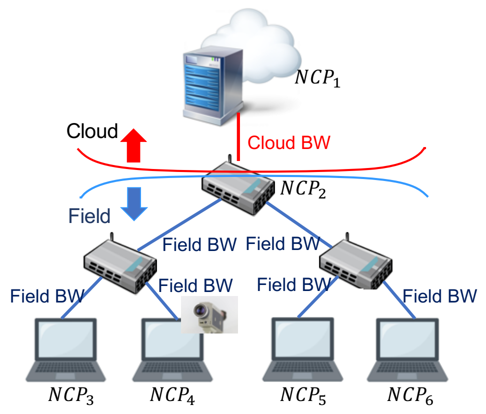


Fig. 4. Experimental testbed for dispersed computing network.

TABLE I
DISPERSED COMPUTING NETWORK PARAMETERS.

Network element	Capacity
Cloud CPU	4*3.8 (GHz)
Field CPU	3000 (MHz)
Cloud BW	100 (Mbps)

For large scale evaluation, we conduct an experiment using Mininet [2]. With Mininet, we can create a virtual network, which runs the real kernel, switch, and application code, on a single machine [2]. Our emulator first reads the experiment scenario file describing NCPs and their CPU capacities, links, and their bandwidths, routing paths, and the CT/TT requirements. Then it creates the virtual network according to the scenario, runs the experiment, and reports the performance results. We use the network topology shown in Figure 4. We use a face detection application with a task graph and task parameters shown in Figure 5 and Table II, respectively. For the TTs, we utilize the Linux `scp` command. The processing rate results using different task assignment algorithms along with cloud computing-based processing rates are depicted in Figure 6. We observe that SPARCLE can follow the optimal task assignments (which is found by exhaustive search) in all tested cases. We also find that SPARCLE outperforms existing algorithms such as HEFT, T-Storm, and VNE algorithms as well as cloud computing The processing rate achieved by SPARCLE can have about 300%, 63%, and 1350% improvement compared to the HEFT, T-Storm, and VNE algorithms, respectively. This improvement becomes dramatic when the field bandwidth is limited, as neither HEFT nor T-Storm jointly considers link and NCP resources.

Consistent with what is expected, when the field bandwidth is limited (e.g., 0.5 Mbps), the achieved processing rate by

TABLE II
FACE DETECTION APPLICATION PARAMETERS.

Task	Resource requirement
resize	9880 (MC/image)
denoise	12800 (MC/image)
edge detection	4826 (MC/image)
face detection	5658 (MC/image)
raw image transport	3.1 (MB/image)
resized image transport	182 (kB/image)
denoised image transport	145 (kB/image)
edge map transport	188 (kB/image)
detected faces transport	11 (kB/image)

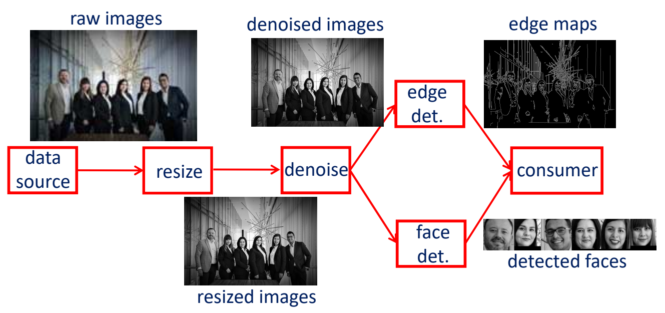


Fig. 5. The real face detection application for experimentation.

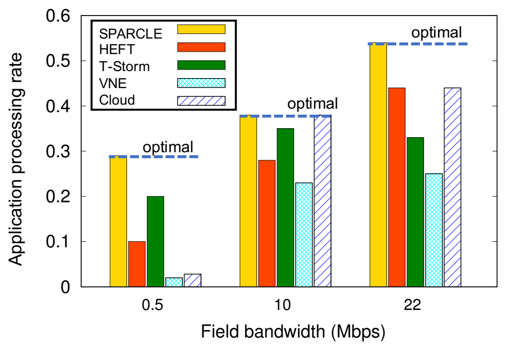


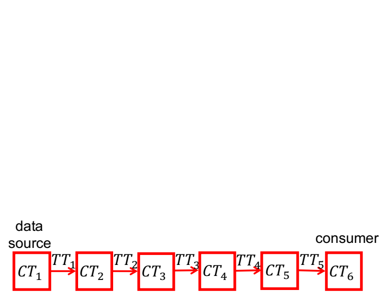
Fig. 6. Experimental results for face detection application's processing rate. Although better performance is expected for dispersed computing scheme in limited field bandwidths compared to cloud computing, the SPARCLE-based dispersed computing can increase by 23% in high bandwidths (e.g., 22 Mbps).

