Response Time Aware Operator Placement for Complex Event Processing in Edge Computing

Xinchen Cai[1], Hongyu Kuang[1], Hao Hu[1](✉), Wei Song[2], and Jian Lü[1]

[1] State Key Lab for Novel Software Technology, Nanjing University, Nanjing, Jiangsu, China
smldcxx@126.com, {khy,myou,lj}@nju.edu.cn

[2] School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, Jiangsu, China
wsong@njust.edu.cn

Abstract. A typical complex event processing (CEP) service is composed by a set of operators organized as a directed acyclic graph. This kind of service is usually used to handle large amounts of real-time data. Meanwhile, edge computing has been widely accepted as a new paradigm to improve the QoS of deployed services by making the services closer to the data. Thus, the response time, which is a crucial QoS metric for CEP services, can be significantly reduced by deploying CEP services on the edge network. However, it is often unlikely for a single node of the edge network to host all operators of a CEP service due to the limited computing resources. Therefore, it is desirable for a CEP service to place its operators on different nodes of the edge network to keep the response time low, especially when the input rate of the CEP service significantly increases. In this paper, we reduce the average response time of CEP services by deploying the operators on the edge nodes dynamically according to the predicted response time of CEP services. Specifically, we first propose a system model to capture the response time of the CEP services, based on which we formulate the problem of the optimal placement of CEP operators in the edge network. We then propose an algorithm that predicts the response time of CEP services and deploys the operators on the edge nodes with the minimum predicted delay. A simulation-based evaluation demonstrates that, compared with two state-of-the art algorithms, our algorithm can reduce the total response time by 33% and 45% on average, respectively.

Keywords: Complex event processing · Edge computing
Operator graph · Operator placement · Response time

C. Pahl et al. (Eds.): ICSOC 2018, LNCS 11236, pp. 264–278, 2018.
https://doi.org/10.1007/978-3-030-03596-9_18

1 Introduction

Typically, a complex event processing (CEP) service is used to handle large amounts of real-time data by using a set of operators organized as a directed acyclic graph called *operator graph*. In practice, CEP services are usually deployed on cloud data centers. However, this deployment results in delivering data from the edge of the network to remote cloud data centers and thus seriously reduces QoS of CEP services due to the long distance between data sources and remote cloud data centers, as well as the mass data transmission.

To address this issue, a new paradigm, called edge computing [12], has been proposed to improve the QoS of deployed services by moving services from cloud data centers to the edge of network. Consequently, a growing body of work [2,4,14] focuses on deploying CEP services on the edge network (i.e., nearest to the user [14]) to improve the QoS of CEP services, e.g., to reduce the response time. However, due to the limited computing capacity, lots of users queries, and large amounts of input events, congestion can easily occur at the nearest edge node and increase the response time of CEP services dramatically. Therefore, it is important to be aware of the response time of a CEP service during the placement of its operators on suitable nodes of the edge network to keep the response time low, especially when the input rate of the CEP service significantly increases. We use the following example to demonstrate this situation.

Consider the example shown in Fig. 1. This figure depicts the placement of a CEP service that queries traffic accident events, which are defined as a lane switch event after a decreased speed event. The CEP service has an operator graph consisting of a lane operator, a speed operator, and an accident operator. The edge network contains three edge nodes: v_0, v_1, and v_2. The resources capacity of the nodes v_0, v_1, and v_2 are 1, 2, and 2, respectively. The resources refers to computing resources, such as CPU, etc. Connected devices deliver and cache original data to the nearest node v_0 in the edge network. An intuitive strategy of reducing the response time is to deploy operators on the nearest edge node [14]. As Fig. 1 shows, the lane operator is deployed on the nearest edge node v_0. At this moment, v_0 has no resources to host other operators. Thus, the speed operator and the accident operator are deployed on the edge node v_1 which is the second nearest edge node except v_0. However, when the input rate of the CEP services largely increases, large amounts of data have to be delivered from the edge node v_0 to v_1. The congestion will occur and result in increasing the response time. On the other hand, another placement strategy is to deploy operators distributed in the edge network to balance the load between edge nodes in advance [6]; for example, deploying the accident operator to the edge node v_2 to reduce the load of v_1. When the input rate is high, this placement can reduce the response time compared to the first placement. However, when the input rate remains low, this placement increases the transmission delay between edge nodes and makes the response time even worse. In this paper, we argue that it is important to balance the load of different edge nodes during the placement of CEP operators by being aware of the response time of the operator graphs.

