Data Stream Processing at Network Edges

Thanasis Loukopoulos*, Nikos Tziritas†, Maria Koziri*, George Stamoulis*,
Samee U. Khan‡§, Cheng-Zhong Xu † and Albert Y. Zomaya¶
*University of Thessaly, Lamia, Greece
†Chinese Academy of Sciences, Shenzhen, China
‡North Dakota State University, Fargo, USA
§National Science Foundation, Washington DC, USA
¶University of Sydney, Sydney, Australia

***Abstract*—This paper studies the problem of finding an assignment of data stream processing components onto servers under the objective to minimize both energy consumption and average delay experienced by end users within the system. The aforementioned problem is studied under the context of internet of things (IoT), where servers belong to micro-datacenters placed at network edges and thus close to data. We propose two algorithms to tackle the aforementioned problem taking into account limited server and network resources at micro-datacenters. The first algorithm is called Delay Aware Algorithm (DA) and results in a delay efficient assignment without taking into account energy consumption. The second algorithm is called Energy Efficient and Delay Aware Algorithm (EFDA) and results in a both energy and delay efficient assignment of application processing components onto servers within the system. We provide an experimental evaluation to show the behavior of the proposed algorithms against algorithms in the literature.**

I. Introduction

The rapid proliferation of computing devices dictates the adoption of internet of things (IoT) model, where the aforementioned devices can communicate with each other without human intervention. IoT environments is expected to generate huge amount of data where their analysis must take place in a real-time or an almost real-time manner. As a result, big data processing solutions will be of paramount importance in IoT environments [28]. To efficiently handle the sea of data generated within IoT environments, there is a dire need to push computing close to data [27]. Pushing computing close to data is called edge computing, which is formally defined as a method of performing data processing at the edges of the network. Edge computing is expected to play a pivotal role in IoT environments, where the amount of data grows rapidly in such environments. Therefore, pushing computing close to data is expected to result in lower bandwidth requirements as well as lower delays experienced by applications deployed in IoT systems.

Because IoT data are expected to overwhelm networks, edge computing dictates the aggregation and processing of those data in geographically distributed micro-datacenters. For instance, to leverage the aforementioned benefits of edge computing, micro-datacenters must be deployed at malls, hospitals, mobile base stations, etc. Below we give an example to show the benefit of adopting edge computing in mobile communication environments. By deploying micro-datacenters at the edge of the network of a mobile communication environment, i.e., inside mobile stations, mobile subscriber requests can be handled directly at the mobile station instead of forwarding them to a centralized entity (such as conventional cloud). In that way, mobile applications are expected to experience less delay against a conventional mobile communication system.

In this paper we study the problem of application assignment onto micro-datacenters. Each application is composed of a set of components handling either raw data or intermediate data produced by other components. The components work in a collaborative fashion to reach a common goal, which is the response to the corresponding application posed by some end-user. Components handling directly raw data perform just a simple transformation of those data to stream them towards data processing components. The location of such components is assumed fixed to the location of raw data. Application components consuming data streams must be assigned onto servers within the system by respecting network and server resource violations. The aforementioned assignment must take place in such a way such that the total average delay experienced by applications within the system as well as the overall energy consumption be minimized.

Our contributions are summarized as follows:

- We introduce the problem of minimizing energy consumption as well as the average delay experienced by end users of data stream applications at edge computing environments.
- We propose two heuristics resulting in a single solution. Specifically, the first heuristic tackles the problem by finding a delay aware placement of data processing components. On the other hand, the second heuristic tackles the problem by finding a both energy and delay efficient way.
- The algorithms take into account the current server and network utilization such that to decide the what is the average delay experienced by end-users.
- We provide an experimental evaluation to show the performance of the proposed algorithms against state-of-the-art algorithms.

The remainder of the paper is organized as follows. Section II presents the closest works found in the literature against our paper. Section III discusses the application model, the system model, and rigorously formulates the problem. The

delay aware algorithm as well as the energy efficient and delay aware algorithm are presented in Section IV and Section V, respectively. The experimental evaluation is discussed in Section VI; while Section VII concludes the paper and gives some future directions.

II. Related Work

Many works related with data stream processing has been researched over the past years. The reader can find a complete survey in [25]. Even though stream processing has been exhaustively studied in the past under various objectives and constraints, there is no work in the literature tackling the problem of minimizing energy consumption and delay regarding data stream applications submitted in edge computing environments.

Data stream processing has been extensively studied by the research community under the context of wireless sensor networks (WSNs) [8], [9], and [10] (to name a few). Specifically, the data stream processing problem is addressed in [9] to minimize the total network overhead within the system, with the application consisting of a single data processing component. The proposed algorithm is called dFNS, which is based on the Fermat node. The same objective with [9] is also targeted in [11]. The main difference between [9] and [11] is that the latter tackles the problem by considering multiple data processing components, with the components being organized as a binary tree. MCFA is proposed by [7] to also minimize the total network overhead in an optimal way for general tree-structured application graphs. The data stream processing problem is addressed by [12] and [13] to reduce network traffic. The authors in [10] propose power aware algorithms to decide which part of the application is executed on the server and which on the hand-held device. An energy-driven approach is proposed in [14] for choosing the appropriate materialized views such that to reduce processing cost and energy consumption. Lastly, authors in [15] address the application placement problem to minimize network traffic under capacity constraints on the WSN nodes. R-Storm [26] addresses the problem of deciding which parts of the application will be assigned onto which servers to minimize delay experienced by end-users in the context of cloud computing systems. All the aforementioned works (except R-Storm) resemble with our work in that the assignment of raw data is fixed, while the assignment of data processing components must be decided to optimize a specific objective function. The difference between R-Storm and ours is that the former assumes that raw data assignment is not fixed.

