Data Distribution and Scheduling for Distributed Analytics Tasks

Stephen Pasteris*, Shiqiang Wang†, Christian Makaya†, Kevin Chan‡ and Mark Herbster*
*Department of Computer Science, University College London, UK, Email: {s.pasteris, m.herbster}@cs.ucl.ac.uk
†IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, Email: {wangshiq, cmakaya}@us.ibm.com
‡Army Research Laboratory, Adelphi, MD, USA, Email: kevin.s.chan.civ@mail.mil

***Abstract*—We consider a distributed edge computing system where we have a number of interconnected machines with limited communication bandwidth and storage capacity. Analytics tasks run on the machines, where each task runs on a single machine but may require data from multiple other machines. Every task requires a given amount of data to run, and it needs to receive all its data within a specific deadline. The application scenario is that each machine has limited storage, thus we usually cannot place the entire amount of data for a specific task on a single machine that executes the task. We assume that the task execution is sparse in time, so that at most one task is executed in the system at any time. The problem we study in this paper is how to distribute the data on machines in the system, without violating the bandwidth and storage constraints, while ensuring that the data transfer deadlines are met. We prove that the optimal solution to this problem is equivalent to that of a max-flow problem on a specifically constructed graph. We present how to construct this graph so that the problem can be solved using standard algorithms for max-flow problems, and also provide some numerical results and further discussions.**

***Keywords*-Data placement, Internet of Things (IoT), maximum flow problem, mobile edge computing, optimization**

I. Introduction

The emergence of the Internet of Things (IoT) [1] gives rise to a large number of computational devices widely distributed in the physical world. This set of connected devices collect a huge amount of data in the physical world, which enables a range of new applications that will improve people's daily lives. One of such applications is distributed analytics based on collaborative sensing [2], where a person can send a query to a cloud platform, and the cloud answers this query using available data and analytics code. An example of such a distributed analytics application can be the identification and tracking of people and objects. For example, Bob sees someone on the street who appears to be a criminal suspect but is unsure about whether it is indeed the same person. In this case, Bob can take a picture of the person and analyze it using distributed analytics.

One key challenge of distributed analytics applications is how to distribute the analytics code and data in such a distributed scenario where data might be coming from everywhere, while taking into account constraints such as limited storage and processing time. Various work have started in this direction and there is a clear need for enhancing the existing architectures and algorithms to fulfill new requirements. There are two common architectures that can be used for distributed analytics.

The centralized architecture is the most common today. It takes advantage of existing (centralized) cloud or cluster resources. With this architecture, data is moved to a centralized location with abundant computing and storage resources, where powerful analytics are applied to analyze the data. The main shortcoming of this approach is the need of moving data to a centralized place. If the data sources are counted in millions, sending data to a central location requires very high bandwidth (because the volume of the data to analyze is very large) and may lead to slow response and synchronization issues amongst data sources.

In the distributed architecture, the computational devices are located everywhere and any device can use the data. The computations and analytics are moved closer to the data sources, for example, to an edge server or even to a mobile device. This architecture is of great interest in IoT applications, because the analytics models can be built on the IoT device or close to it. It avoids issues associated with the centralized architecture, as the distributed architecture does not require sending large amounts of data across the network. The computations and analytics models run in a distributed manner. The data is kept close to the edge and the analytics models are built by coordinating data from multiple sources. The collection of all devices in the distributed architecture form up a distributed cloud platform with usually a limited amount of computational power, which is also known as an edge cloud [3].

In a distributed edge computing environment, it is often impractical to store all the code and data at a single machine, because all the devices in such an environment have limited storage and computational capacity, and the analytics models can be large and often consist of both code and data. For example, in face recognition models, a database stores a large amount of face features for individual persons. Therefore, once the cloud receives Bob's analytics request, it can process it in two different ways. One way is to send the relevant code and data from different cloud nodes to Bob, so that Bob can perform the analysis on its own device. Bob has to delete the code and data after processing due to its limited storage capacity. Another way is to send the picture of the suspect to all the related analytics components for processing. In both ways, data transfer is needed to perform Bob's analytics task.

One natural question to ask is how to distribute the analytics code and data, so that whenever a request comes in, the related code and data can be fetched in an efficient way such that the request can be completed within a limited amount of time. We study this problem in this paper. We first present the problem formulation, then show that the problem is equivalent to a max-flow problem on a graph constructed in a specific way. The solution to the original problem can be derived from the solution to this max-flow problem.