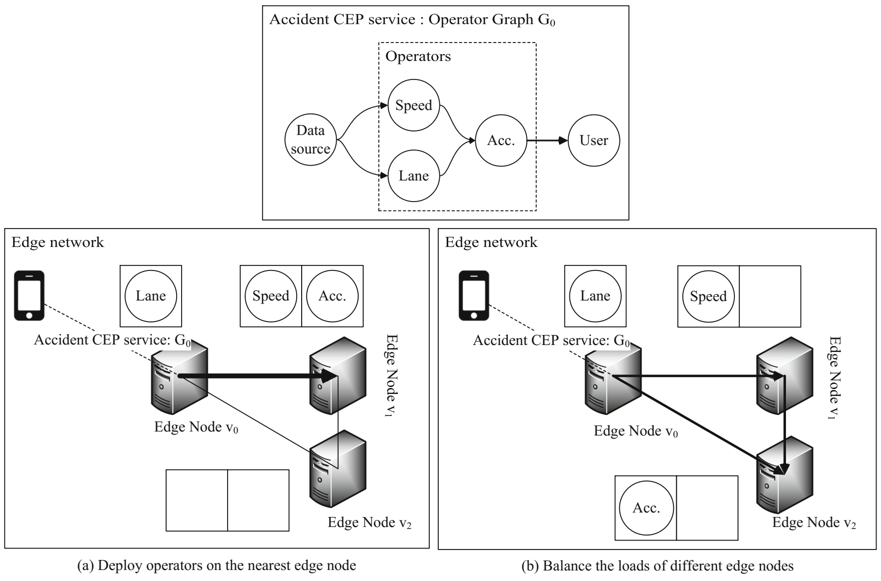

Accident CEP service : Operator Graph G_0
Operators
Data source
Speed
Lane
Acc.
User
Edge network
Lane
Speed
Acc.
Accident CEP service: G_0
Edge Node v_0
Edge Node v_1
Edge Node v_2
(a) Deploy operators on the nearest edge node
Edge network
Lane
Speed
Accident CEP service: G_0
Edge Node v_0
Edge Node v_1
Acc.
Edge Node v_2
(b) Balance the loads of different edge nodes

Fig. 1. Deploy operators of a CEP service that queries accident events on the edge network.

To achieve this goal, we first propose a combined model including CEP model and edge model to capture the response time of CEP services. Based on the models, we predict the delay of these edge nodes according to the monitored information. We then prove the optimal operator placement for CEP in edge computing is NP-hard. Finally, we propose a novel approximation-based algorithm that deploys the operators on the edge node with the lowest predicted delay. We generate 100 operator graphs and 30 edge networks for our simulation-based evaluation. The result demonstrates that our approach is able to maintain low response time when the input rate of CEP services significantly increases.

This paper makes the following three contributions: (1) We propose a system model to capture the response time of CEP services, and formulate a novel optimization problem, that is, to find an optimal placement of CEP operators in the edge network such that the response time of the CEP services is minimized. (2) We prove the optimal operator placement problem for CEP in edge computing is NP-hard. (3) We propose a novel response time aware operator placement algorithm that keeps the average response time of the operator graphs low.

The rest of the paper is structured as follows. Section 2 reviews related work. Section 3 describes the system models and problem formulation. Section 4 presents our response time aware operator placement algorithm. Section 5 reports the evaluation results, and finally, Sect. 6 concludes the paper.

2 Related Work

Operator placement problem has been widely studied in the last decades. The basic form of the *operator placement problem* is stated as: given a network of N nodes with some or all of them generating data processed by an operator, place the operator in the network so as to reduce the network traffic [16]. Several placement algorithms [2,10,11,16] have been proposed. These algorithms are characterized by different assumptions and optimization goals. Pietzuch et al. [10] design a stream-based overlay network for operator placement in distributed stream-processing (DSP) systems that minimizes the network usage of a query, while keeping the query delay low and picking nodes with adequate bandwidth. However, their operator model is not organized as operator graphs. Rizou et al. [11] present a distributed placement algorithm that minimizes the bandwidth-delay product of data streams between operators. However, their approach does not take the resources of the nodes into consider. Tziritas et al. [16] propose an approach enabling both single and group operator migrations using evictions of hosted operators. Although this work takes resources of nodes in the network into consider, it focuses on how to migrate operators to improve a placement of operators. Cardellini et al. [2] propose a general formulation of DSP placement in distributed network. This work focuses on the scenes that data sources are distributed in large scale network and do not take the input rate of CEP service into consideration. In contrast, we focus on deploying CEP services to process data near an edge node to infer meaningful events, especially when the input rate of CEP services significantly increases.

Edge computing is the computational infrastructures that make services closer to the end users [12]. The same concept is also called as cloudlet [13] or fog computing [1]. Taneja et al. [15] conduct experiments to demonstrate that edge computing can effectively improve its QoS. Hong et al. [4] proposed an opportunistic spatio-temporal CEP hosted by edge computing. As far as we know, this is the earliest work to deploy CEP on edge computing. How to offload applications to edge nodes [8,18] is another research interest that quite close to the operator placement problem in edge computing. The goal of this work is to minimize the cost of a user. However, our goal is to minimize the average response time of the CEP services queries. Saurez et al. [14] propose an incremental deployment approach to greedily deploy operators on the nearest edge node. However, they all do not consider the change of the input rate. Jia et al. propose an user to edge computing allocation in wireless metropolitan area networks [5] and a load balance approach to finding an optimal redirection of tasks between edge nodes [6]. However, the task model of this work has no graph structure. Our work is designed for CEP operator graphs whose input rate change according to the data sources.