using dispersed computing is about 9 times better than the cloud computing's achieved processing rate. If we increase the field bandwidth to 10 Mbps, SPARCLE only uses the cloud, which is the optimal choice. Interestingly, if we keep increasing the field bandwidth (e.g., 22 Mbps), SPARCLE-based dispersed computing can still be beneficial with the improvement of 23% in the processing rate, compared to the cloud computing scenario.

B. Simulation Results

1) Setup: We consider two task graphs, a linear task graph and a diamond task graph, which are commonly used in the literature and real industry applications [22]. Figure 7 depicts those task graphs. We also use three different topologies for the dispersed computing network (star, linear, and fully connected) which are consistent with typical IoT scenarios [32].

For extensive comparison of different algorithms, we consider three cases: link-bottleneck case, NCP-bottleneck case, and balanced case. In the link-bottleneck case, the connecting links of the computing network have very limited communication resources compared to the resource requirements of the TTs, but the NCPs have a 10x larger ratio of available resource capacity to the CTs' computation resource requirements. So, the computing network links are the bottleneck in the application processing rate. Conversely, in the NCP-bottleneck case, the NCPs are the bottleneck: they have very limited computation resources compared to the computation resource requirements of the CTs, but the link bandwidths are sufficient for the applications' TTs. Finally, in the balanced case, both NCPs and links can be the bottleneck in the


(a) A linear task graph.

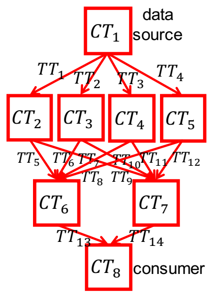
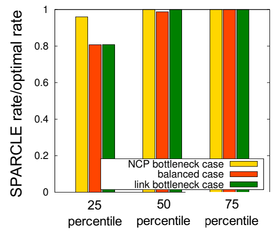

(b) A diamond task graph.

Fig. 7. Two commonly found task graphs in real applications.


(a) Linear network

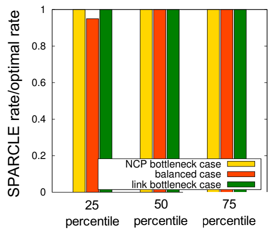

(b) Fully-connected network

Fig. 8. The 25, 50, and 75 percentiles of the application processing rate achieved by SPARCLE over optimal rate (including the linear task graph, fully-connected and linear network topologies). SPARCLE almost always finds the optimal rates.

application processing rate. To assess the effect of resource sharing between multiple applications, we initially consider assigning the tasks of only one stream processing application to the computing network and then report our results for hosting multiple stream processing applications.

2) One stream processing application: First, we suppose that there is only one stream processing application (consisting of multiple tasks) that needs to be placed on the computing network; thus, we only consider the task assignment as the application does not share resources. We find that the SPARCLE algorithm achieves a near-optimal processing rate with high energy efficiency, particularly in the link-bottleneck case.

Processing rates. Figure 8 depicts the 25, 50, and 75 percentiles of the ratio of SPARCLE's processing rate to the optimal processing rate obtained by exhaustive search for the three cases of NCP-bottleneck, balanced, and link-bottleneck. We observe that the SPARCLE task assignment algorithm can achieve very close performance to the optimal processing rate. Here, we have used a linear task graph with four CTs on linear and fully-connected network topologies.