II. Problem Formulation

For simplicity, we do not distinguish between code and data in this paper, and we refer to both of them as data or information. We provide an abstract description of the problem in the following.

We have a complete directed graph on a set of vertices V. The vertices of the graph represent machines that can store and process data. The machines are connected by communication pipelines (edges) that carry data. Each edge (i, j) has a weight $B(i, j)$ which represents the bandwidth of the communication pipeline from i to j, measured in bits per second. If there is no communication pipeline between i and j, then $B(i, j) = 0$. The bandwidth is measured in bits per second and is the maximum rate that information can be sent between two machines. We assume that every communication pipeline has zero latency. Each machine i has a weight S_i which represents the maximum amount of information (called the "storage capacity") that can be stored in machine i. If a machine i is a router without storage capability, then we have $S_i = 0$. We are given a set W of "tasks". Every task k has an associated machine $a(k)$ and a weight D_k. Each task k represents an application that is run on machine $a(k)$ and requires D_k bits of information to run. With every task k we are given a maximum time, T_k, in which we need to send all D_k bits of information to machine $a(k)$. The problem is to find a distribution of the data across all machines that satisfies this constraint, given that no two tasks are called at the same time.

A. Information Streams

A propagation of information around the network is formally defined as an "information stream" which is a function $f : V \times V \to \mathbb{R}^+$, where $\mathbb{R}^+$ denotes the set of non-negative real numbers, satisfying $f(i, j) \leq B(i, j)$ for all machines i and j. Intuitively, the value $f(i, j)$ is the rate of information being transmitted from machine i to machine j, and the constraints mean that the rate of information being transmitted from one machine to another is no more than the bandwidth of the communication pipeline between them.

Given an information stream f and a machine i, the "emission rate" $l_f(i)$ is defined as $l_f(i) := \sum_{j \neq i} f(i, j) - \sum_{j \neq i} f(j, i)$, which is the amount of information leaving the machine minus the amount of information entering the machine. Intuitively, the emission rate is the rate of information being sent from a machine (if the emission rate is negative then the machine is receiving information).

Given a machine i, an "i-targeted information stream" is an information stream f satisfying $l_f(i) \leq 0$ and $l_f(j) \geq 0$ for all $j \neq i$. Intuitively an i-targeted information stream is an information stream in which machine i receives information and all other machines send information. Let Ω_i be the set of all i-targeted information streams.

B. Data Distribution and Task Time

Let W be the set of tasks. Given a task k, let D_k be the quantity of data that it requires and let $a(k)$ be the machine that it is run on. Our algorithm will find a distribution of the data across the network, which is a function $d : W \times V \to \mathbb{R}^+$ where $d(k, i)$ is the amount of data for task k which is stored at machine i. This implies the following constraints: $\sum_i d(k, i) = D_k$, which means the total amount of data for task k stored across the network is equal to D_k; and $\sum_k d(k, i) \leq S_i$, which means the total amount of data stored at a machine is no more than the storage capacity at that machine.

Given a task k, and an $a(k)$-targeted information stream $f_k \in \Omega_{a(k)}$, the "task time" is equal to $\max_{i \in V \setminus \{a(k)\}} d(k, i)/l_{f_k}(i)$ which is the maximum of time for the required data to leave a given machine and is hence the time taken to propagate the data to machine $a(k)$ (as the communication pipelines have zero latency).

C. The Problem

With each task we are given the machine it runs on, $a(k)$, the quantity of data that it requires, D_k, as well as a maximum acceptable task time T_k. With each machine i we are given a storage capacity S_i. The problem is to find a distribution $d(\cdot, \cdot)$ (as defined in the above section), and $a(k)$-information streams such that the task time of every task is no greater than its maximum acceptable task time. Formally, this problem is as follows:

Find a function $d : W \times V \to \mathbb{R}^+$ and, for every $k \in W$, an $a(k)$-targeted information stream $f_k \in \Omega_{a(k)}$ such that the following constraints are met:

1) For all $k \in W$, $\sum_{i \in V} d(k, i) = D_k$
2) For all $i \in V$, $\sum_{k \in W} d(k, i) \leq S_i$
3) For all $k \in W$, $\max_{i \in V \setminus \{a(k)\}} d(k, j)/l_{f_k}(i) \leq T_k$

III. Algorithm

We now outline the algorithm for solving the above problem. The algorithm works by converting the problem into a max-flow problem [4] as follows:

1) Construct a directed graph G (with capacities on edges) as follows (note that we will refer to vertices of the original graph as "machines" and vertices and edges of G as "G-vertices" and "G-edges"):
   a) For every machine i and every task k create a G-vertex $v(i, k)$.
   b) For every machine i create a G-vertex $w(i)$.