3 Models and Problem Formulation

In this section, we first discuss our CEP model and edge computing model. Then we use these models to calculate the response time of CEP services, to give the

Table 1. Main notation adopted in the paper

Symbol	Description
$G_{edge} = (V_{edge}, E_{edge})$	The edge network G_{edge} consists of the edge nodes V_{edge} and the connections between edge nodes E_{edge}
v_i	The ith edge node in V_{edge}
(v_i, v_j)	The connection between the edge nodes v_i and v_j
$\mathrm{c}(v_i)$	The resources capacity of the edge node v_i
$\lambda(v_i)$	The execution rate of the edge node v_i
$\mathrm{w}(v_i, v_j)$	The transmission rate between edge nodes v_i and v_j
$G_{cep} = (\Omega, \mathrm{L})$	An operator graph consists of the operators Ω and the event streams L
$\mathrm{c}(\omega_i)$	The resources requirement of the operator ω_i
$T_r(X)$	The response time of the operator graph according to a placement X
T_p	The end-to-end delay of a path
$X(\omega_i) = v_u$	Deploy the operator ω_i on the edge node v_u
θ_t	The thread of response time to redeploy operator graphs
Δ_t	The time interval of replacement judgement

operator placement problem statement, and to prove this problem is NP-hard. Table 1 summarizes the symbols that are used in this paper.

3.1 CEP Model

We assume that N_{cep} CEP services queries are sent to an edge node. A CEP service can be represented as an operator graph $G_{cep} = (\Omega, L)$. Ω denotes the set of operators. L denotes the event streams between the data sources, the operators, and the consumer. Each operator $\omega_i \in \Omega$ has the attributes $\mathrm{c}(\omega_i)$, denoting the amount of resources required for the operator ω_i execution. The resources required to execute an operator include CPU, memory, etc. We simplified these resource models as unit resources, like other works [2,6].

For example, in Fig. 1, Ω contains the speed operator ω_{speed}, the lane operator ω_{lane}, and the accident operator $\omega_{accident}$. L contains (source, ω_{speed}), (source, ω_{lane}), (ω_{speed}, $\omega_{accident}$), (ω_{lane}, $\omega_{accident}$), and ($\omega_{accident}$, user). $\mathrm{c}(\omega_{speed})$, $\mathrm{c}(\omega_{lane})$, and $\mathrm{c}(\omega_{accident})$ are all one-resource units.

3.2 Edge Computing Model

Edge nodes are represented as vertexes V_{edge}. The network connections between edge nodes are represented as E_{edge}. The edge network are represented as $G_{edge} = (V_{edge}, E_{edge})$. Every node v_i in the edge network has two attributes: (1) $\mathrm{c}(v_i)$, the amount of resources available in the edge node v_i; (2) $\lambda(v_i)$, the events execution rate of the edge node v_i. Every network connection (v_i, v_j) has the attribute w(v_i, v_j), the transmission rate between the edge nodes v_i and v_j.

When a user sends a query to the nearest edge node for computing resources, the edge node manages the placement of the operator graph. This edge node is called as *manage edge node*. The manage edge node monitors the information of the nearby edge nodes within H hops. These nearby edge nodes are called *candidate edge nodes*. The operators can be placed on either the manage edge node or candidate edge nodes. Thus, the size of the edge network is limited by the number of hops H. H becomes larger when the operators have no suitable edge nodes to be placed.

3.3 Response Time

The response time of an operator graph G_{cep} can be calculated as:

$$T_r(G_{cep}) = \max_{p \in \pi_{G_{cep}}} T_p \tag{1}$$

where $\max_{p \in \pi_{G_{cep}}} T_p$ means the worst end-to-end delay from a data source to the consumer. T_p denotes the delay of a path in the operator graph G_{cep}. The path p can be represented as $(\omega_{p_1}, \omega_{p_2}, \ldots, \omega_{p_{n_p}})$, where n_p denotes the number of operators in p. These operators are deployed on the edge nodes $(v_{p_1}, v_{p_2}, \ldots, v_{p_{n_p}})$.

For example, in Fig. 2, the end-to-end delay of $path_1$ is the worst. Thus, the response time of operator graph G_0 is $T_r(G_0)$ that equals to 120 ms. $path_1$ is (source, ω_{speed}, $\omega_{accident}$, user). In Fig. 1(a), X($path_1$) = (v_0, v_1, v_1, v_0).

For T_p, we have:

$$T_p = \sum_{i=0}^{n_p-1} d(v_{p_i}, v_{p_{i+1}}) + \sum_{i=0}^{n_p} T_E(\omega_i) + \sum_{i=0}^{n_p} T_q(\omega_i) \tag{2}$$

where $d(v_{p_i}, v_{p_{i+1}})$ denotes the transmission delay between the edge nodes v_{p_i} and $v_{p_{i+1}}$. $T_E(\omega_i)$ denotes the processing delay of events in the operator ω_i. $T_q(\omega_i)$ denotes the queuing delay of events in the operator ω_i. In addition to the above three delays, there is also the propagation delay. However, in the edge network, the propagation delay, which is too low, can be ignored.