Our problem is also close to the agent placement problem, whereby the agent application graph consists of general purpose agents and node specific agents. Specifically, the general purpose agents can be hosted by any processing node within the system, since they demand only general-purpose resources. On the other extreme, the node specific agents have some specific resource requirements that can be met only by a subset of processing nodes within the system. POBICOS [16] is an agent-based platform making a clear definition of the agent placement problem. POBICOS allows general-purpose agents to migrate within the system, while node-specific agents are immobile. Therefore, an operator and a data source play the role of the general-purpose and node-specific agents, respectively. The problem of agent placement problem is tackled in [17] in an optimal and fully distributed way to minimize the total network overhead. Distributed online algorithms along with their competitive ratios are proposed in [18] to address the aforementioned problem. Authors in [19] propose heuristics to minimize energy consumption as well as network traffic under storage capacity constraints for the agent placement problem.

The virtual machine (VM) placement problem resembles also to our problem. Specifically, in VM placement problem, VMs communicate with each other as well as they demand to obtain data from data sources that are immobile within the system. The most common objective when tackling the VM placement problem is to minimize energy consumption and service level agreement (SLA) violations. The authors in [6] explore solutions to minimize the Service Level Agreement (SLA) violations experienced within a cloud due to the workload consolidation strategies. The authors propose a workload consolidation algorithm that modifies the Best Fit Decreasing (BFD) approach [23]. The proposed algorithm is named Modified Best Fit Decreasing (MBFD) and it works as follows. The MBFD sorts all of the VMs in the decreasing order, keyed by the computing capacity, and allocates each of the VM to a host that provides the least increase in the power consumption due to such an allocation. The VM placement problem is addressed in [20], whereby the objective is to minimize the network congestion within the system. The dynamic service placement problem is tackled in [21], with the objective being to reduce the hosting cost over time according to both demand and resource price fluctuation. Lastly, [22] tackles jointly the problem of both minimizing energy consumption and network traffic under storage capacity constraints. The closest work to our problem is that of MBFD where VMs are assigned such that to minimize energy consumption.

Unfortunately, to the best of our knowledge we do not know any paper in the literature to tackle the problem of minimizing energy consumption and average delay experienced by end-users regarding data stream applications submitted in edge computing systems.

III. Application Model, System Model, and Problem Formulation

A. Application Model

An application consists of a topology representing the data processing graph. The data processing graph provides a logic view of how data flow across the processing components and how they are processed on them. Before proceeding to the explanation of the data processing graph we give the following definitions:

- **Tuple** is a sequence of a predefined number of elements.
- **Stream** is an unbounded sequence of tuples.

- **Component** is an element within the application (more details are given later on).
- **Selectivity** of a given component is defined as the ratio between the number of input tuples and the number of output tuples. For example, consider a filtering component with its selectivity equalling to 0.5. The above signifies that when the number of input tuples equal 100, then the number of output tuples must equal 50.

The data processing graph consists of three types of components:

- **Spout** is a data stream generator, where it generates an unbounded number of tuples that must be processed by some components belonging to the application. For example, a spout can be a program monitoring a server within a system and generates logs in the form of tuples that must be processed by one or more applications (e.g., an application performing intrusion detection).
- **Bolt** is a component taking as input one or more data streams, which performs some processing on the input data streams and then generate an output data stream. The input data streams of a bolt are tuples emitted by either spouts or other bolts. Operations performed by a bolt may be filtering, aggregation, joins, etc.
- **Sink** is the component receiving the final data stream of a given application.

For the definition of spouts and bolts the reader is also referred to [26].

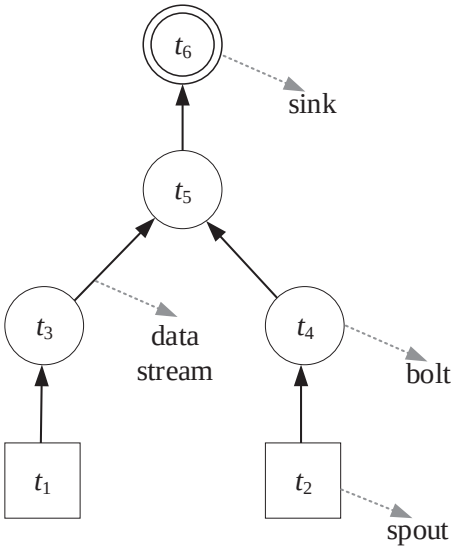

Fig. 1. Data processing graph.