Energy efficiency. We define the energy efficiency as the number of data units processed using a unit amount of energy. We use a realistic energy consumption model for different computing devices, based on previous works. The energy drain in smaller computing devices like a smartphone is mostly due to CPU, WiFi and cellular network transmissions, screen power, etc. The CPU energy consumption rate can be assumed to be proportional to CPU utilization [11], and the uplink and downlink data transmission energy drain rate over LTE or WiFi are proportional to the uplink and downlink data transmission rates, respectively [19].

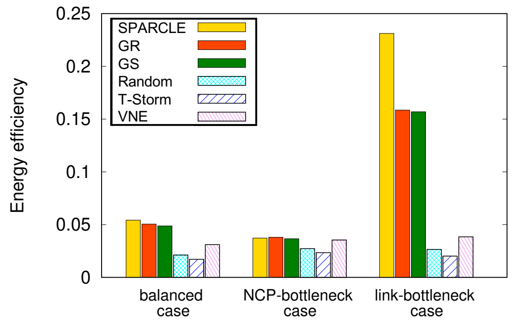
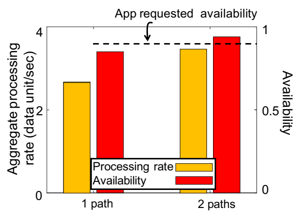


Fig. 9. Energy efficiency comparison in three different cases (Linear task graph, linear network topology). The average energy efficiency using SPARCLE can be improved by more than 53% compared to GS or GR algorithm.


(a) The availability and aggregate processing rate of a BE application. The application requested availability is provided with 2 paths.

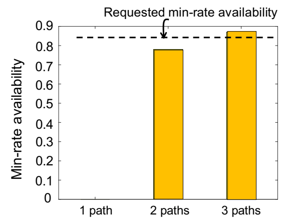
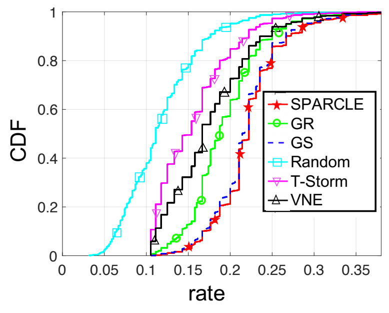

(b) The min-rate availability of a GR application with increasing number of paths. The application requested min-rate availability is with 3 paths.

Fig. 10. Availability of BE and min-rate availability of GR applications in presence of failure versus the number of task assignment paths.

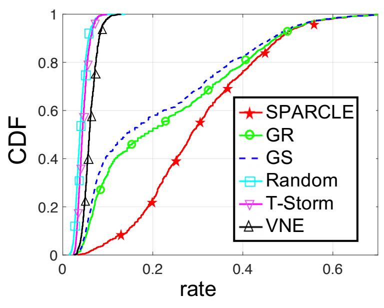
Figure 9 presents the average energy efficiency in the three different bottleneck scenarios. Although we do not specifically optimize for energy usage, SPARCLE improves the average energy efficiency compared to other algorithms. In the balanced case, the energy efficiency of SPARCLE is improved by about 126%, 190%, and 59% compared to the Random, T-Storm, and VNE algorithms, respectively. This dramatic improvement is due to the SPARCLE algorithm's giving a higher rank to CTs whose neighbors are already placed, especially in the link-bottleneck case, where the energy efficiency is improved by more than 53% compared to the GS or GR algorithm. Therefore, when the bandwidth is limited, these CTs will be placed first on their best NCP host, which is likely the same host as their neighbors (unless other CTs support a higher processing rate). Concentrating CTs on fewer NCPs reduces transmission energy and thus is generally better in terms of energy efficiency as well as latency.

QoE. For QoE evaluation, we consider the failure events of computing network elements (NCPs/links). In order to show how increasing the number of task assignment paths will help meet the requested QoEs of different BE and GR applications, we investigate two cases of BE and GR applications with a linear task graph. Here, we assume that the failure probability of links of the considered star computing network is 2%.