The transmission delay $d(v_{p_i}, v_{p_{i+1}})$ can be calculated as:

$$d(v_{p_i}, v_{p_{i+1}}) = \frac{sz}{w(v_{p_i}, v_{p_{i+1}})} \tag{3}$$

where sz is the size of an event packet.

The processing delay of events in the operator ω_i can be calculated as:

$$T_E(\omega_i) = \frac{1}{\lambda_(v_{p_i})} \tag{4}$$

where the average event process rate of v_{p_i} is $\lambda(v_{p_i})$ events per second (eps).

The queuing delay mainly depends on the congestion level of the router. The event rate from v_{p_i} to $v_{p_{i+1}}$ can be represented as $r(p_i, p_{i+1})$ eps. The number of channel between edge node v_{p_i} and $v_{p_{i+1}}$ is $\mathrm{n}(v_{p_i}, v_{p_{i+1}})$. The queuing delay of an operator ω_i in a period t is calculated as:

$$T_q(\omega_i) = \begin{cases} \frac{t \times sz \times r(p_i, p_{i+1})}{w(v_{p_i}, v_{p_{i+1}})}, & \text{if } E_c(p_i) > 1 \\ \frac{E_c(p_i)}{w(v_{p_i}, v_{p_{i+1}}) - r(p_i, p_{i+1})} + \frac{1}{w(v_{p_i}, v_{p_{i+1}})}, & \text{if } E_c(p_i) <= 1 \end{cases} \tag{5}$$

$$E_c(p_i) = E_c(n(v_{p_i}, v_{p_{i+1}}), \frac{sz \times r(p_i, p_{i+1})}{w(v_{p_i}, v_{p_{i+1}})}) \tag{6}$$

$$E_c(n, u) = \frac{\frac{u^n}{n!}}{\frac{u^n}{n!} + (1 - u/n) \sum_{k}^{n-1} \frac{u^k}{k!}} \tag{7}$$

Equation 7 is known as Erlang's C formula to calculate the queuing delay [7].

If the bandwidth is larger than the event rate in the connection, the queuing time is calculated as Erlang's C formula. Otherwise, the queuing time is calculated as $\frac{N_t}{w(v_{p_i}, v_{p_{i+1}})}$, $N_t = t \times sz \times r(p_i, p_{i+1})$. N_t is the size of the data sent to the queue in a period t. To avoid frequent replacements caused by excessive changes in the input rate of operator graphs, we calculate the average event rate in the last ten seconds as $\mathrm{r}(p_i, p_{i+1})$.

Initially, when an operator ω_i has not been decided where to be deployed, the transmission delay is calculated as $\frac{sz}{\bar{w}}$, where $\bar{w}$ denotes the average transmission rate of the connections in the edge network. The execution time is calculated as $\frac{1}{\bar{\lambda}}$, where $\bar{\lambda}$ denotes the average event process rate of edge nodes.

3.4 Operator Placement Problem in Edge Computing

When the manage edge node receives a CEP service query, the operator graph is pushed into an operator-graphs-queue $\mathrm{Q} = \{G_{cep}(1), \ldots, G_{cep}(N_{cep})\}$. Optimal operator placement problem in edge computing consists in determining a suitable mapping between operators Ω and edge nodes V_{edge} to minimize the average response time of operator graphs. For every operator ω_i, we can get an edge node v_u that place ω_i on v_u, to minimize the average response time $T_r(Q)$:

$$T_r(Q) = \frac{\sum_{k=1}^{N_{cep}} T_r(G_{cep}(k))}{N_{cep}} \tag{8}$$

and to satisfy:

$$\forall v_u \in V_{edge}, c(v_u) \geq \sum_{i=1}^{|\Omega|} ((X(\omega_i) == v_u) ? c(\omega_i) : 0) \tag{9}$$

Then, we prove this optimal problem is a NP-hard problem.

Theorem 1. *Optimal operator placement problem for CEP in edge computing is an NP-hard problem.*

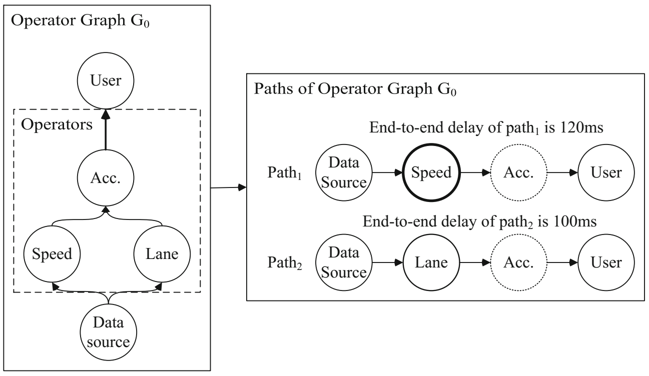


Fig. 2. Algorithm 1 preferentially deploys the speed operator.