The data processing graph is structured in a hierarchical way. Specifically, D spouts reside at the bottom level, B bolts reside at the intermediate levels, while one or more sinks(e.g., a mobile users) are located at the highest level of the application. A spout is depicted as a rectangle, while a bolt as a circle as shown in Figure 1. It must be noted that edges are directed and represent data transferred from one application component towards another one. Let t_i denote an application graph node, with the index i reflecting: (a) a data source if $0 < i \leq D$; (b) a bolt when $D < i \leq D + B$; and (c) a sink when $D + B < i \leq D + B + K$. Each component of a given applications has zero or more input streams as well as at most one output stream. Specifically, a spout has zero input streams, while a single output stream. A bolt has at most one input streams and a single output stream, while a sink has a single input stream while zero output streams. The set of input streams as well as the set of output streams of a component t_i is defined by IS_i and OS_i, respectively. Note that a set of input (output, resp.) streams for a given component t_i is defined by the set of component ids that forward (receive, resp.) tuples towards (from, resp.) t_i. A spout can emit tuples directed only towards bolts. A bolt can transfer tuples towards either another bolt or a sink. The number of tuples received from t_i per time unit is defined by tp_i^{in} and expressed by Eq. (1). On the other hand, the number of tuples emitted by component t_i per time unit within the system is captured by tp_i^{out}. When t_i represents a spout, then tp_i^{out} is fixed and provided by the type of spout. When t_i represents a bolt, then tp_i^{out} is calculated as per Eq. (2). Last, when t_i represents a sink, then it must hold that $tp_i^{out} = 0$. Let r_i be the computing resources required by t_i to process a tuple (measured in number of cycles per second). The selectivity of t_i is expressed by sel_i. The volume of a tuple produced by t_i is captured by v_i.

$$tp_i^{in} = \sum_{\forall t_j \in IS_i} tp_j^{out} \tag{1}$$

$$tp_i^{out} = tp_i^{in} \cdot sel_i, D < i \leq D + B \tag{2}$$

In Fig. 1 we show an example of an application consisting of six components $\{t_1, t_2, t_3, t_4, t_5, t_6\}$. Specifically, t_1 and t_2 play the role of spout, t_3, t_4, and t_5 play the role of bolt, while t_6 plays the role of sink. Bolt t_5 has two input data streams originating from t_3 and t_4, therefore it holds that $IS_5 = \{t_3, t_4\}$. On the other hand, there is a single output stream regarding t_5 towards t_6, and thus it holds that $OS_5 = \{t_6\}$.

B. System Model

As discussed previously, MEC dictates the location of services at base stations as shown in Figure 2. Base stations are equipped with micro datacenters which can be envisioned as a set of servers connected to a switch. Base stations are connected to each other through either wireless communication links or optical links. Each active server is accompanied with its current utilization level and depicted in black colour; otherwise it is depicted in grey colour.

The network consists of M micro-datacenters, with DC_x signifying the x-th micro-datacenter within the system. Each micro datacenter consists of a number of servers. The total number of servers within the system equals N, with the j-th server being denoted by s_j. The computing capacity of s_j is expressed by c_j and measured in number of cycles per second. A server s_j reaches its maximum power consumption (denoted

by P_j^{max}) when its utilization (denoted by u_j) is at 100%, while it reaches its minimum power consumption (denoted by P_j^{min}) when it is idle.

The location of i-th server servers within x-th micro datacenter is encoded by setting $h(i)$ equal to x. On the other hand, $h(j)$ equals i when t_j is assigned onto s_i (i.e., when $A_{ij} = 1$). BW is an $M \times M$ matrix capturing the bandwidth capacity between micro datacenters. BW_{xy} represents the bandwidth capacity between DC_x and DC_y. When BW_{xy} is equal to zero, we assume that there is no direct link between DC_x and DC_y. It must be noted that BW_{xx} represents the internal link bandwidth capacity of DC_x. Let $path_{xy}$ represent the path employed for the communication between DC_x and DC_y. Note that $path_{xx}$ represents the internal network link of DC_x, while $path_{xy}$ consists of the links required to establish a minimum delay connection between DC_x and DC_y. In case there are more than one minimum delay paths, then we choose randomly. Let $Z_{ij}(x, y)$ be a boolean variable denoting whether the communication between s_i and s_j employ the link between DC_x and DC_y.

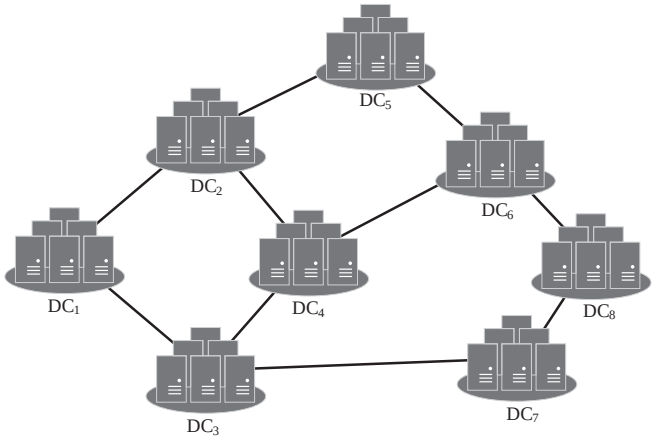

Fig. 2. Network of datacenters.

An example of a system consisting of eight micro-datacenters is shown in Fig. 2. Consider that all inter-datacenter links have the same bandwidth (let's say 10 bytes per second), while the intra-datacenter network bandwidth equals 100 bytes per second. then $path_{1,7}$ consists of two links: (a) the link between DC_1 and DC_3, and (b) the link between DC_3 and DC_7. Assuming the utilization of links is zero, then the delay of transferring one byte from a server hosted by DC_1 to a server hosted by DC_7 is equal to $\frac{1}{10} + \frac{1}{10} = 0.2$. On the other hand, the delay to transfer one byte between two different servers hosted by DC_1 is equal to $\frac{1}{100} = 0.01$, under the assumption that the utilization of DC_1's intra-datacenter network bandwidth equals zero. Below we introduce the average delay to measure the average time needed to transfer one byte between two servers when the utilization of network links is not zero.