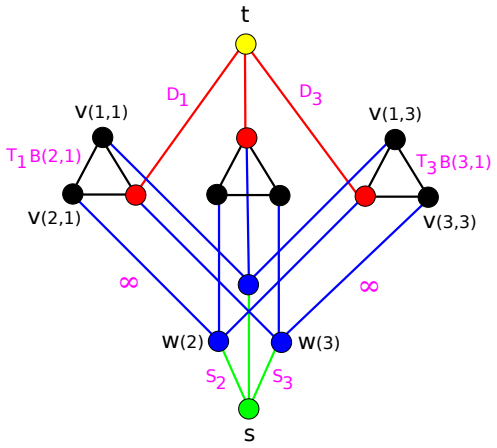

Figure 1. The graph G with $|V| = 3$ and $|W| = 3$. The black labels label G-vertices and the purple labels label G-edges. The black lines represent two directed G-edges (one in each direction) and the other lines represent a single directed G-edge which is directed upwards. The green and yellow G-vertices are the source and sink respectively. The blue G-vertices are $\{w(i) : i \in V\}$ and the black and red G-vertices are $\{v(i,k) : i \in V, k \in W\}$. The red G-vertices are $\{v(a(k),k) : k \in W\}$.

c) Create a source G-vertex s and a sink G-vertex t.
d) For all machines i and j and tasks k, add a G-edge from $v(i,k)$ to $v(j,k)$ with capacity $T_k B(i,j)$.
e) For all machines i add a G-edge from s to $w(i)$ with capacity S_i.
f) For all machines i and tasks k add a G-edge from $w(i)$ to $v(i,k)$ with infinite capacity.
g) For all tasks k add a G-edge from $v(a(k),k)$ to t with capacity D_k.

2) Find a maximum flow from s to t. Given G-vertices x and y, let F_{xy} be the flow from x to y.
3) If there exists a task k with $F_{v(a(k),k)t} < D_k$ then a required data distribution does not exist, i.e., the problem is not feasible. Else, the required data distribution is found by setting $d(k,i) := F_{w(i)v(i,k)}$ for all $i \in V$ and $k \in W$.

An example of graph G created based on the above procedure is shown in Fig. 1.

IV. Proof of Correctness

A. Equivalent Flows

Given a task k and an $a(k)$-targeted information stream $f_k \in \Omega_{a(k)}$ with task time at most T_k, we define the function $\hat{f}_k : V \times V \to \mathbb{R}^+$ as follows: Given machines i and j, let $\hat{f}_k$ be the total amount of information relevant to the task (i.e., that stored in the machines) which passes from i to j when the information for task k is transferred to $a(k)$ via the information stream f_k. For all $i,j \in V$, we clearly have $\hat{f}_k(i,j) \le T_k B(i,j)$ as this is the maximum amount of information that can be passed from i to j in time T_k. For all $i \neq a(k)$ we have that $\sum_j \hat{f}_k(i,j) - \sum_j \hat{f}_k(j,i)$ is the total amount of relevant information sent from machine i which is equal to $d(k,i)$. We also have that $\sum_j \hat{f}_k(j,a(k)) - \sum_j \hat{f}_k(a(k),j)$ is the total amount of relevant information entering machine $a(k)$ which is equal to $D_k - d(k,a(k))$. So, to summarise we have that $\hat{f}_k$ satisfies the following conditions:

1) $\hat{f}_k(i,j) \le T_k B(i,j)$ for all $i,j \in V$
2) $\sum_j \hat{f}_k(i,j) - \sum_j \hat{f}_k(j,i) = d(k,i)$ for all $i \neq a(k)$
3) $\sum_j \hat{f}_k(j,a(k)) - \sum_j \hat{f}_k(a(k),j) = D_k - d(k,a(k))$

We now show the converse: that given an $\hat{f}_k$ satisfying the above conditions, there is an $a(k)$-information stream, f_k, with task time at most T_k. This is found by setting $f_k(i,j) := \hat{f}_k(i,j)/T_k$ for all $i,j \in V$ since, directly from the above respective conditions, we have:

1) $f_k(i,j) \le B(i,j)$ for all $i,j \in V$
2) $l_{f_k}(i) = d(k,i)/T_k \ge 0$ for all $i \neq a(k)$
3) $l_{f_k}(a(k)) = -(D_k - d(k,a(k)))/T_k \le 0$

so f_k is an $a(k)$-targeted information stream. Note that, by above, the task time is equal to $\max_{i \in V \setminus \{a(k)\}} d(k,i)/l_{f_k}(i) = T_k$. We have hence shown that an $a(k)$-targeted information stream with task time at most T_k exists if and only if an $\hat{f}_k$ exists that satisfies the above conditions.