Proof. First, we prove the decision problem of the optimal operator placement problem is NP-hard. The decision problem is stated as: N_{cep} operator graphs containing $|\Omega|$ operators and an edge network, *can the operator graphs have a feasible placement?* In the special case: the input rate of N_{cep} operator graphs are the same; the edge network has only two edge nodes, $V_{edge} = \{v_i, v_j\}$, with capacity c(v_i) = c(v_j) = $(\sum_{k=1}^{|\Omega|} c(\omega_k))/2$; w($v_i$, v_j) is infinite, $\lambda(v_i) = \lambda(v_j)$. The resulting problem is the Partition problem [3] which is known to be NP-hard. Finally, because the special case is NP-hard, the general decision problem is NP-hard as well. The optimization problem is at least as hard as the decision problem. Thus, optimal operator placement problem for CEP in edge computing is an NP-hard problem.

Thus, if P $\neq$ NP, optimal operator placement problem for CEP in edge computing cannot be solved in polynomial time. Thus, we propose an approximation-based algorithm to solve this problem.

4 Algorithm

In this section, we propose a novel algorithm that balances the response time of different paths to achieve the minimum average response time of operator graphs (Eq. 8). The basic idea is to predict the end-to-end delay of all paths, and then improve the placement of the path with the worst end-to-end delay, which can directly improve the response time of the operator graph. The algorithm deploys the operator logically closest to data source in the path on the edge node with the minimum delay.

4.1 Response Time Aware Operator Placement Algorithm

The response time of the operator graphs is calculated by Eq. 1. Algorithm 1 first calculates the response time of different paths in an operator graph, and

Algorithm 1. Response Time Aware Operator Placement

```
Input: Q = ($G_{cep}(1)$,$G_{cep}(2)$,...,$G_{cep}(N_{cep})$),$G_{edge}$
Output: X(Q)
 1: //initial parameters
 2: for all i = 1 $\rightarrow N_{cep}$ do
 3:     Get all paths p[i][] in $G_{cep}(i)$ //the operators in p is in the order of from the
        user to the data source
 4:     $N_p(i)$ = total number of paths in $G_{cep}(i)$
 5: end for
 6: X = $\emptyset$
 7: for i = 1 $\rightarrow N_{cep}$ do
 8:     for j = 1 $\rightarrow N_p(i)$ do
 9:         In[i][j] = $G_{cep}(i).Source$ ;
10:     end for
11: end for
12:
13: // make placement decision for every operator
14: while $\exists$ X($\omega \in \Omega$) == $\emptyset$ do
15:     $g_i$ = the index of the operator graph with the max response time;
16:     $p_i$ = the index of the path with the max response time in $G_{cep}$(i)
17:     $\omega_{src}$ = In[$g_i$][$p_i$] //the operator whose output is the input of $\omega_{tar}$
18:     $\omega_{tar}$ = (p[$g_i$][$p_i$]).top //the operator connecting to the output stream of $\omega_{src}$
19:     $v_{tar}$ = the edge node with the minimum delay to $\omega_{src}$
20:     if $v_{tar}$ == $\emptyset$ then //cannot find an edge node that has resources
21:         H = H + 1 ;
22:         Restart Algorithm.
23:     else
24:         X($\omega_{tar}$) = $v_{tar}$ // place $\omega_{tar}$ on $v_{tar}$
25:         In[$g_i$][$p_i$] = $\omega_{tar}$ //update input streams
26:         p.erase($\omega_{tar}$) //$\omega_{tar}$ has been deployed
27:         c($v_{tar}$) = c($v_{tar}$) - c($\omega_{tar}$) //calculate new capacity of $v_{tar}$
28:     end if
29: end while
30: return X
```

then improves the placement of the path with the largest end-to-end delay. We first find the operator graph g_i with the maximum response time (line 15) and the path p_i with the maximum end-to-end delay in gi (line 16). The end-to-end delay of the path p_i limits the response time of the operator graph g_i. Thus, we try to improve the placement of p_i. Then, we find the operator ω_{tar} whose input stream is the output stream of last deployed operator ω_{src} in the path p_i, and the edge node v_{tar} that has the minimum delay of the connection to ω_{src}. Algorithm 1 deploys the operator ω_{tar} (line 18) on the edge node v_{tar} (line 19).

For example, in the case of Fig. 1, at the moment that only the operator ω_{lane} is deployed, Fig. 2 shows how to select the operator ω_{tar} in the operator graph. Because the response time of $path_1$ is the largest response time in the paths ($path_1$ and $path_2$) and ω_{speed} is the undeployed operator connecting to the output stream of data source in $path_1$, the Algorithm 1 preferentially makes placement decision on ω_{speed}. Then, only the edge nodes v_1 and v_2 have capacity to host ω_{speed}. The algorithm first assumes that ω_{speed} is deployed on v_1 and v_2, calculates the response time of these two placements, and selects the placement with the minimum response time. Thus, ω_{speed} is deployed on the edge node v_1.

If all edge nodes within H hops have no resources, return $\emptyset$ (line 20). The hops number of candidate edge nodes should be increased (line 21). the algorithm is then restarted based on the new monitored information (line 22).