C. Problem Formulation

We assume that the placement of sinks and spouts is fixed. The assignment of application components onto the servers within the system is captured by an $(D + B + K) \times N$ matrix named A. A_{ij} is a boolean variable encoding whether t_i is assigned onto s_j or not. Eq. (4) enforces the constraint that the computing capacity of a server cannot exceeded by the aggregated computing resources of components assigned onto server in question. On the other hand, by Eq. (5) we enforce that each bolt is assigned onto a single server. The utilization of s_j is expressed by Eq. (3). The average tuple processing delay of component t_i is captured by Eq. (6), given an assignment A (for more details the reader is referred to [2]).

Before proceeding to the average delay to transfer one byte over two servers, we define the utilization of a link between two micro datacenters DC_x and DC_y. The utilization of the link between DC_x and DC_y is defined as the total amount of data traversing across that link divided by the bandwidth capacity of that link. The aforementioned is expressed by Eq. (7). The average delay of transferring one byte over a path between two servers s_i and s_j is equal to zero when $i = j$, expressed by Eq. (8a). The above is because when two components connected by a data link are co-located on the same server, then the data transfer between those components takes place by copying the data onto memory of the server in question. On the other hand, when $i \neq k$, the average delay of transferring one byte across a path between two datacenters $h(i)$ and $h(j)$ is equal to the maximum average delay to transfer one byte among the links contained on $path_{xy}$. The aforementioned is expressed by Eq. (8b). The total delay of incurred within the system for a tuple to arrive on and be processed by component t_i is equal to the delay of processing a tuple at t_i plus the maximum delay for a tuple to arrive on t_i among the input streams of t_i, expressed by Eq. (9). The average delay for a tuple to arrive on t_i from t_j is equal to the total delay for a tuple to arrive on and be processed by t_j (i.e.,$dl^{tot}(A, t_j)$) plus the average delay to transfer the tuple between the servers hosting t_i and t_j (i.e., $v_j \cdot dl^{net}(A, s_{h(i))}, s_{h(j)})$). The total delay incurred within the system is captured by Eq. (10), which is actually the accumulated delays for a given tuple to reach each sink within the system.

$$u_j(A) = \frac{\sum_{i=1}^{D+B+K} A_{ij} \cdot r_i \cdot tp_i^{in}}{c_j} \tag{3}$$

$$\sum_{i=1}^{D+B+K} A_{ij} \cdot r_i \leq c_j, 1 \leq j \leq N \tag{4}$$

$$\sum_{j=1}^{N} A_{ij} = 1, P < i \leq P + B \tag{5}$$

$$dl^{pr}(A, t_i) = \frac{\frac{r_i}{c_{h(i)}}}{1 - u_{h(i)}} \tag{6}$$

$$u_{xy} = \frac{\sum\limits_{\forall i} \sum\limits_{\forall t_j \in IS_i} tp_j^{out} \cdot v_j \cdot z_{h(i),h(j)}(x,y)}{BW_{xy}} \quad (7)$$

$$dl^{net}(A, s_i, s_j) = \begin{cases} 0, i = j & (8a) \\ \max\limits_{l \in path_{h(i),h(j)}} \frac{\frac{1}{BW_l}}{1 - u_l}, i \neq j & (8b) \end{cases}$$

$$dl^{tot}(A, t_i) = dl^{pr}(A, t_i) + \max_{t_j \in IS_i} \left\{ v_j \cdot dl^{net}(A, s_{h(i)}, s_{h(j)}) + dl^{tot}(A, t_j) \right\} \quad (9)$$

$$dl_{sys}(A) = \sum_{D+B<i\leq D+B+K} dl^{tot}(A, t_i) \quad (10)$$

The instant power consumption of a server s_j at point in time t is expressed by Eq. (11). The first component of Eq. (11) represents the static power, which is constant for a commissioned server, while the second component reflects the power consumed when varying servers utilization ignoring the static power. According to [1] the second component of Eq. (11) is a linear function of utilization. The total energy consumption incurred within the system over a period T is captured by Eq. (12).

$$P_j = P_j^{min} + u_j \cdot (P_j^{max} - P_j^{min}) \quad (11)$$

$$En_{sys}(A) = \sum_{j=1}^{N} P_j \cdot T \quad (12)$$

The problem can formally be stated as follows: *Given a fixed assignment of spouts and sinks, find an assignment A for bolts such that such that the total system delay, expressed by Eq. (10), as well as the total energy consumption, expressed by Eq. (12), be minimized. The aforementioned must respect the constraints expressed by Eq. (4) and Eq. (5).*

IV. Delay Aware Algorithm (DA)

The objective of DA is to minimize the total delay experienced by sinks within the system. Initially, the algorithm decides the placement of each application within the system as follows. DA starts from the bottom level bolts of a given application and proceeds gradually to the upper levels. When deciding the placement of a bolt t_i onto a server s_j, DA evaluates the decision placement of a bolt through Eq. (9). Each time an application is accepted the network links' and servers' utilization is recomputed. In case a component of an application cannot be assigned onto any server due to server processing capacity and/or network link bandwidth violation, the corresponding application is rejected. After finishing with the placement decisions of all bolts within the system, DA records both the total delay experienced by sinks as well as the total energy consumption within the system. The pseudocode of the algorithm is shown in Algorithm **??**.