B. Existence of Flow

We now turn to the weighted directed graph G that the algorithm constructs. We first show that, given a required distribution $d(\cdot,\cdot)$ exists, then any maximal flow F from s to t has $\sum_k F_{v(a(k),k)t} = \sum_k D_k$. First note that since the sum of the capacities of the G-edges entering t is equal to $\sum_k D_k$ the flow can never be greater than this value. Hence, all that is required is to construct a flow, F, with $\sum_k F_{v(a(k),k)t} = \sum_k D_k$. We now construct such a flow. First define $F_{w(i)v(i,k)} := d(k,i)$ for all $i \in V$ and $k \in W$. Define $F_{s,w(i)} := \sum_k d(k,i)$. Given a task k choose $\hat{f}_k$ as defined in the previous subsection. We then define $F_{v(i,k)v(j,k)} := \hat{f}_k(i,j)$ for all $i,j \in V$, and define $F_{v(a(k),k)t} := D_k$. We clearly have $\sum_k F_{v(a(k),k)t} = \sum_k D_k$. We now show that F satisfies the conditions of a flow:

1) For all $i \in V$:

$$\begin{aligned} & F_{sw(i)} - \sum_k F_{w(i)v(i,k)} \\ &= \sum_k d(k,i) - \sum_k F_{w(i)v(i,k)} \\ &= \sum_k d(k,i) - \sum_k d(k,i) \\ &= 0 \end{aligned}$$

2) For all $k \in W$ and $i \in V \setminus \{a(k)\}$:

$$\begin{aligned}
&F_{w(i)v(i,k)} + \sum_{j \neq i} F_{v(j,k)v(i,k)} - \sum_{j \neq i} F_{v(i,k)v(j,k)} \\
&= d(k,i) + \sum_{j \neq i} F_{v(j,k)v(i,k)} - \sum_{j \neq i} F_{v(i,k)v(j,k)} \\
&= d(k,i) + \left(\sum_{j \neq i} \hat{f}_k(j,i) - \sum_{i \neq j} \hat{f}_k(i,j) \right) \\
&= d(k,i) - d(k,i) \\
&= 0
\end{aligned}$$

3) For all $k \in W$:

$$\begin{aligned}
&F_{w(a(k))v(a(k),k)} + \sum_{j \neq a(k)} F_{v(j,k)v(a(k),k)} \\
&\quad - \sum_{j \neq a(k)} F_{v(a(k),k)v(j,k)} - F_{v(a(k),k)t} \\
&= d(k,a(k)) + \sum_{j \neq a(k)} F_{v(j,k)v(a(k),k)} \\
&\quad - \sum_{j \neq a(k)} F_{v(a(k),k)v(j,k)} - D_k \\
&= d(k,a(k)) - D_k \\
&\quad + \left(\sum_{j \neq a(k)} \hat{f}_k(j,a(k)) - \sum_{a(k) \neq j} \hat{f}_k(a(k),j) \right) \\
&= d(k,a(k)) - D_k - (d(k,a(k)) - D_k) \\
&= 0
\end{aligned}$$

4) For all $i \in V$:

$$F_{sw(i)} = \sum_k d(k,i) \leq S_i$$

5) For all $i,j \in V$ and $k \in W$:

$$F_{v(i,k)v(j,k)} = \hat{f}_k(i,j) \leq T_k B(i,j)$$

which proves the existence of such a flow.

C. Sufficiency of Flow

We now show that given any flow F from s to t in G, with $\sum_k F_{v(a(k),k)t} = \sum_k D_k$, we have a sufficient distribution $d(\cdot,\cdot)$ defined as $d(k,i) := F_{w(i)v(i,k)}$. First note that since $\sum_k F_{v(a(k),k)t} = \sum_k D_k$ which is the maximum amount of flow that can enter t, we must have that every G-edge $(v(a(k),k),t)$ is at full capacity: i.e., $F_{v(a(k),k)t} = D_k$ for all k.