When the algorithm gets ω_{tar} and v_{tar}, the request is sent to the edge node v_{tar} for hosting the operator ω_{tar}. If v_{tar} accepts the request, system deploys the operator ω_{tar} on the edge node v_{tar} (line 24). Otherwise, the algorithm deletes the edge node and finds v_{tar} again.

The manage edge node monitors information every Δ_t second. If the response time of an operator graph exceeds the thread θ_t, the manage edge node redeploys operators to improve the response time of the operator graphs. To avoid fluctuation in the edge network transmission or the input rate of the operator graphs, this replacement judgement is performed every Δ_t second.

4.2 Time Complexity

Algorithm 1 makes placement decision for every operator. The number of operators is $|\Omega|$. For every operator graph, Algorithm 1 calculates its response time. The number of operator graphs is N_{cep}. The length of an path p_i is n_{p_i}. The time complexity of calculating the response time for an operator graph is $O(\sum_{i=1}^{N_p} n_{p_i}$. The largest number of operators in an operator graph is $|G_{cep}|$. The worst case time complexity of calculating the response time for an operator graph is $O(|G_{cep}|^2)$. The time complexity of finding the edge node with minimum delay is $O(|V_{edge}|)$. Thus, the time complexity of Algorithm 1 is $O(|\Omega| \times |G_{cep}|^2 \times N_{cep} + |\Omega| \times |V_{edge}|)$.

5 Evaluation

We conduct simulations by using Omnet++ [17] that is a network simulation tool widely used in the field of the discrete event simulation, e.g. the simulation of MCEP [9]. There is no widely accepted real data set for CEP currently. By using Omnet++, we change the input rate of the CEP services to evaluate the performance of our algorithm in different workloads. We discuss the results of the simulations in this section.

5.1 Reference Algorithms

As a reference for the results achieved by response time aware operator placement algorithm, we compare our algorithm with two baseline algorithms: greedy algorithm and load balance algorithm. Because both the load balance algorithm and the greedy algorithm do not consider the application structure, we formulate the following rule for both algorithms: the order of placing operators is determined by the shortest logical distance from the data source.

Greedy Algorithm. Greedy algorithm [14] deploys operators on the edge node closest to the input streams of the operators except the edge nodes which do not have sufficient resources.

Load Balance Algorithm. Load balance algorithm [6] calculates the average response time of every edge node, and then reduces the load of the overloaded edge nodes by redirecting some of the operators to the underloaded edge nodes.

5.2 Simulation Environment

We randomly generate 30 different topologies of edge networks and 100 operator graphs to evaluate the response time aware operator placement algorithm. We run two sets of simulation. In the first set of simulations, we run our algorithm and two baseline algorithms in the different edge networks where the same operator graphs are deployed. The initial capacity of edge nodes are 3, 4, and 5 resource units, respectively. In the second set of simulations, we run the algorithms in a network where different operator graphs are deployed. The network is the one whose simulation result is the most similar to the average result in the first set simulation. In this network, the resources capacity of the edge nodes is 5 resource units. We report the average results of these simulations.

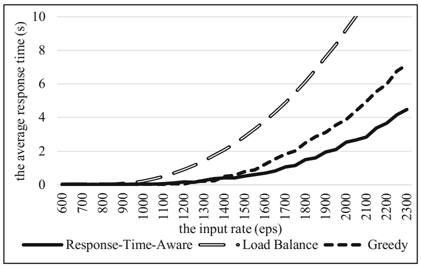
We set $\theta_t = 1\,\text{s}$ and $\Delta_t = 20\,\text{s}$. Every 20 s, if the response time of the operator graphs exceeds 1 s, the manage edge node redeploys operators to achieve lower response time.

Edge Networks. Considering the goal of this paper and the core of our proposed algorithms, i.e., deploying operators on the nearby edge nodes to keep the response time low, relatively small sized edge networks are sufficient for our evaluation. Specially, we set up three different kinds of 10-node edge networks. The first kind is a ring; the second kind is generated randomly with 15 links; and the third kind is generated randomly with 20 links. We repeat 10 times simulations on each network. In each simulation, we randomly choose a node in the edge network as the manage edge node. The transmission rate is generated randomly between 10 Mbps and 20 Mbps. The manage edge node initially monitors nearby edge nodes within 2 hops.

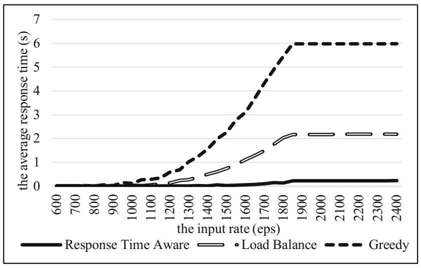
CEP Operator Graphs. To simulate different structures of operator graphs, 100 operator graphs are generated randomly, 10 of which contains 3 operators, 30 of which contains 4 operators, and 60 of which contains 5 operators. In each simulation, only one kind of operator graphs is sent to the manage edge node and the number of operator graphs is 3. For each operator, 25% of events conform to user-defined complex events sent to output streams. The resources capacity required for an operator execution is one-resource units. We gradually increase the input rate of the operator graphs to simulate a system from idle to busy.