Algorithm 1 Delay Aware Algorithm

```
1: procedure DA(AppQueue)        ▷ AppQueue contains all applications
2:     while AppQueue ≠ ∅ do
3:         App← AppQueue.remove()
4:         SCHEDULE(App)
5:     end while
6: end procedure
7: procedure SCHEDULE(App)
8:     place App bolts into BQueue in a bottom-up fashion
9:     while BQueue ≠ ∅ do
10:        t_i ← BQueue.remove()
11:        choose A_ij = 1 such that dl^tot(A, t_i) is maximized
12:        reject App if t_i cannot be hosted by any server within the system due to network link bandwidth and/or server computing capacity violations
13:    end while
14: end procedure
```

V. Energy Efficient Delay Aware Algorithm (EFDA)

EFDA assigns applications as a whole to servers. The reason for doing this is that by assigning an application as a whole to a server, we can save energy consumption. The energy reduction is due to the fact that by employing the aforementioned way, there cannot exist an assignment where a server is assigned only a single component. Servers that are not assigned any component can be decommissioned and save thus energy. In case of DA, there may exist an assignment where a server is assigned a single component. Therefore, we expect that DA will result in higher energy consumption against EFDA. If there is no server that can host a whole application due to server computing capacity and network bandwidth limitations, then the corresponding application is rejected. In case there are more than one servers that can be assigned an application, then EFDA chooses the server such that the corresponding sink experiences the least average delay as per 9.

VI. Experimental Evaluation

A. setup

The experimental evaluation to investigate the behavior of the proposed algorithms was conducted by implementing our simulator in c++. Network topologies are modeled through BRITE [3]. It must be noted that the nodes of the aforementioned network topologies represent micro datacenters. Ten network topologies were generated, with the number of micro datacenters being fixed at 10. Each micro datacenter is assigned five servers. We employ the Cisco Server Power Calculator Tool [4] as well as [5] to generate a pool of servers. According to the above references, the aforementioned pool includes C-Series servers. The maximum power consumption of the aforementioned servers may range roughly between 80 to 1200 watts. Specifically, the least power consuming server is a server of type C220-LFF-M4 with its maximum

Algorithm 2 Energy Efficient and Delay Aware Algorithm

```
1: procedure PADA(AppQueue) ▷ AppQueue contains all applications
2:     while AppQueue ≠ ∅ do
3:         App← AppQueue.remove()
4:         SCHEDULE(App)
5:     end while
6: end procedure
7: procedure SCHEDULE(App)
8:     minDelay ← INF
9:     for each server s_i do
10:        curDelay ← average delay as per 9 when assigning App onto s_i
11:        if curDelay<minDelay then
12:            bestServer ← s_i
13:            minDelay ← curDelay
14:        end if
15:    end for
16:    if minDelay ≠ INF then
17:        bestServer ← App
18:    else
19:        reject App
20:    end if
21: end procedure
```

and idle power being 79.59 and 35.72 Watts, respectively. The aforementioned C-Series server is equipped with a processor Intel E5-2650L v4 1.7 GHz/65W 14C/35MB Cache, a memory of 8 GB DDR4-2400-MHz RDIMM/PC4-19200/single rank/x4/1.2v, a power supply 770W PSU, and a hard disk 400GB SSD SFF. On the other extreme, the maximum power consuming server is a server of type C480-M5 with its maximum and idle power being 1243.69 and 739.11 Watts, respectively. The aforementioned server is equipped with four processors of Intel 8180M 2.5 GHz/205W/28C/38.5MB Cache each, 4 memories of 32 GB DDR4-2666-MHz RDIMM/PC4-23100/dual rank/x4/1.2v each, two power supplies of 1600W PSU1 each, and two hard disks of 16TB 2.5-inch U.2 NVMe SSD (Intel P4500) each. The intra datacenter bandwidth is fixed to 10Gbps, while the bandwidth of inter-datacenter links is fixed to 1 Gbps.

We create an application pool of ten different types of applications. The smallest application type consists of two spouts, one bolt and one sink, while the biggest application type consists of six spouts, ten bolts and one sink. Therefore, for each plot we conduct 100 experiments. The output stream rate of a spout is uniformly distributed between 1000 and 5000 tuples. The size of a tuple is also uniformly distributed between 50 and 200 bytes. The selectivity of bolts is defined according to the needs of each experiment (see further down). The bolt processing requirements are uniformly distributed between 100 and 500 cycles.

The experimental evaluation of the proposed algorithms is conducted against MBFD [6] and MCFA [7].

B. Experiments

1) Varying the number of applications: The first experiment is conducted to investigate the behaviour of proposed algorithms when fixing the selectivity to one, while ranging the number of applications from 10 to 200. in Fig. 3, we show the normalized sum of the delays experienced by all sinks within the system. As can be seen DA achieves the best performance, with th next best candidate being EFDA. It is worth mentioning that both EFDA and DA are not sensitive when scaling the number of applications hosted by the system. On the other hand, MBFD and MCFA are both sensitive when increasing the number of applications within the system. The aforementioned can be seen at the transition of 100 to 200 applications hosted by the system.