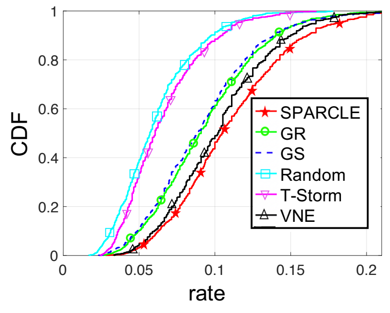
Figure 10(a) shows the aggregate processing rate of the BE application as well as the availability of the application with increasing number of paths. The availability of the application is 0.85 with a single task assignment path, which cannot satisfy the requested application availability of 0.9. However, adding



(a) The NCP-bottleneck case. The SPARCLE and the GS algorithms are equivalent in the NCP-bottleneck case.



(b) The link-bottleneck case. While the application processing rate achieved by the Random, T-Storm, and VNE algorithms are always less than 0.15, the achieved processing rate by SPARCLE is more than 0.15 in about 90% of the time.



(c) The balanced case. The average of the achieved processing rate by SPARCLE is about 82%, 69%, 22%, 17%, and 8% better than the random, T-Strom, GS, GR, and VNE task assignment algorithms, respectively.

Fig. 11. The CDF of the processing rate of one task assignment for different cases (Diamond task graph, star network topology).

one more path increases the application availability to 0.94, so SPARCLE lets the application be placed on two paths on the dispersed computing network.

Next, we consider a GR application with a requested minimum rate of 2.7 ($data\ units/sec$) and a min-rate availability of 0.85. The first path found by SPARCLE has a processing rate 2.67 ($data\ units/sec$), while the second path has a processing rate 1.2 ($data\ units/sec$). In order to satisfy the QoE, both of these paths should be working, with a probability of 0.78, which does not satisfy the requested min-rate availability. The third found path has a processing rate 0.42 ($data\ units/sec$). Thus, the first path and either one of the second or the third path should be working, in order to satisfy the min-rate availability. The increase of the min-rate availability versus the total number of task assignment paths is shown in Figure 10(b).

Diamond task graph. We now report the results for applications with diamond task graphs, as shown in Figure 7(b), and star dispersed computing networks with eight NCPs. Since the diamond task graph has more TTs and CTs than the linear one, the assignment problem is now more complicated; and we obtain different results for the three bottleneck cases. First, we assume there is only one resource type, e.g., CPU cycles per second for NCPs and bandwidth for links. The CDF of the maximum stable processing rate achieved by different algorithms for the NCP-bottleneck case is shown in Figure 11(a). As expected, in this bottleneck case, the SPARCLE and the GS algorithm have the same performance. That is, the bottleneck processing rate ($\gamma_{i,j}$ in Algorithm 2) depends only on NCP capacities, not on link capacities, and thus the SPARCLE algorithm will rank the CTs in the order of their resource requirements, just as the GS algorithm does.

For the link-bottleneck case, the CDF of the application processing rate achieved by different algorithms is depicted in Figure 11(b). While the application processing rate achieved by the Random, T-Storm, and VNE algorithms are always less than 0.15, our SPARCLE algorithm's achieved processing rate is more than 0.15 about 90% of the time. More importantly, the performance difference between SPARCLE, GR, and GS algorithms shows the effectiveness of the dynamic CT ranking algorithm used in SPARCLE. The average processing rate achieved by SPARCLE is increased by 30% compared to the GS algorithm, which uses the same task assignment algorithm, but orders the CTs by their resource requirements only, which shows the importance of considering relevant TTs in Sparcle.

Lastly, Figure 11(c) shows the CDF of the processing rate for the balanced case. In terms of the achieved processing rate, the improvement by SPARCLE is about 82%, 69%, 22%, 17%, and 8%, from the random, T-Storm, GS, GR, and VNE task assignment algorithms, respectively. From the result, we confirm that SPARCLE can assign the tasks of the application effectively using the dynamic ranking algorithm based on both available resources of NCPs and links.

The case with more resource types. We extend to the case where there are more than one computation resource type for CTs, e.g., CPU and memory requirements. Figure 12 depicts the 25 and 75 percentiles of the application processing rate using different task assignment algorithms in two cases of NCP memory-bottleneck and link-bottleneck. As shown, with more than one resource type, the performance of the GS and VNE algorithms is drastically degraded. SPARCLE instead uses a dynamic ranking algorithm, which takes into account all resource requirement types of CTs and TTs.