Given $k \in W$, let G_k be the subgraph of G induced by the G-vertices $\{v(i,k) : i \in V\}$. Note that the only G-edges entering G_k are $\{(w(i),v(i,k)) : i \in V\}$ and the only G-edge exiting G_k is $(v(a(k),k),t)$. Since F is a flow, this implies that

$$\sum_i d(k,i) = \sum_i F_{w(i)v(i,k)} = F_{v(a(k),k)t} = D_k$$

as required.

Note that for all $i \in V$ we have, since F is a flow, that

$$S_i \geq F_{sw(i)} = \sum_k F_{w(i)v(i,k)} = \sum_k d(k,i)$$

as required.

For all $k \in W$ define $\hat{f}_k : V \times V \to \mathbb{R}^+$ as $\hat{f}_k(i,j) := F_{v(i,k)v(j,k)}$. We now show that the above conditions (in Section IV-A) for $\hat{f}_k$ are satisfied:

1) For all $i,j \in V$ we have

$$\hat{f}_k(i,j) = F_{v(i,k)v(j,k)} \leq T_k B(i,j)$$

2) For all $i \neq a(k)$ we have

$$F_{w(i),v(i,k)} + \sum_{j \neq i} F_{v(j,k),v(i,k)} = \sum_{j \neq i} F_{v(i,k)v(j,k)}$$

so

$$d(k,i) + \sum_{j \neq i} \hat{f}(j,i) = \sum_{j \neq i} \hat{f}_k(i,j)$$

and hence

$$\sum_{j \neq i} \hat{f}_k(i,j) - \sum_{j \neq i} \hat{f}_k(j,i) = d(k,i)$$

3) We have

$$\begin{aligned}
&F_{w(a(k))v(a(k),k)} + \sum_{j \neq a(k)} F_{v(j,k)v(a(k),k)} \\
&\quad = \sum_{j \neq a(k)} F_{v(a(k),k)v(j,k)} + F_{v(a(k),k)t}
\end{aligned}$$

so

$$d(k,a(k)) + \sum_{j \neq a(k)} \hat{f}_k(j,a(k)) = \sum_{j \neq a(k)} \hat{f}_k(a(k),j) + D_k$$

and hence

$$\sum_{j \neq a(k)} \hat{f}_k(j,a(k)) - \sum_{j \neq a(k)} \hat{f}_k(a(k),j) = D_k - d(k,a(k))$$

which, as shown above, implies the existence of an $a(k)$-targeted information stream with task time T_k.

V. Numerical Results

We present some numerical results of the proposed data placement algorithm. We focus on the following aspects:

- The percentage of tasks for which not all the required constraints defined in Section II-C are satisfied, referred to as the "missing rate"
- The total amount of data that is required by all tasks, i.e., $\sum_{k \in W} D_k$
- The total amount of "missed" data, i.e., $\sum_{k \in W} \left(D_k - F_{v(a(k),k)t} \right)$