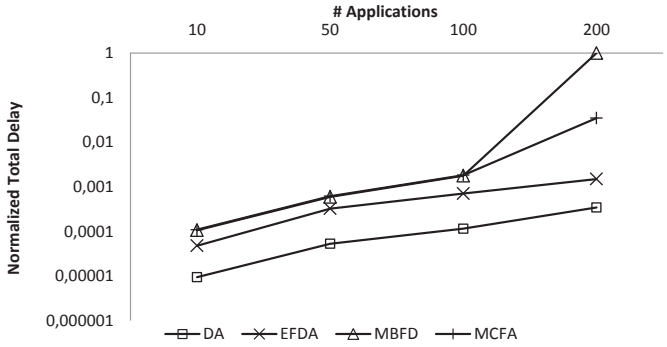

Fig. 3. Normalized total delay (Selectivity=1).

Fig. 4 shows the behaviour of the algorithms regarding the maximum delay achieved within the system among all sinks. The best results are achieved by DA. As observed, the performance of both DA and EFDA changes linearly or almost linearly with as per the number of applications hosted by the system. On the other hand, the maximum delay achieved by MCFA and MBFD increases almost exponentially when the system utilization increases (i.e., the number of applications increases).

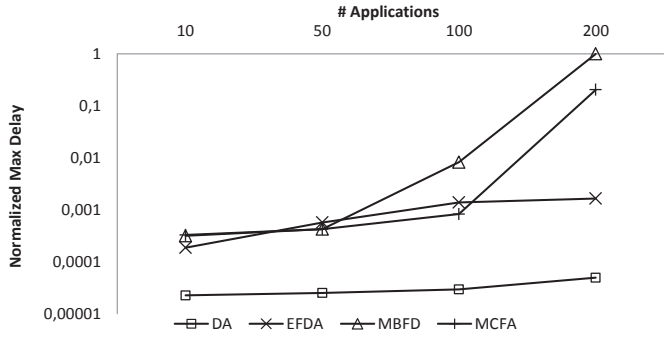

Fig. 4. Normalized maximum delay (Selelectivity=1).

Last, the energy consumption achieved by the algorithms is shown in Fig. 5. It is shown that the least energy consumption is achieved by MBFD, which was expected. The next best

performance is achieved by EFDA. On the other hand, DA and MCFA result in the maximum energy consumption for all scenarios, with the reason being that they are not power aware algorithms.

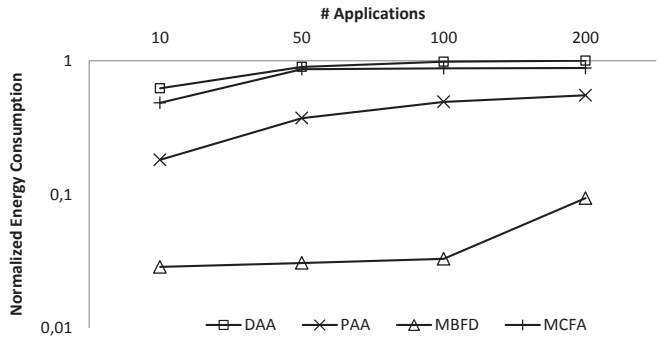

Fig. 5. Normalized energy consumption (Selectivity=1).

2) Varying selectivity: At second experiment, we fix the number of applications to 200 and vary selectivity between 0.4 and 1. In Fig. 6, we show the performance of algorithms regarding the total delay experienced by sinks within the system. We observe that selectivity affects mostly MBFD. Specifically, MBFD performance decreases as selectivity increases. The above is justified by the fact that, MBFD accumulates all applications onto a few servers, increasing both server and network utilization and thus affecting the delay experienced by sinks. The best performance is achieved by DA, while the second best by EFDA.

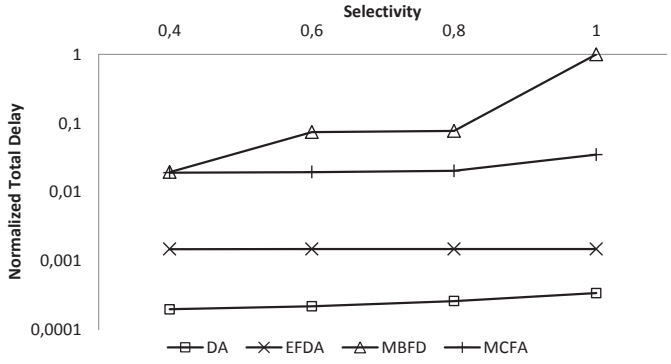

Fig. 6. Normalized total delay (Applications=200).

The maximum delay experienced among all sinks within the system (see Fig. 7) is also affected in the same way as in Fig. 6.

Last, from Fig. 8 we notice that the energy consumption is not affected when varying selectivity. The above is due to the fact that selectivity may affect network utilization but not server utilization.

3) Evaluating the bolt rejection ratio: The last experiment concerns the evaluation of how many bolts get rejected by each algorithm when varying the number of applications injected within the system while keeping selectivity fixed to 1. As

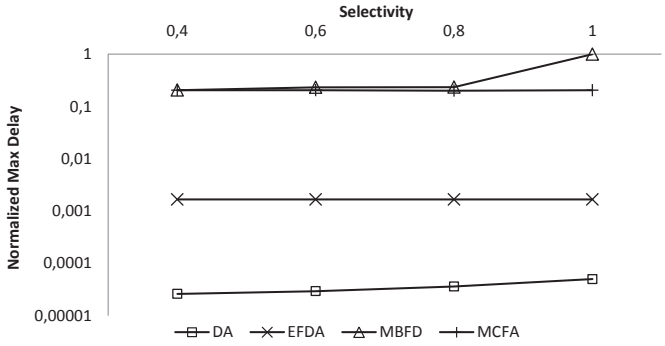

Fig. 7. Normalized maximum delay (Applications=200).