5.3 Simulation Results

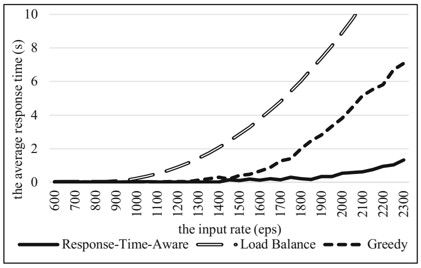
As Fig. 3(a), (b), and (c) shows, in the first set of simulation, our algorithm performs the best among the three algorithms. On average, our algorithm can reduce the total response time by 76% compared to the greedy algorithm and by 82% compared to the load balance algorithm. Because of the extra monitored information about the edge network, our algorithm adapts well to different networks.



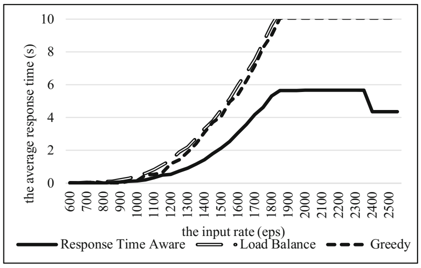
(a) The edge node capacity is 3.



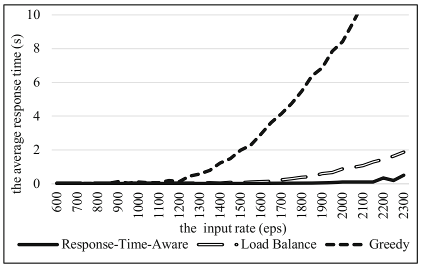
(a) An operator graph has 3 operators.



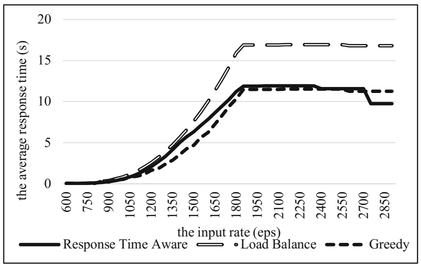
(b) The edge node capacity is 4.



(b) An operator graph has 4 operators.



(c) The edge node capacity is 5.



(c) An operator graph has 5 operators.

Fig. 3. Place the same operator graphs on the different edge networks.

Fig. 4. Place different operator graphs on the same network.

When the capacity of edge nodes is limited, load balance algorithm performs the worst (Fig. 3(a), (b)). In this case, congestion is inevitable. Load balances algorithm redirects operators to other edge nodes, which makes data have to be delivered to the operators distributed in the congested edge nodes. The extra network traffic aggravates the congestion. In contrast, our algorithm predicts the delay to the edge nodes, and deploys operators on the edge node with the lowest delay to keep the low response time. When the capacity of the edge nodes becomes sufficient, greedy algorithm performs the worst. Because the average load of edge nodes can be reduced by more resources, the load balance performs much better in this situation. Greedy algorithm makes operators gathered around the data source to reduce transmission delay. However, when the input rate increases, the congestion occurs due to large amounts of data are sent to the same edge node. Our algorithm avoids this by predicting the delay to different edge nodes and deploying operators on the edge node with low delay.

As Fig. 4(a), (b), and (c) shows, in the second set of simulation, our algorithm performs the best among the three algorithms. On average, our algorithm can reduce the response time by 33% compared to greedy algorithm and by 45% compared to load balance algorithm. We find that congestion is more likely to happen if the operator graph has more paths, because more events from different input streams are sent to an operator.

When the number of operators increases, the response time becomes higher because there are more transmission between operators. The performance of load balance algorithm becomes the worst in Fig. 4(c), because the algorithm distributes more operators than before to balance the load of edge nodes, which increases the transmission delay.

In Fig. 4(c), when the event input rate is low, our algorithm performs better than the other two algorithms. When the input rate increases, our algorithm performs worse than greedy algorithm, because the input rate increases faster than the prediction of our algorithm. However, In Fig. 4(b) and (c), when response time exceeds thread θ_t, the manage edge node updates the information about the edge network and the CEP services (including the input rate), and then redeploys the operator graphs resulting in the response time decreasing. These results show that our algorithm can adapt to the dynamic environment, improve the operator placement, and reduce the response time of CEP services.

Besides the response time, we also use bandwidth-delay to measure network usage as an additional criterion. The lower bandwidth-delay product indicates that the network load is lower [11]. Figure 5 shows the total network usage after running the three algorithms. With operator number increasing in an operator graph, due to the network transmission between different operators increases, the network usage increases. Our algorithm performs the best because the algorithm gives priority to the operator graphs with large input rate. Load balance algorithm performs the worst because the algorithm distributes the operators in the edge network resulting in more network transmission.