3) Multiple stream processing applications: Now we look at the scenario with multiple BE and GR stream processing applications being hosted in the computing network. Let's assume two BE applications with diamond task graphs that are placed on a star computing network with eight NCPs. We consider the balanced case and show the achieved objective function in (4) in Figure 13, when application 1 has higher priority ($P_1 = 2P_2$). The SPARCLE algorithm outperforms all of the baselines in this case, as well.

We evaluate the presence of multiple GR stream processing applications. Here, each GR application has either a diamond or a linear task graph, and a random requested processing rate. Figure 14 depicts the total processing rate of the admitted GR applications using different task assignment algorithms. As shown, the total processing rate of admitted applications is considerably increased using the SPARCLE algorithm for

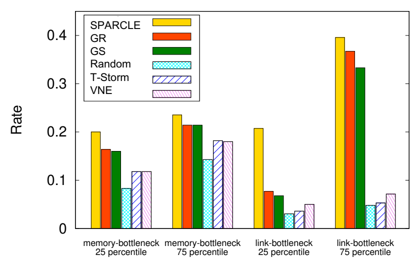


Fig. 12. The 25 and 75 percentile of the application processing rate for the case with multiple resource types (Diamond task graph, star network topology).

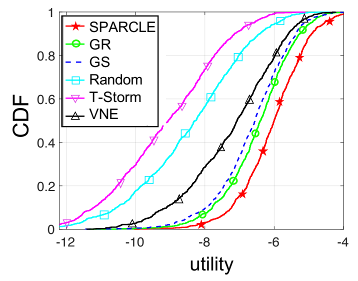


Fig. 13. The CDF of the utility in (4), where $P_1 = 2P_2$, achieved by different algorithms (Star network topology).

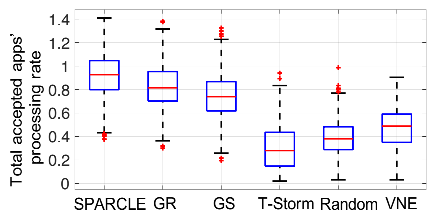


Fig. 14. The total processing rate of admitted GR applications. (Diamond and line task graphs, star network topology).

task assignment, indicating that more applications are admitted with the SPARCLE algorithm than with our benchmarks.

VI. Conclusion

In this paper, we presented SPARCLE, a scheduling system framework consisting of polynomial-time dynamic ranking task assignment and resource allocation algorithms for stream processing applications in dispersed computing networks. In particular, we showed that optimally assigning tasks to elements in dispersed computing networks and allocating resources to be shared among multiple stream processing applications are both challenging, non-trivial generalizations of classical optimization problems. Based on these generalizations, we devised a new polynomial-time algorithm to solve these problems. Through extensive evaluations and simulations, we demonstrated that SPARCLE outperforms other state-of-the-art algorithms in terms of both achievable processing rate and energy efficiency. Considering computing network resource fluctuation is our future work.

References

[1] Scheduler Algorithm in Kubernetes. https://github.com/eBay/Kubernetes/blob/master/docs/devel/scheduler_algorithm.md, 2015.

[2] Mininet. http://mininet.org/, 2018.

[3] Apache Flink. https://flink.apache.org, 2019.

[4] Apache Spark. https://spark.apache.org, 2019.

[5] Apache Storm. https://storm.apache.org/index.html, 2019.

[6] Open Source Computer Vision Library. https://opencv.org/, 2019.

[7] SPARCLE: Stream Processing Applications over Dispersed Computing Networks. https://tinyurl.com/ufjfcf4, 2020.

[8] M. Bramson. *Stability of Queueing Networks*. Springer, 2008.

[9] V. Cardellini, V. Grassi, F. L. Presti, and M. Nardelli. Optimal operator placement for distributed stream processing applications. In *Proceedings of the ACM International Conference on Distributed and Event-Based Systems*, 2016.

[10] N. Chen, Y. Chen, Y. You, H. Ling, P. Liang, and R. Zimmermann. Dynamic urban surveillance video stream processing using fog computing. In *Proceedings of IEEE International Conference on Multimedia Big Data (BigMM)*, 2016.