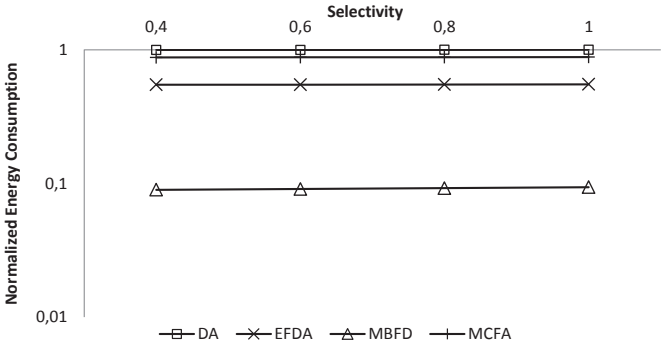

Fig. 8. Normalized energy consumption (Applications=200).

can be seen in 9, EFDA results in the least rejected bolts in all scenarios. On the other hand, MBFD results in the worst performance, which is justified by the fact that MBFD over-utilizes network resources, leaving no room to many applications to get accepted by the system.

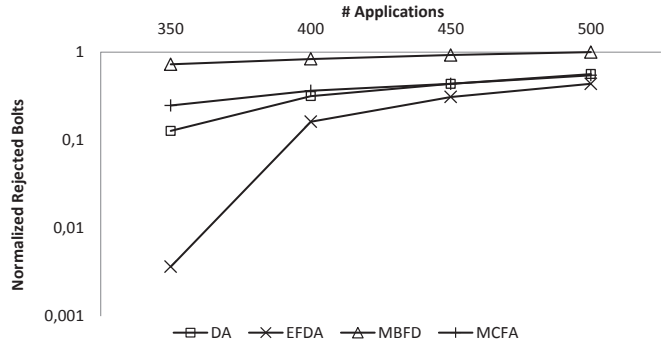

Fig. 9. Normalized Number of Rejected Bolts (Selectivity=1).

The number of bolts get rejected when fixing selectivity to 0.5 is depicted in Fig. 10. As seen, MCFA performs better than the rest algorithms, with DA following closely. The reason that MCFA and DA perform better than DA when selectivity is low can be justified as follows. When selectivity is low, then bolts perform compression to the data received by their input stream bolts or spouts. Therefore, by placing bolts as close to spouts

or input stream bolts, we reduce the amount of data transferred across the network and thus increasing the chances for more application to get accepted by the system.

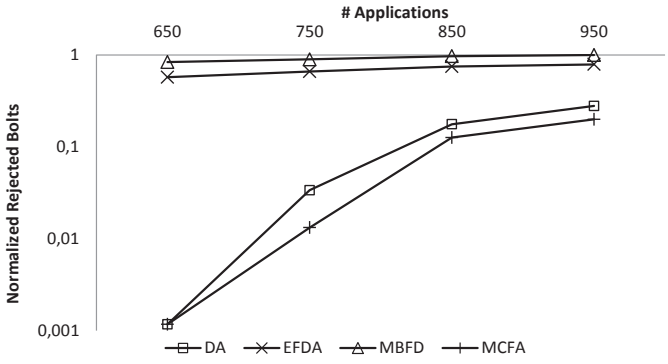

Fig. 10. Normalized Number of Rejected Bolts (Selectivity=0.5).

VII. Conclusions and Future Directions

In this paper we tackled the problem of minimizing average delay experienced by end users as well as energy consumption within an edge computing environment by taking the appropriate decisions for the assignment of application data processing components. The experimental evaluation showed that the proposed algorithms outperform MCFA and MBFD in most scenarios.

As future directions we intend (a) to tackle the problem by jointly minimizing energy consumption and latency experienced by mobile users through Pareto-efficient techniques; (b) to propose fully distributed solutions, other than the centralized ones; and (c) to tackle the problem in an online fashion.

Acknowledgment

Samee U. Khans work was supported by (while serving at) the National Science Foundation. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] R. Raghavendra, P. Ranganathan, V. Talvar, Z. Wang, X. Zhu, "No power struggles coordinated multi-level power management for the data center," *Thirteenth International Conference on Architectural Support Programming Languages and Operating Systems (ASPLOS)*, 2008, pp. 48-59.

[2] T. Repantis, X. Gu, and V. Kalogeraki, "QoS-aware shared component composition for distributed stream processing systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 20(5), 2007,pp. 968-982.

[3] A. Medina, A. Lakhina, I. Matta, J. Byer, "BRITE: an approach to universal topology generation," *Ninth International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2001, pp. 346-353.

[4] https://www.cisco.com/c/en/us/support/web/tools-catalog.html.

[5] H. Block, J. Beckett, K.-D. Lange, J. A. Arnold, S. Kounev, "Analysis of the influences on server power consumption and energy efficiency for CPU-intensive workloads," *Sixth ACM/SPEC International Conference on Performance Engineering*, 2015, pp. 223-234.