5.4 Threats to Validity

Construct Validity. In the simulations, we observe that our algorithm can reduce the response time of the operator graphs deployed on the edge networks. Because the problem is NP-hard, we cannot get the optimal solution in polynomial time. Our algorithm greedily deploys the selected operators on the edge

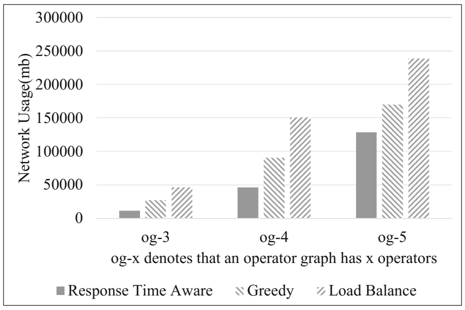


Fig. 5. Network usage versus different operator number.

node with the minimum delay, which is an approximate solution. We can reduce threats to construct validity by searching more edge nodes instead of only the edge node with the minimum delay.

External Validity. The threats to external validity come from our simulation environment. Although we gradually increase the input rate of operator graphs, dynamic network environment cannot be fully simulated.

6 Conclusions

We study response time aware operator placement for CEP in edge computing to reduce the average response time of operator graphs. Since the optimal operator placement problem is NP-hard, we present an approximation-based algorithm to ensure the response time of operator graphs low. The evaluation results show that our algorithm outperforms other approaches in the average response time of operator graphs. We plan to improve our algorithm by searching more placements and use real-world data in the experiments on actual systems in the future.

Acknowledgments. This work is supported by the National Key R&D Program of China (Grant No. 2017YFB1001801), National Natural Science Foundation of China (Grant No. 61690204, No. 61761136003), and the Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

1. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC 2012, pp. 13–16 (2012)
2. Cardellini, V., Grassi, V., Presti, F.L., Nardelli, M.: Optimal operator placement for distributed stream processing applications. In: Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, pp. 69–80 (2016)

3. Hartmanis, J.: Computers and intractability: a guide to the theory of NP-completeness (Michael R. Garey and David S. Johnson). SIAM Rev. **24**(1), 90–91 (1982)
4. Hong, K., Lillethun, D.J., Ramachandran, U., Ottenwalder, B., Koldehofe, B.: Opportunistic spatio-temporal event processing for mobile situation awareness. In: Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems, DEBS 2013, pp. 195–206 (2013)
5. Jia, M., Cao, J., Liang, W.: Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks. IEEE Trans. Cloud Comput. **5**(4), 725–737 (2017)
6. Jia, M., Liang, W., Xu, Z., Huang, M.: Cloudlet load balancing in wireless metropolitan area networks. In: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, pp. 1–9 (2016)
7. Kleinrock, L.: Queueing Systems: Theory, vol. 1. Wiley-Interscience, Hoboken (1975)
8. Liu, Y., Lee, M.J., Zheng, Y.: Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system. IEEE Trans. Mob. Comput. **15**(10), 2398–2410 (2016)
9. Ottenwalder, B., Koldehofe, B., Rothermel, K., Hong, K., Lillethun, D.J., Ramachandran, U.: MCEP: a mobility-aware complex event processing system. ACM Trans. Internet Technol. **14**(1), 6 (2014)
10. Pietzuch, P.R., Ledlie, J., Shneidman, J., Roussopoulos, M., Welsh, M., Seltzer, M.I.: Network-aware operator placement for stream-processing systems. In: 22nd International Conference on Data Engineering (ICDE 2006), p. 49 (2006)
11. Rizou, S., Durr, F., Rothermel, K.: Solving the multi-operator placement problem in large-scale operator networks. In: 2010 Proceedings of 19th International Conference on Computer Communications and Networks, pp. 1–6 (2010)
12. Satyanarayanan, M.: The emergence of edge computing. Computer **50**(1), 30–39 (2017). https://doi.org/10.1109/MC.2017.9
13. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The case for VM-based cloudlets in mobile computing. IEEE Pervasive Comput. **8**(4), 14–23 (2009)
14. Saurez, E., Hong, K., Lillethun, D., Ramachandran, U., Ottenwalder, B.: Incremental deployment and migration of geo-distributed situation awareness applications in the fog. In: Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, DEBS 2016, pp. 258–269. ACM (2016)
15. Taneja, M., Davy, A.: Resource aware placement of IoT application modules in fog-cloud computing paradigm. In: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), 8–12 May 2017, Lisbon, Portugal, pp. 1222–1228 (2017). https://doi.org/10.23919/INM.2017.7987464
16. Tziritas, N., Loukopoulos, T., Khan, S.U., Xu, C.Z., Zomaya, A.Y.: On improving constrained single and group operator placement using evictions in big data environments. IEEE Trans. Serv. Comput. **9**(5), 818–831 (2016)
17. Varga, A., Hornig, R.: An overview of the OMNET++ simulation environment, p. 60 (2008)
18. Zhang, Y., Niyato, D., Wang, P.: Offloading in mobile cloudlet systems with intermittent connectivity. IEEE Trans. Mob. Comput. **14**(12), 2516–2529 (2015)