[11] X. Chen, N. Ding, A. Jindal, Y. C. Hu, M. Gupta, and R. Vannithamby. Smartphone energy drain in the wild: Analysis and implications. In *Proceedings of ACM SIGMETRICS*, 2015.

[12] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang. Virtual network embedding through topology-aware node ranking. *ACM SIGCOMM Computer Communication Review*, 41(2):38–47, Apr. 2011.

[13] G. Chochlidakis and V. Friderikos. Mobility aware virtual network embedding. *IEEE Transactions on Mobile Computing*, 16(5):1343–1356, May 2017.

[14] T. Das, Y. Zhong, I. Stoica, and S. Shenker. Adaptive stream processing using dynamic batch sizing. In *Proceedings of ACM Symposium on Cloud Computing*, 2014.

[15] A. Eryilmaz and R. Srikant. Joint congestion control, routing, and mac for stability and fairness in wireless networks. *IEEE Journal on Selected Areas in Communications*, 24(8):1514–1524, 2006.

[16] R. C. Fernandez, M. Migliavacca, E. Kalyvianaki, and P. Pietzuch. Integrating scale out and fault tolerance in stream processing using operator state management. In *Proceedings of the ACM SIGMOD*, 2013.

[17] L. Ghalami and G. Daniel. Scheduling parallel identical machines to minimize makespan:a parallel approximation algorithm. *Journal of Parallel and Distributed Computing*, 2018.

[18] R. L. Graham. Bounds on multiprocessing timing anomalies. In *SIAM Journal on Applied Mathematics*, volume 17, pages 416–429, Mar. 1969.

[19] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. A close examination of performance and power characteristics of 4g lte networks. In *Proceedings of ACM MobiSys*, 2012.

[20] J. Kwak, Y. Kim, J. Lee, and S. Chong. Dream: Dynamic resource and task allocation for energy minimization in mobile cloud systems. *IEEE Journal on Selected Areas in Communications*, 33(12):2510–2523, Dec. 2015.

[21] N. Ogino, T. Kitahara, S. Arakawa, and M. Murata. Virtual network embedding with multiple priority classes sharing substrate resources. *Computer Networks*, 112:52–66, 2017.

[22] B. Peng, M. Hosseini, Z. Hong, R. Farivar, and R. Campbell. R-storm: Resource-aware scheduling in storm. In *Proceedings of the Annual Middleware Conference*, 2015.

[23] L. Pu, X. Chen, J. Xu, and X. Fu. D2d fogging: An energy-efficient and incentive-aware task offloading framework via network-assisted d2d collaboration. *IEEE Journal on Selected Areas in Communications*, 34(12):3887–3901, Dec. 2016.

[24] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng. A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*, 2016(1):67, May 2016.

[25] E. G. Renart, J. Diaz-Montes, and M. Parashar. Data-driven stream processing at the edge. In *Proceedings of IEEE International Conference on Fog and Edge Computing (ICFEC)*, 2017.

[26] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura. Serendipity: Enabling remote computing among intermittently connected mobile devices. In *Proceedings of ACM MobiHoc*, 2012.

[27] H. Topcuoglu, S. Hariri, and M. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, Mar. 2002.

[28] J. Wang, L. Li, S. H. Low, and J. C. Doyle. Cross-layer optimization in tcp/ip networks. *IEEE/ACM Transactions on Networking (TON)*, 13(3):582–595, 2005.

[29] J. Xu, Z. Chen, J. Tang, and S. Su. T-storm: Traffic-aware online scheduling in storm. In *Proceedings of IEEE ICDCS*, 2014.

[30] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan. A framework for partitioning and execution of data stream applications in mobile cloud computing. *SIGMETRICS Perform. Eval. Rev.*, 40(4):23–32, Apr. 2013.

[31] S. Yang. Iot stream processing and analytics in the fog. *IEEE Communications Magazine*, 55(8):21–27, Aug. 2017.

[32] I. Yaqoob, E. Ahmed, I. A. T. Hashem, A. I. A. Ahmed, A. Gani, M. Imran, and M. Guizani. Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges. *IEEE Wireless Communications*, 24(3):10–16, Jun. 2017.

[33] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica. Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters. In *Proceedings of USENIX HotCloud*, 2012.