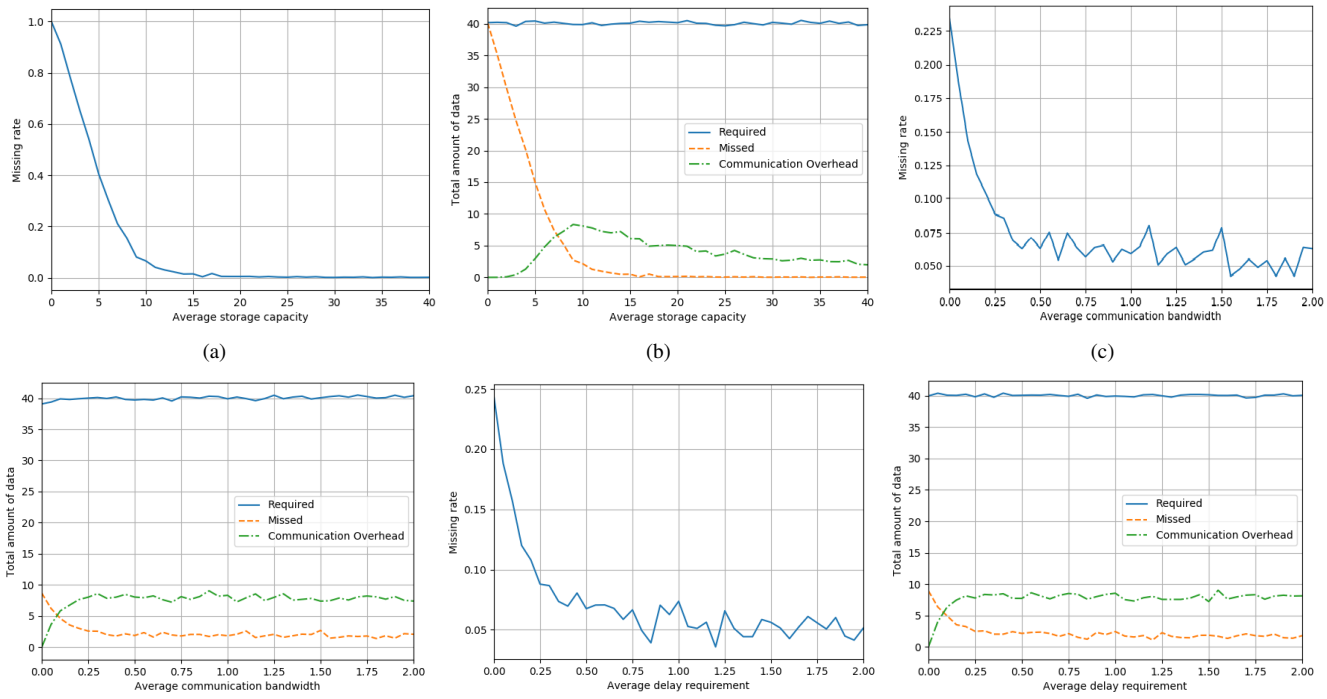

Figure 2. Simulation results: (a), (b): Performance under different average storage capacities $\overline{S}$, (c), (d): Performance under different average communication bandwidths $\overline{B}$, (e), (f): Performance under different average delay requirements $\overline{T}$.

- The sum amount of data that needs to be transferred over all communication pipelines for all tasks, referred to as the "communication overhead"

These performance indicators are evaluated under different storage capacities, communication bandwidths, and delay (task time) requirements. In the simulation, there are 5 machines and 40 tasks. For each task k and pair of machines i, j ($i \neq j$), the amount of required data D_k, storage capacity S_i, communication bandwidth $B(i,j)$, and delay requirement T_k are randomly generated according to a uniform distribution within $[0, 2\overline{D})$, $[0, 2\overline{S})$, $[0, 2\overline{B})$, $[0, 2\overline{T})$, respectively. Unless otherwise specified, the mean (average) values of the distributions are given by $\overline{D} = 1$, $\overline{S} = 10$, $\overline{B} = 1$, $\overline{T} = 1$. The results are shown in Fig. 2, where the average results of 200 independent simulation runs are plotted.

We see that, as expected, the missing rate and amount of missed data decrease when either one of $\overline{S}$, $\overline{B}$, and $\overline{T}$ increases. When $\overline{S}$ increases, the communication overhead first increases and then decreases. The increase in communication overhead is due to the increased amount of data that is stored in the system and sent to the task location (and thus a decreased missing rate). The decrease in communication overhead is because, when the storage capacity is large, more data can be stored locally at the machine where the task is located, and thus less data need to be transferred from remote locations. We also see that only increasing $\overline{B}$ or $\overline{T}$ cannot drive the missing rate down to zero. This is intuitive because, ultimately, the system needs to have enough capacity to store the data in order to guarantee a zero missing rate. The communication overhead does not change much with $\overline{B}$ and $\overline{T}$ after they are beyond a certain threshold, because the bandwidth and the delay requirement do not affect the data storage pattern when they are large enough and the storage capacity becomes the main bottleneck.

VI. Distributed Algorithms for Finding Max-Flow

The numerical results shown in the above section are obtained from the Edmonds-Karp algorithm [4], which is a centralized algorithm for finding the max-flow. In practice, when the system size is large, it is more practically feasible to find the max-flow in a distributed manner. While we leave the implementation and evaluation of distributed max-flow algorithms as future work, we present a literature review of related methods in this section.

Work on distributed implementation of max-flow algorithm which always requires sub-quadratic number of rounds is very limited in the literature [5], [6], [7]. The recent efforts to obtain fast algorithms to compute or approximate the max flow solutions focused on the undirected graph case.