[6] A. Beloglazov, J. Abawajy, R. Buyya, "Energy-aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing," *In Journal Future Generation Computing Systems, Vol 28(5)*, 2012, pp. 755-768.

[7] Z. Lu, Y. Wen, "Distributed and Asynchronous Solution to Operator Placement in Large Wireless Sensor Networks," *Eighth International Conference on Mobile Ad-hoc and Sensor Networks*, 2012.

[8] G. Chatzimilioudis, H. Hakkoymaz, N. Mamoulis, D. Gunopoulos, "Operator Placement for Snapshot Multi-predicate Queries in Wireless Sensor Networks," *IEEE International Conference on Mobile Data Management: Systems, Services and Middleware (MDM)*, 2009, pp. 21-30.

[9] G. Chatzimilioudis, A. Cuzzocrea, D. Gunopoulos, N. Mamoulis, "A novel distributed framework for optimizing query routing trees in wireless sensor networks via optimal operator placement," *Journal of Computer and System Sciences*, 2013, pp. 349-368.

[10] P. Neophytou, M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis, "Power-aware operator placement and broadcasting of continuous query results," *Ninth ACM International Workshop on Data Engineering for Wireless and Mobile Access*, 2010.

[11] Z. Lu, Y. Wen, R. Fan, S.-L. Tan, J. Biswas, "Toward Efficient Distributed Algorithms for In-Network Binary Operator Tree Placement in Wireless Sensor Networks," *IEEE Journal on Selected Areas in Communications*, 2013, pp. 743-755.

[12] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, M. Seltzer, "Network-aware operator placement for stream-processing systems," *22nd International Conference on Data Engineering (ICDE)*, 2006.

[13] S. Rizou, F. Durr, K. Rothermel, "Solving the multi-operator placement problem in large-scale operator networks," *19th International Conference on Computer Communications and Networks (ICCN)*, 2010.

[14] A. Roukh, L. Bellatreche, A. Boukorca, S. Bouarar, "Eco-DMW: Eco-design methodology for data warehouses," *Eighteenth ACM International Workshop on Data Warehousing and OLAP*, 2015.

[15] N. Tziritas, T. Loukopoulos, S. U. Khan, and C.-Z. Xu, "Distributed Algorithms for the Operator Placement Problem," *IEEE Transactions on Computational Social Systems*, 2016.

[16] N. Tziritas, G. Georgakoudis, S. Lalis, T. Paczesny, J. Domaszewicz, P. Lampsas, T. Loukopoulos, "Middleware mechanisms for agent mobility in wireless sensor networks", *International Conference on Sensor Systems and Software (S-CUBE)*, 2012.

[17] N. Tziritas, S. U. Khan, T. Loukopoulos, S. Lalis, C.-Z. Xu, P. Lampsas, "Single and Group Agent Migration: Algorithms, Bounds, and Optimality Issues," *IEEE Transactions on Computers*, 2014, pp. 3143-3161.

[18] N. Tziritas, S. Lalis, S. U. Khan, T. Loukopoulos, C.-Z. Xu, and P. Lampas, "Distributed online algorithms for the agent migration problem in WSNs," *ACM/Springer Mobile Networks and Applications*, 2013, pp. 622-638.

[19] N. Tziritas, T. Loukopoulos, S. Lalis, P. Lampsas, "Algorithms for energy-driven agent placement in wireless embedded systems with memory constraints," *Simulation Modelling Practice and Theory*, 2011, pp. 1445-14464.

[20] J. W. Jiang, T. Lan, S. Ha, M. Chen, M. Chiang, "Joint VM placement and routing for data center traffic engineering," *IEEE International Conference on Computer Communications (INFOCOM)*, 2012.

[21] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, J. L. Hellerstein, "Dynamic service placement in geographically distributed clouds," *IEEE Journal on Selected Areas in Communications (JSAC)*, 2013, pp. 762-772.

[22] N. Tziritas, C.-Z. Xu, T. Loukopoulos, S. U. Khan, Z. Yu, "Application-aware Workload Consolidation to Minimize both Energy Consumption and Network Load in Cloud Environments," *IEEE International Conference on Parallel Processing (ICPP)*, 2013, pp. 449-457.

[23] S. Martello, P. Toth, "Knapsack Problems Algorithms and Computer Implementations," *John Wiley & Sons*, 1990.

[24] L. Amini, N. Jain, A. Sehgal, J. Silber, O. Verscheure, "Adaptive Control of Extreme-scale Stream Processing Systems," *26th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2006.

[25] M. D. Assuncao, A. S. Veith, R. Buyya, "Distributed Data Stream Processing and Edge Computing: A Survey on Resource Elasticity and Future Directions," *Journal of Network and Computer Applications*, Elsevier, A Paratre. ¡hal-01653842¿

[26] B. Peng, M. Hosseini, Z. Hong, R. Farivar, R. Campbell, "Rstorm: Resource-aware scheduling in storm," *16th ACM Annual Middleware Conference*, 2015.

[27] L. Atzori, A. Iera, G. Morabito, “The internet of things: A survey”, *Computer Networks*, vol. 54(15), 2010, pp. 27872805.
[28] M. D.!Assuncao, R. N. Calheiros, S. Bianchi, M. A. S. Netto, R. Buyya, “Big data computing and clouds: Trends and future directions”, *Journal of Parallel and Distributed Computing*, 2015 315.