In [7], the authors addressed the questions of how to run distributed protocols simultaneously as fast as possible and what are the limitations on how fast that can be done. A near-optimal distributed algorithm for $(1 + o(1))$-approximation of single-commodity maximum flow problem in undirected weighted networks that runs in $(D+n)n^{o(1)}$ communication rounds in the CONGEST [8] model is presented, where, n

and D denote the number of nodes and the network diameter, respectively. The proposed algorithms use randomization and succeed with high probability. The term protocol is referred for any distributed algorithm in the CONGEST model and problem being solved by each protocol is unknown. CONGEST is a distributed model that takes bandwidth limitations into account, i.e., it restricts the message size. Two variants of CONGEST have been proposed [8]: V-CONGEST and E-CONGEST. In V-CONGEST, the congestion is assumed on vertices and in each round, each node can send one $O(\log n)$-bit of the same message to all of its neighbors. However, in the E-CONGEST, i.e., congestion on edges, in each round, each node can send one $O(\log n)$-bit message to each of its neighbors and the model allows the node to send different messages to different neighbors.

In [9], a parallelized max-flow algorithm based on the Ford-Fulkerson [10] method on a cluster using the MapReduce framework is presented. The algorithm exploits the property that graphs are small-world networks with low diameter and employs optimizations to improve the effectiveness of MapReduce and increase parallelism. The experiments conducted in the paper show that it is possible to compute max-flow efficiently and effectively on very large real-world graphs (e.g., from Facebook) with billions of edges, although the best sequential max-flow algorithms have around quadratic runtime complexity. The work in [11] presents a two-stage distributed parallel algorithm (TSDPA), which is a distributed parallel algorithm with MPI for solving the max-flow problem. To improve the parallel performance, the strategy of TSDPA has two stages, which push excess flows separately along cheap and expensive paths identified by a new distance estimate function. The first strategy involves omitting high-cost paths, which aims at limiting the amount of communication. The second strategy entails the decentralization of computations. The work in [12] suggested that the parallelization of the max-flow algorithm is not theoretically guaranteed to be fast for every network. Hence, suitable partitioning methods, sequential max-flow algorithms, and distance functions for boundary nodes should be designed and tested for particular topological networks [11]. Also, hybrid parallel technologies, such as the use of distributed- and shared-memory modes in tandem, require further investigation [11] for solving max-flow problem.

VII. Conclusion

We have proposed an optimal algorithm to solve the data placement problem for distributed analytics tasks in an edge computing environment. The algorithm is based on converting the original problem to a max-flow problem on an auxiliary graph. The resulting max-flow problem can be solved using the Edmonds-Karp algorithm which has a strongly polynomial time complexity. This is an improvement over generic methods, such as solving the problem by means of linear programming, which is only known to guarantee weakly polynomial time complexity for general problem structures.

Acknowledgment

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] A. Mukherjee, H. S. Paul, S. Dey, and A. Banerjee, "ANGELS for distributed analytics in IoT," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*. IEEE, 2014, pp. 565–570.

[3] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *IFIP Networking Conference (IFIP Networking), 2015*. IEEE, 2015, pp. 1–9.

[4] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM*, vol. 19, no. 2, pp. 248–264, Apr. 1972.

[5] A. V. Goldber and S. Rao, "Beyond the flow decomposition barrier," *J. ACM*, vol. 45, no. 5, pp. 783–797, Sept. 1998.

[6] A. V. Goldber and R. E. Tarjan, "Efficient maximum flow algorithms," *Commun. ACM*, vol. 57, no. 8, pp. 82–89, Aug. 2014.

[7] M. Ghaffari, A. Karrenbauer, F. Kuhn, C. Lenzen, and B. Patt-Shamir, "Near-optimal scheduling of distributed algorithms," in *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, ser. PODC'15, 2015, pp. 81–90.

[8] D. Peleg, *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics.

[9] F. Halim, R. H. Yap, and Y. Wu, "A mapreduce-based maximum-flow algorithm for large small-world network graphs," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*. IEEE, 2011, pp. 192–202.

[10] L. Ford and D. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1956.

[11] J. Jiang and L. Wu, "Two-stage distributed parallel algorithm with message passing interface for maximum flow problem," *Journal of Supercomputing*, vol. 71, no. 2, pp. 629–647, 2015.

[12] P. Strandmark and F. Kahl, "Parallel and distributed graph cuts by dual decomposition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, San Francisco, CA, USA, 2010, pp. 2085–2092.